UNIVERSITY
OF SUSSEX

Data acquisition software development and physics studies for future lepton colliders

Tom Coates

Submitted for the degree of Doctor of Philosophy

University of Sussex

June 2019

Declaration

I hereby declare that this thesis has not been and will not be submitted in whole or in part to another University for the award of any other degree.

Signature:

Tom Coates

UNIVERSITY OF SUSSEX

Tom Coates, Doctor of Philosophy

Data acquisition software development and physics studies for future lepton colliders

Summary

In this thesis, a software framework called Data Quality Monitoring for High Energy Physics (DQM4hep) is presented, intended as a generic and adaptable online monitoring and data quality monitoring framework for high-energy physics experiments and testbeams. The framework and its development and deployment is discussed, using a number of testbeams as examples. The first group of these testbeams took place within the AIDA-2020 and CALICE collaborations, using the framework on the CALICE-AHCAL prototype. Following this, the framework was also used in the IDEA combined testbeam at the CERN Super Proton Synchrotron. The result of these testbeams was proof that the framework is capable of being adapted easily to a wide variety of detector types and experiments, demonstrating that it has fulfilled the requirements of the AIDA-2020 collaboration. Following this, it was also shown that DQM4hep can be used for online analysis of the IDEA testbeam, performing a similar role to more traditional offline analysis using ROOT.

Also presented is a physics analysis as part of the detector and physics for the Compact Linear Collider collaboration (CLIDdp). The analysis was performed using the hadronic decay channel of the $e^+e^- \to tth$ process at a centre-of-mass energy of 1.4 TeV. The goal of this analysis was to obtain an updated sensitivity on the measurement of the top-Higgs Yukawa coupling at the planned Compact Linear Collider. The analysis used Monte Carlo generated physics samples and several stages of modern processing, including Pandora Particle Flow Algorithms. Combined with a similar study of the semi-leptonic decay channel, the uncertainty of the coupling measurement was found to be 3.86%.

Acknowledgements

Contents

Chapter 1

Future Colliders

Progress is not a straight line.

An Wang

In the post-LHC era, particle physics is at somewhat of an impasse. The Standard Model (SM) has held up to most experiments and observation, and its predictive power is now exhausted. There are tantalising hints at a theory beyond the Standard Model – in the form of CP-violation and dark matter – but as of yet all new ways to probe the SM for its weaknesses, to see the greater theory behind it, have yielded very little.

There are many planned investigations to attempt to identify physics Beyond the Standard Model that use the Large Hadron Collider (LHC), or plan to leverage the upgrades for the High Luminosity Large Hadron Collider (HL-LHC). However, now that the Higgs boson has been identified successfully, one of the most fruitful avenues for further research is the construction and operation of a lepton collider at the energy frontier, with sufficient centre-of-mass energy to produce Higgs bosons in large numbers.

This is what motivates the several proposals for future lepton colliders around the world today, that would operate to complement and expand the reach of particle physicists beyond what the LHC is currently capable of. These proposed lepton colliders are broadly split into two groups: linear colliders and circular colliders.

Linear colliders use two accelerator arms pointed towards a single interaction point, and in general are capable of high centre-of-mass energies. The main candidates for this style of collider are the International Linear Collider (ILC) and the Compact Linear Collider (CLIC).

Circular colliders use a similar layout to the LHC, with a circular accelerator capable of creating collisions at multiple different interaction points around the circumference of the ring, which permits multiple detectors. The downsides of a circular collider are that

lighter particles like electrons are more prone to energy losses via synchrotron radiation, limiting the centre-of-mass energies that a circular lepton collider can operate at. However, in exchange they tend to have much higher luminosities, allowing greater numbers of collisions and higher yields of certain processes or channels. The main candidates for circular colliders are the Future Circular Collider (FCC) and the Circular Electron Positron Collider (CEPC).

These proposed colliders share many of the same motivations, design considerations, features, and challenges, as does the ongoing international effort in research and development to make these colliders a reality.

1.1 The Standard Model

The Standard Model of particle physics divides nature up into two domains: particles and forces. The four fundamental forces are electromagnetism, the strong interaction, the weak interaction, and gravitation[1].

The particle domain is further split along several different axes. The most fundamental is the distinction between the leptons and the quarks, both of which are spin-half fermions. Quarks are affected by the strong interaction, whereas leptons are not.

In addition to this, there is another class of particles with integer spin called the bosons. These are the *force carriers* or *gauge bosons* – the exchange of these particles is the medium through which the fundamental forces have their effect. The electromagnetic interaction is mediated by the photon (γ), the weak interaction is mediated by the W and Z bosons, and the strong interaction is mediated by the gluon (g).

Electroweak theory describes the unification of electromagnetism with the weak interaction at high energies, and was awarded the Nobel prize in 1979. One of the important features was electroweak symmetry, which describes the electroweak interaction as an $SU(2) \times U(1)$ gauge group. This then produces three bosons from the $SU(2)$ group (W_1, W_2, W_3), and one B boson from the $SU(1)$ group. All of these are massless. The $W_{1,2,3}$ bosons correspond to the gauge bosons for the weak interaction (W^+, W^-, Z^0) and the B boson to the photon, which mediates electromagnetism.

However, experimental evidence shows that the W and Z bosons are *not* massless. The W^+ and W^- have the same mass of 80 GeV, while the Z^0 boson has a mass of 91 GeV.

[1] Gravitation is not contained within the SM, although many BSM models attempt to integrate it.

The Higgs mechanism and electroweak symmetry breaking

The solution to this problem comes in the form of the Higgs mechanism and electroweak symmetry breaking. This posits the existence of a scalar field permeating all of space – the Higgs field. The potential of the Higgs over all space is found to be in the shape of the so-called 'Mexican hat potential', with a high value at the origin, and a minimum value elsewhere. This means that the Higgs must 'roll down' the slope, settling in the lower-energy regions. This in turn causes the Higgs to have a nonzero vacuum expectation value, and then spontaneous breaking of the electroweak symmetry.

With the breaking of the electroweak symmetry, three of the Higgs' four degrees of freedom (which would normally yield Goldstone bosons) instead mix with the W^+, W^-, and Z^0 bosons, giving them mass. The remaining fourth degree of freedom emerges as *the* Higgs boson.

The Higgs boson is an important probe of physics, as it couples to all massive particles in proportion to their mass. The top quark, being the most massive elementary particle, has the strongest coupling to the Higgs boson. This makes the top quark an extremely useful way to search the Higgs sector, as the strength of its coupling means that any divergences from the SM in the Higgs sector will be most visible in the Higgs boson's interactions with the top quark.

1.2 The physics case for a lepton collider

With the observation of a Higgs boson with a mass of 125 GeV, based on data from the Large Hadron Collider, the Standard Model of particle physics is now functionally complete – all of it's major predictions have been observed. This is a testament to it's quality as a theory, where it's predictive power and accuracy is one of the best in all of the sciences.

But despite this, a multitude of observations have shown that the Standard Model cannot be a complete theory of nature. Many phenomena have been observed that the Standard Model cannot predict, or that don't seem to interact with the Standard Model in any way.

Particle physicists are now forced to seek answers to three questions that the Standard Model cannot solve:

1. What is dark matter? Astrophysics observations support the existence of a neutral, weakly-interacting substance that composes around 85% of all mass in the universe.

Yet this substance cannot be explained by any known form of matter, and is completely unprecedented by the Standard Model.

2. Why is there so little antimatter? The symmetries inherent in the Standard Model predict that the Big Bang would have created an equal quantity of matter and antimatter. Yet the universe today is dominated by matter.

3. Why does the Higgs field fill space and give mass to elementary particles? The existence of the Higgs field and the coupling of the Higgs boson to other particles can be understood from the Standard Model but their origin or cause is still unexplained.

In order to answer these questions, new theories of physics Beyond the Standard Model (BSM) have been made, and need to be experimentally tested. To do this, particle collider experiments at the energy frontier are needed. The Large Hadron Collider (LHC) has already been used extensively in searches for new physics, in the forms of new particles, rare and exotic decays, supersymmetry, and dark matter.

However, the running of a lepton collider at the energy frontier would be complementary to the LHC's continuing physics programme – there are many events or channels that are inaccessible or difficult to examine in one environment that are much simpler or higher precision in the other. In this way, a lepton collider would help to improve and refine measurements already taken at the LHC, while also allowing physicists to examine new channels and decays that were not accessible to it. A number of processes and their discovery potential can be seen in Table 1.1.

This follows a historical pattern in particle physics – as the energy frontier advances, hadron colliders are used to discover new physics and new phenomena, followed by lepton colliders to examine these phenomena in higher precision.

1.2.1 Higgs physics

Of specific interest to searches at lepton colliders would be the Higgs boson itself. Many BSM models predict differences from the Standard Model in the Higgs sector – such as several Higgs bosons with different masses, composite Higgs, charged Higgs etc. The comparatively 'quiet' environment of a lepton collider allows higher precision measurements of the properties of the Higgs boson, placing better constraints on the presence of new physics. In addition, lepton colliders can easily operate at specific thresholds and "hot spots" for Higgs production, permitting a much greater number of events yielding Higgs bosons, and thus a greater sample to examine.

Additionally, the most common decay of the Higgs boson, at a branching ratio of 57.7%, is the $H \to b\bar{b}$ process. Despite the high branching ratio, the huge QCD backgrounds in a hadron collider have made this decay incredibly difficult to observe at the LHC – in fact, the $H \to b\bar{b}$ decay has only fairly recently been experimentally confirmed by the ATLAS experiment. However, in a lepton collider these QCD backgrounds are significantly smaller, meaning this decay channel is much easier to analyse, opening up a huge number of Higgs events for analysis and examination.

Another highly interesting process uniquely accessible to lepton colliders is the higgstrahulung reaction: $e^+e^- \to Zh$. The decay of the Z boson into lepton pairs e^+e^- or $\mu^+\mu^-$ allows high precision kinematic measurements of the process without directly measuring the Higgs boson itself. This allows unprecedented measurement of the Higgs mass, while also making measurements of missing energy possible. If the Higgs boson has invisible decays – such as dark matter particles or other undiscovered particles that don't couple to the SM – the higgstrahlung process allows their existence to be identified.

An additional benefit of the higgstrahlung process is that the recoiling Z boson can be used to identify all the Higgs decay modes, and thus direct measurement of Higgs couplings. With this understanding of the Higgs couplings and their rates and branching ratios, it is possible to perform a model-independent determination of the total rate of Higgs decay Γ_h. This result then allows calculation of the absolute size of all other Higgs couplings.

In addition, the production of top quark pairs in combination with a Higgs boson will allow direct measurement of the top-Higgs Yukawa coupling. This is the strongest Higgs coupling in the SM, and as such is most sensitive to new physics. A study examining the top-Higgs Yukawa coupling in detail can be found in Chapter 5 of this thesis.

1.2.2 Top physics

In addition to the Higgs, the top quark is of high interest to possible physics programmes at a future lepton collider. Since it's discovery at Fermilab in 1995, there have been no lepton colliders operating with sufficient energy to produce top quarks – unlike the charm and bottom quarks, which were studied in further detail in lepton colliders after their discovery in hadron machines. The production threshold for top quarks is 350 GeV, and no lepton colliders capable of this energy have been constructed, meaning that the advantages of a lepton collider have never been utilised to understand the top quark in greater detail.

The detailed study of top quarks and their properties that would be made possible by a lepton collider would have many benefits, primarily in the precision of the measurements. Similar to the Higgs, low QCD backgrounds make top quark events easier to reconstruct. The fact that the collision is between elementary particles, rather than a parton-parton collision as in a hadron collider, also permits are more well-defined centre of mass energy, which is not possible in hadron colliders.

Precision measurement of the top quark is an important test of the SM – many BSM models envision their additional particles as partners of the top quark. The upshot of this is that because of this, these theories propose properties of the top quark that diverge from the SM. In-depth and precise analysis of the top quark's properties would place limits on these theories.

1.2.3 Supersymmetry

Supersymmetry (SUSY) is one of several candidate models for BSM physics, and currently one of the most widely-researched and well-motivated. A large programme of searches for supersymmetric particles or signals at the LHC is ongoing, though as of writing no new particles or signals of supersymmetry have been confirmed.

For the same reasons that lepton colliders provide ideal environments for doing precision Higgs and top physics, they also provide ideal conditions for supersymmetry searches. One example is searches for the first- and second-generation sleptons $\tilde{e}$ and $\tilde{\mu}$ that decay by the process

$$e^+e^- \to \tilde{e_R^+}\tilde{e_R^-} \to e^+e^-\tilde{\chi_0^1}\tilde{\chi_0^1}$$

$$e^+e^- \to \tilde{\mu_R^+}\tilde{\mu_R^-} \to \mu^+\mu^-\tilde{\chi_0^1}\tilde{\chi_0^1}$$

This requires high-efficiency reconstruction of leptons, as well as the usage of Boosted Decision Trees (BDT) to distinguish the process from SM processes and SUSY backgrounds.

A variety of similar processes involving supersymmetric particles would be available to study at a high-energy lepton collider, using the higher-precision detectors and lower backgrounds to improve current limits. Precise measurement of these processes and their cross-sections, as well as the masses of any resulting particles, will help to provide evidence for or put limits on supersymmetric theories.

1.3 The International Linear Collider

The International Linear Collider (ILC) is a proposed high-luminosity linear electron-positron collider based upon 1.3 GHz superconducting radio frequency (SCRF) accelerating technology. The ILC would have a centre of mass energy of 250 GeV in the initial stage, upgradable to 500 GeV and then to 1 TeV at a later date. The total footprint of the complex would be 31km in length, with the arms using magnets with an accelerating gradient of 31.5 MVm^{-1} in metre-long superconducting nine-cell niobium cavities operating at 2K.

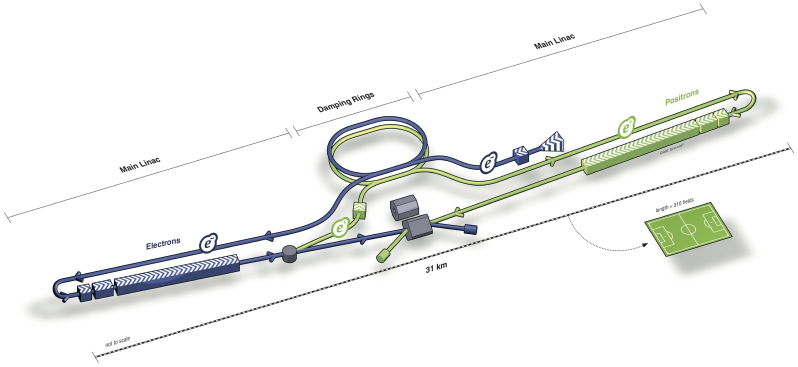

Figure 1.1: View of the accelerator complex for the International Linear Collider, showing the two linacs and storage and damping rings, with football field for scale.

One of the unique features of the ILC is the "push-pull" detector system. This is a moving platform in the chamber housing the interaction point (IP), upon which two detectors can be mounted. The platform can be moved to exchange which detector is in the interaction point, allowing a linear collider to function with multiple detectors. This allows the two detectors to specialise for different physics studies and goals, much like the various experiments at the LHC at CERN, which would normally not be possible with a linear collider.

There were a number of proposed sites for the ILC, including Fermilab in the United States, CERN in Geneva, DESY in Hamburg, and JINR near Moscow. The most recent possible site, which has had significant attention and planning devoted to it, is the Kitakami Highlands region of Iwate prefecture in Japan. This would be a greenfield site, located on the side of a mountain range and requiring that a significant amount of the facility's infrastructure be located underground, in tunnels dug within the granite rock of the region.

The ILC project published a Technical Design Report (TDR) in [20XX?] with ex-

tensive details of the technology and ongoing research and development into making the collider a reality. Current timeline estimates say that construction could start in [year], commissioning of the experiment would begin in [year], and first physics would be achieved in [year].

A report from the Science Council of Japan (a representative organisation of the Japanese science community) released in early 2019 expressed that they had not reached a consensus as to whether to support hosting the ILC in Japan. Some of the reasons cited were concerns over international cost-sharing in the long-term, as well as whether the expected scientific outcomes would justify the unprecedented human resource requirements and infrastructure necessary to make the ILC a reality [1].

On 7th March 2019, the Japanese government released a statement to say that they would not be making a proposal to host the collider. They however expressed a keen interest in the future of the project, and that they would be continuing to contribute towards the research and development of the ILC.

1.3.1 ILC detectors

Detector design for the ILC is driven by the requirements of the physics programme – many of the physics goals and targeted processes are highly dependent upon hadronic states, so precise jet reconstruction and high jet energy resolution is critical to meeting the expectations placed upon the ILC.

The only technique thought to be capable of delivering the necessary level of accuracy is *particle flow*. Particle flow is an integrated approach which uses algorithms to examine the flow of energy through all parts of the detector to correlate different signals together to generate particle flow objects (PFOs). The requirements for the use of particle flow is a very good separation of charged and neutral particles, translating into a need for high-efficiency trackers, and calorimeters capable of high-efficiency reconstruction of neutral particles. The design of the detectors for the ILC has proceeded from these requirements.

The need for large separations between charged and neutral particles requires that the vertex detector, tracker, and calorimeter systems are contained within a magnetic field, so that charged particles' path is curved by the field. In general, this separation depends on the physical size of the detector and the strength of the magnetic field. Thus there are essentially two basic approaches to detector design: a large detector with a lower magnetic field, using the size of the detector to separate charged particles from neutral particles; or a more compact detector utilising a much stronger magnetic field to create the same

separation.

These two approaches are shown in the two detector concepts for the ILC – the International Large Detector using a 3.5T magnetic field, and the more compact Silicon Detector using a much stronger 5T magnetic field. These two detectors will be discussed in detail below.

The International Large Detector (ILD)

The International Large Detector (ILD) is a detector concept for the ILC intended as a multi-purpose detector, with a strong focus on optimising the performance of particle flow algorithms as much as possible. To attain this, it uses several technologies for very high-resolution and high-efficiency tracking, as well as highly-granular calorimeters.

For vertex tracking, the ILD uses three double-layers of pixel detectors using monolithic active pixel sensor (MAPS) technology, with a spatial resolution of 4 μm and a timing resolution of 2-4 μs. For tracking, the ILD uses a hybrid system, combining a gaseous time projection chamber (TPC) with silicon detector layers placed both inside and outside the TPC volume. This combination allows a high tracking efficiency with a low material usage.

The calorimeter system must be highly granular in order to best utilise particle flow. The calorimeters are split into the electromagnetic calorimeter (ECAL) and hadronic calorimeter (HCAL). The ECAL utilises a silicon diode sampling calorimeter with diode pads of 5×5 mm^2. An option for an ECAL using thin scintillator strips is also being investigated. The HCAL has two possible options available. The Analogue Hadronic Calorimeter (AHCAL) uses silicon photomultipliers (SiPM) on tiles of plastic scintillator with a resolution of 3×3 cm^2 using a fully analogue readout. The Semi-Digital Hadronic Calorimeter (SDHCAL) uses resistive plate chambers (RPC) with a higher granularity of 1×1 cm^2, but does not use a fully analogue output, meaning that amplitude information is more limited.

These detectors are placed within a solenoid capable of generating a 3.5T magnetic field, and then within an iron flux return yoke which is instrumented for muon identification and tail catching, as well as providing structural support for the detector. The finished ILD is expected to weigh 14,000 metric tonnes.

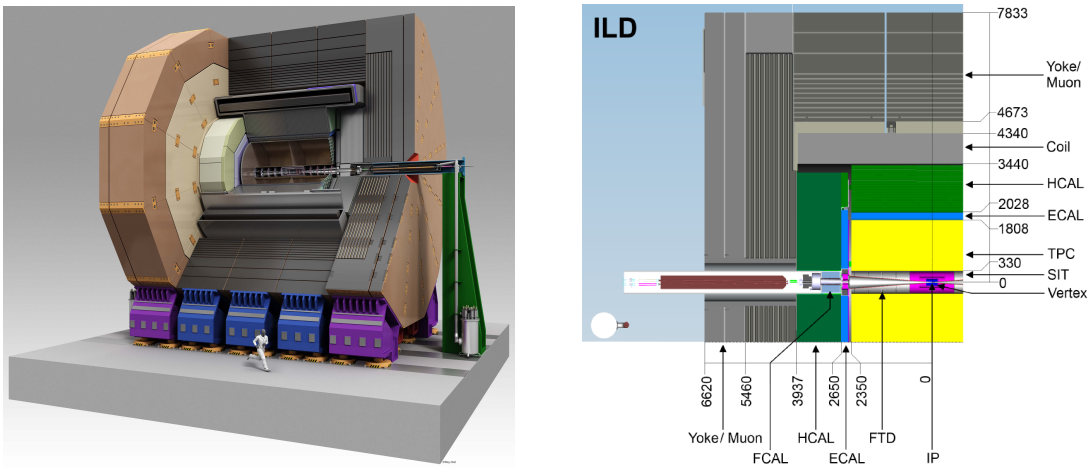

Figure 1.2: Rendering of the finished ILD cutaway to show the internal features (left); and a quadrant view of the ILD components (right).

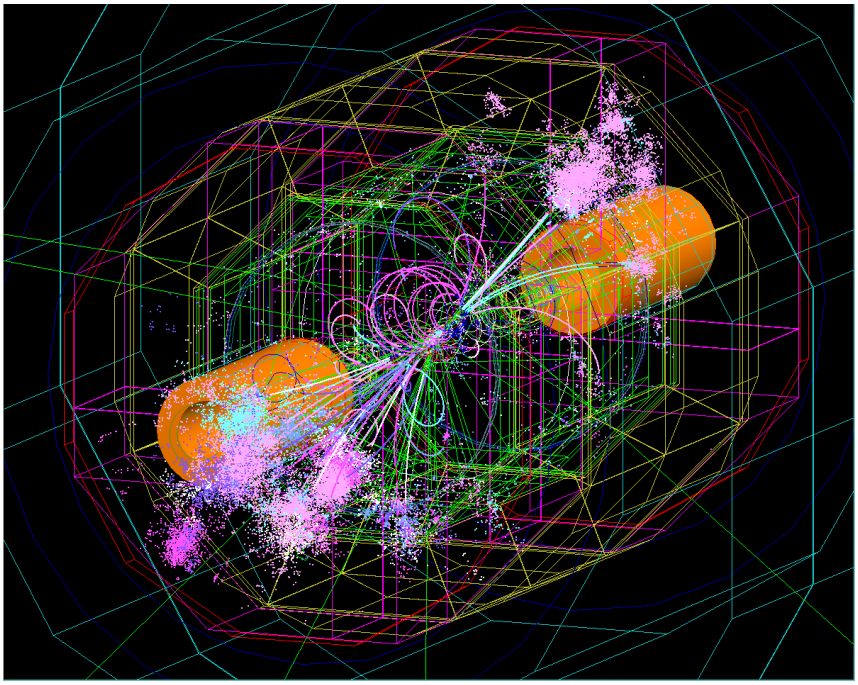

Figure 1.3: Visualisation of a simulated $e^+e^- \to t\bar{t}h$ event in the ILD. Charged particles can be easily identified by the curved or spiral paths they take within the magnetic field, and the jets are visible as the light pink and purple areas near the beampipes on either side.

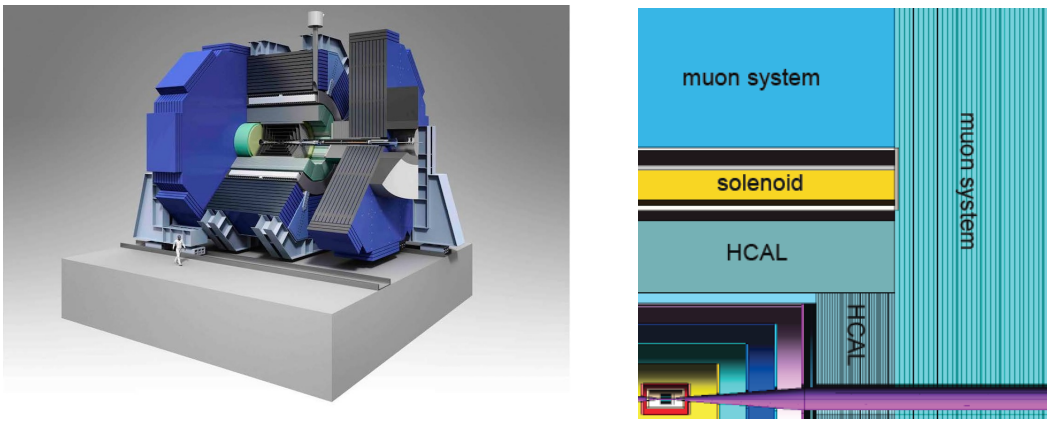

Figure 1.4: Isometric view of the finished SiD cutaway to show the interal features (left); and a quadrant view of the SiD components (right).

The Silicon Detector (SiD)

The Silicon Detector (SiD) is a detector concept for the ILC that uses primarily silicon-based technology, with the aim to reduce cost while still maintaining high performance and attaining the ILC's physics goals. The SiD is also more compact than the ILD, utilising a stronger magnetic field to compensate.

The vertex and tracker systems are both composed of silicon sensors, using a cylindrical configuration. The vertex uses silicon pixel sensors while the tracker uses silicon strip sensors, both designed to be used with power pulsing – the electronics are only powered and active when it is known that bunches will be colliding. This reduces power and cooling requirements.

The high-granularity calorimeters are both nested within the barrel, inside the magnetic field. The ECAL uses thirty alternating layers of tungsten absorber and silicon active layers, in 3.5×3.5 mm^2 hexagonal pixels. The HCAL uses alternating layers of steel absorber and a glass resistive plate chamber, with cells of 10×10 mm^2

Outside of the calorimeter, the superconducting solenoid generates a 5T magnetic field, which enables the more compact detector design – a higher magnetic field increases the spatial separation between charged and neutral particles, which is necessary for usage of particle flow algorithms.

Outside of the magnetic field is an iron flux return yoke, which similarly to the ILD concept also acts as a structural support and is instrumented for muon identification and tail catching.

1.4 The Compact Linear Collider

The Compact Linear Collider (CLIC) is a proposed linear electron-positron collider that would be located at CERN in Geneva, Switzerland. The accelerator is a staged design, with the initial stage having a centre of mass energy of 380 GeV, focusing on precision measurements of top quark and Higgs physics. The further stages would increase the centre-of-mass energy to 1.5 TeV, then finally 3 GeV. Physics goals in these later stages would involve searches for new physics processes, as well as precision measurements of rare Higgs processes, and of new states discovered at the LHC or earlier stages of CLIC.

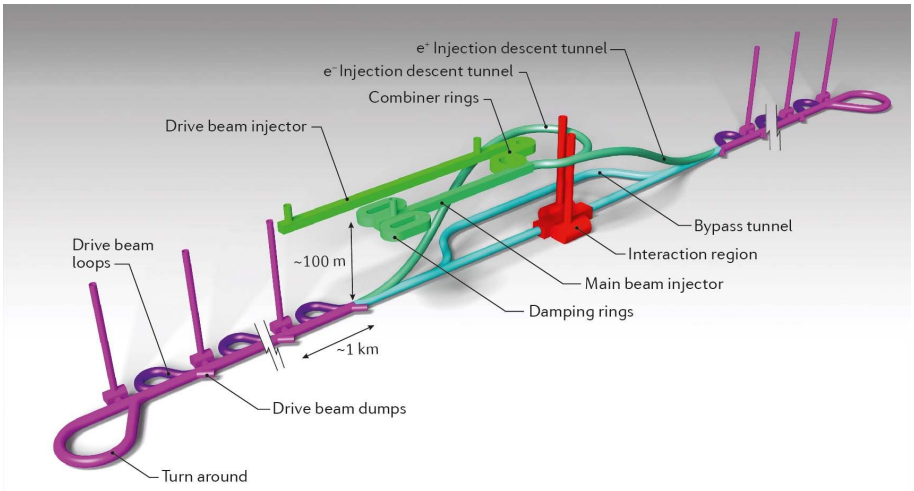

Figure 1.5: View of the accelerator complex for the Compact Linear Collider, showing the main beams and drive beams, and the interaction region.

To attain these extremely high energies, the CLIC accelerators will be able to produce accelerating gradients as high as 100 MVm^{-1}. These cannot be achieved with traditional klystrons due to the extremely high peak RF power needed. Klystrons capable of providing this would be inefficient and prohibitively expensive, so the CLIC accelerators plan to use a two-beam acceleration scheme. A drive beam is used, with low intensity but high power. RF power is then extracted from the drive beam to power the main accelerator. In this way, the high power, shorter duration pulses can be achieved more economically, allowing the high acceleration gradients that will allow each arm of CLIC to accelerate electrons up to 1.5 TeV in only 21km.

The proposed site for the CLIC experiment would be at CERN, built beneath the existing LHC ring and stretching across the French-Swiss border, running parallel to the feet of the Jura mountain range. This placement is determined by the geological features of the region around Geneva and the feet of the Juras, where the tunnels would be dug

into the sedimentary 'molasse' rock in the area.

As of writing, the CLIC project released a Conceptual Design Report (CDR) in 2010 detailing the current state of planning for the experiment. It has also submitted input to the European Particle Physics Strategy Update, which will decide which projects the CERN collaboration chooses to pursue from 2020 onwards.

The detectors envisioned for CLIC are similar in design to those for the ILC, and are usually referred to as CLIC_ILD and CLIC_SiD. See 1.3.1 for a detailed description of these detector concepts.

1.5 The Future Circular Collider

The Future Circular Collider (FCC) is a series of concepts for a future collider that would be located in the Geneva area near the existing LHC ring. The FCC project as a whole has three different accelerator concepts – the FCC-hh for proton-proton and ion-ion collisions, the FCC-ee for electron-positron collisions, and the FCC-he for electron-proton collisions.

The initial proposal is to construct a circular electron-positron collider – the FCC-ee – with a circumference of 100km and delivering a maximum centre-of-mass energy of 365 GeV. The motivation for this is that at this energy range – the electroweak scale – the FCC would be able to access the Z pole, the W- and top-pair production thresholds, as well as producing a large number of Higgs bosons.

Unlike linear colliders, staged energy increases are not part of the FCC-ee plan. The usage of low-mass particles like electrons in a circular collider results in a high energy loss due to synchrotron radiation, which must be mitigated by the constant addition of energy. Significantly higher energies than those planned by the FCC-ee are not currently practical due to these losses. Instead, the FCC-ee would concentrate on a physics programme in the 90-365 GeV energy range, leveraging the much higher luminosities available to a circular collider compared to linear colliders in the same energy range.

A further part of the proposal for the FCC is that following the conclusion of the physics programme of the FCC-ee, the tunnels and infrastructure would be re-used for the FCC-hh, a hadron collider. This follows in the footsteps of the LHC, which was constructed in tunnels originally built to house the Large Electron Positron Collider (LEP). It is claimed that the FCC-hh built in these tunnels would be able to reach centre-of-mass energies of at least 100 TeV.

According to the given timeline, the FCC-ee would begin construction in 2028, and

first physics would take place in 2039.

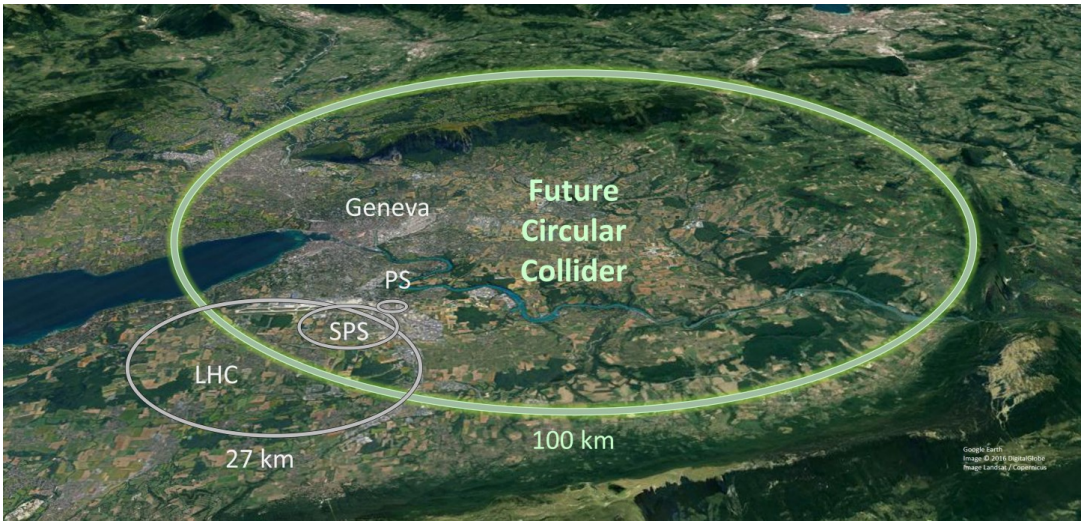

Figure 1.6: The ring of the proposed Future Circular Collider laid over satellite imagery of the region around the French-Swiss border at Geneva. The current accelerators are shown in grey for scale.

1.6 The Circular Electron Positron Collider

The Circular Electron Positron Collider (CEPC) is a proposal for a circular electron-positron collider with a circumference of 100km that would be hosted in China, with a centre of mass energy of 240 GeV to operate as a Higgs factory. It is also intended to operate at 91 GeV and 160 GeV to produce large numbers of Z and W bosons. This lower centre of mass energy is due to considerations of energy loss via synchrotron radiation. However, the CEPC collaboration intends to utilise these to produce several synchrotron radiation light sources, including two gamma-ray beamlines.

The CEPC project released a Conceptual Design Report (CDR) in 2018 detailing the current state of planning for the experiment. It has also been submitted as input to the European Particle Physics Strategy Update. The current timeline envisions a five-year research and development period from 2018-2022, and construction to start in 2022, completing in 2030. The physics programme is expected to last for ten years, concluding in 2040.

Similarly to the FCC, it is expected that technology for superconducting magnets at high field strengths will have developed sufficiently to allow the tunnels to be used to house a high-energy hadron collider, called the Super Proton Proton Collider (SPPC).

The detectors of the CEPC as outlined in the CDR are very similar in design to the

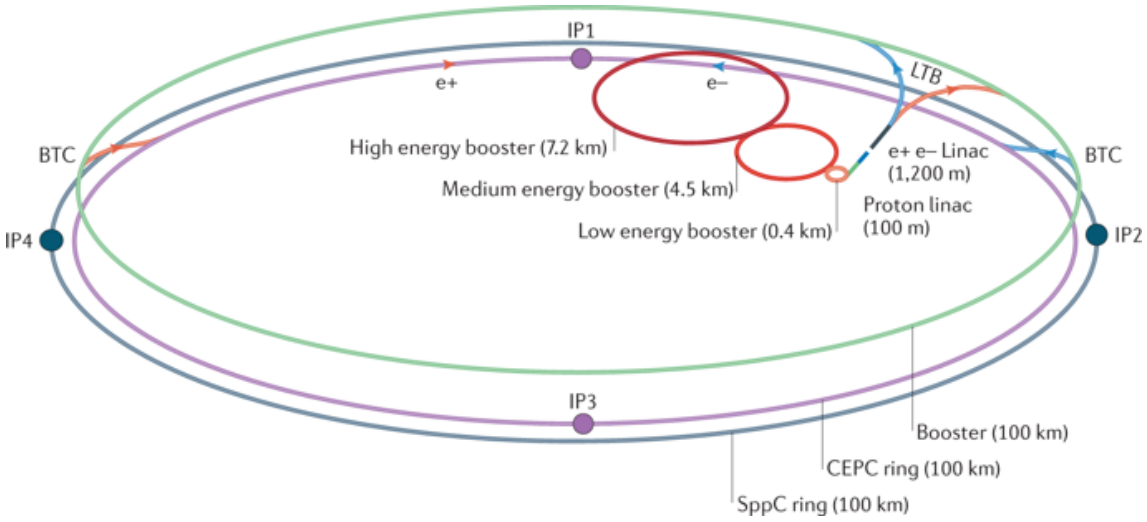

Figure 1.7: Diagram showing the tunnels of the accelerator complex for the proposed Circular Electron Positron Collider, with the multiple booster rings and linacs.

ILD, in the design of using a silicon vertex tracker, a TPC tracker, and similar options for the calorimeters. An additional difference is the optioning of a dual-readout calorimeter to replace the electromagnetic and hadronic calorimeters.

There would be two interaction points around the ring, allowing both detectors to operate at the same time, as opposed to the ILC's push-pull system. More information on the proposed CEPC detectors can be found in [reference].

Energy	**Reaction**	**Physics goal**
91 GeV	$e^+e^- \to Z$	ultra-precision electroweak
160 GeV	$e^+e^- \to WW$	ultra-precision W mass
250 GeV	$e^+e^- \to Zh$	precision Higgs couplings
350-500 GeV	$e^+e^- \to t\bar{t}$	top quark mass and coupling
	$e^+e^- \to WW$	precision W coupling
	$e^+e^- \to \nu\bar{\nu}h$	precision Higgs couplings
500 GeV	$e^+e^- \to f\bar{f}$	precision search for Z'
	$e^+e^- \to t\bar{t}h$	top-Higgs Yukawa coupling
	$e^+e^- \to Zhh$	Higgs self-coupling
	$e^+e^- \to \widetilde{\chi}\widetilde{\chi}$	supersymmetry searches
	$e^+e^- \to AH, H^+, H^-$	extended Higgs states
700-1000 GeV	$e^+e^- \to \nu\bar{\nu}hh$	Higgs self-coupling
	$e^+e^- \to \nu\bar{\nu}VV$	composite Higgs
	$e^+e^- \to \nu\bar{\nu}t\bar{t}$	composite Higgs and top
	$e^+e^- \to \tilde{t}\tilde{t}^*$	supersymmetry searches

Table 1.1: Physics processes of interest at lepton colliders up to 1 TeV.

Chapter 2

Data acquisition software

Before software can be reusable
it first has to be usable.

Ralph Johnson

[...]

2.1 Introduction

Data Acquisition (DAQ) is a critical component of all modern particle physics experiments across all stages of technological readiness, from the very beginning of hardware testing in tabletop experiments to full-scale international experiments like the Large Hadron Collider.

In the modern era of particle physics, the interplay of hardware and software at minuscule timescales drives everything, and almost all results are highly dependent upon the speed and efficiency of the electronics and computer systems that extract data from the detectors. A massive quantity of work goes into creating, testing and optimising the systems that will acquire, process, sort and transport data before it is ever seen by the physicist operating the experiment.

Of particular interest in this thesis is the data acquisition software during the development phase, where individual detector subcomponents are undergoing prototyping and testing. These development and iteration cycles are tied closely to testbeam facilities such as the Super Proton Synchrotron (SPS) at CERN and the DESY II synchrotron at DESY. At this point in the development cycle, the detectors are beginning to take shape and this is where data acquisition (or DAQ) becomes an important consideration.

In addition to this, the data acquisition solutions used during the testbeam phase

of detector development is likely to inform the final data acquisition solution; either by evolving directly into the final software, or by identifying and evaluating the particular features or challenges of the subdetector components that the software must take into account or accommodate.

During this stage, each individual detector component – such as a vertex tracker or hadronic calorimeter – will be developed by small teams, and the natural tendency is for each of these groups to set their own standards and develop their own tools, prioritising the features that are important to their specific case. However, in the past this approach has generated a variety of *ad hoc* solutions for testbeam software, many of which cannot be applied outside of their original scope. This results in different teams solving the same problems and implementing the same solutions for each subdetector.

An alternative to this is to develop a suite of tools or frameworks that are generic – capable of being used and deployed for a wide variety of different uses and detector types. In this way, we could greatly reduce the effort used to recreate the same solutions for each new detector, allowing more science to be done faster.

2.1.1 Online monitoring and data quality monitoring

The area of this that we have chosen to contribute to is the development of online monitoring and data quality monitoring tools. Data from testbeams is often not processed fully until well after it has been taken, sometimes after the testbeam has ended, due to constraints on time or processing power. If there were errors, incongruencies, or any other issues with the data, these issues cannot be identified immediately, and as a result may be present in multiple runs, spoiling data and wasting precious time during the already extremely time-sensitive environment of a testbeam.

Online monitoring addresses these issues by allowing the experimenters to see a "preview" of the data being collected. It can provide both a quantitative and qualitative look into how the detectors are responding, and what the data will look like when properly processed, allowing any potential issues to be identified and fixed in a timely manner. This means that good online monitoring can improve the efficiency of a testbeam, increasing the "yield" of data from a given experiment.

Another aspect of this is data quality monitoring (DQM), which assesses the 'quality' of the data being taken. The definition of data's 'quality' will vary depending on the hardware, software, and goals of the experiment, but will usually constitute a some from of statistical measure. In this regard, data quality monitoring can be seen as an extension

of the concept of online monitoring, focusing more closely on the quantitative aspects. In general, DQM provides the most benefit in more mature experiments, relying on previous experience with the detector and collected data to understand how the data appears when the device is functioning correctly.

In pursuit of all of the above aims, this chapter will discuss the Data Quality Monitoring for High-Energy Physics tool (DQM4hep) developed for online monitoring and data quality monitoring, going into detail on its properties, principles, applications and development. Deployment and usage of the framework will be discussed in greater detail in Chapters 3 and 4.

2.1.2 The AIDA-2020 project

This work on DQM4hep takes place within the context of the AIDA-2020 project, an EU-funded research programme for developing infrastructure and technologies for particle physics detector development and testing, comprising 24 member countries and lead by CERN.

The overarching goal of AIDA-2020 is to develop common tools and infrastructures for physics testbeams. The collaboration is split into work packages focusing on specific areas, and the work detailed in this thesis takes place within Work Package 5 for data acquisition systems for beam tests

The goal of this work package is to create a suite of tools that are designed with a variety of possible uses in mind, thereby reducing the work and development time necessary to implement data acquisition and monitoring setups, speeding up the planning and deployment of physics testbeams.

2.2 Overview of DQM4hep

Data Quality Monitoring for High-Energy Physics (abbreviated DQM4hep) is an online monitoring and data quality monitoring framework developed for physics testbeams for high-energy and particle physics, developed by Rémi Eté and Antoine Pingault. It is designed to be able to fulfil the requirements of monitoring for physics testbeams in a generic way. The framework is written in the C++11 standard and can run on any Linux distribution. The only requirements for installation are a compiler compliant with the C++11 standard, cmake 3.4 or higher, and ROOT 6. All other dependencies are downloaded and compiled automatically during installation.

The two core principles of DQM4hep are genericness and modularity. The framework

is based upon a plugin system that allows shared libraries to be loaded and hook classes for further use [2]. This structure allows for independent components of the framework to be used, not used, or exchanged, by isolating each function of the program into independent processes. The components that are specific to any particular use case are written by the users, and the rest of the framework then handles packaging this information in a useful way and networking to transmit it to where it is needed, meaning that the user does not have to worry about the mechanics of data storage, serialisation or transmission.

The experiment-specific components have to be written by the user, but these components use standard C++ code with a few DQM4hep-specific functions to handle their integration into the framework, making them easy to understand for users who already have experience coding in C++. This also means that the framework is capable of working with any data format that can packed into, decoded from, and accessed with normal C++ methods, including those that can be loaded from external libraries. This results in a framework that is able to deal with any kind of data, including user-defined data types, making it more flexible, portable and easily reusable.

2.2.1 Architecture

DQM4hep is designed with genericness as its core paradigm, using processes and algorithms that are independent of data type. The ability to run multiple instances of each process of the framework is also key to its flexibility, allowing users to, for example, separate sub-detector data from data that has undergone event building, operate in online or offline modes, or distribute the computational load of the analysis over several networked computers.

The generic nature of the framework lies in two core features:

Firstly, the abstract Event Data Model (EDM). An event data model is the structure of the events in the data. For instance, a software 'event' might in fact be a readout cycle, where the detector writes data to the data acquisition device when its memory is full. Or it might be a physical event, in which case the event is further defined by the trigger – an event that is triggered internally by the detector itself will have a different structure to an event that is externally triggered, such as by a bunch-crossing ID. DQM4hep uses an abstract container for the event itself and allows the user to define its type, structure, and how serialisation should be handled. This means it can handle *any* type of data.

Second, the plugin system. This is a system that allows the inclusion of any user-defined classes or methods via external libraries. These can be used to specify the serial-

isation method for data, the procedure for online analysis, or many other purposes.

The plugin system for end-users consists of four different types of plugins: analysis modules, standalone modules, file reader plugins, and file streamer plugins. Each of these will be discussed in-depth in the sections below.

A diagram of the overall structure of the framework can be seen in Fig. 2.1.

Analysis modules

Analysis modules receive events from the data acquisition system, processing the data according to a user-specified procedure to create ROOT TObjects like histograms, graphs, plots, etc. The analysis module then handles encapsulating these objects as monitor elements, and sending them to the rest of the framework for display and storage.

An analysis module is specific to one use case, and is intended to be written by the user with their data format and processing needs in mind. However, the framework provides both templates and examples for how to write an analysis module.

An example of the structure of the framework utilising an analysis module can be seen in Fig. 2.2.

Standalone modules

Standalone modules are identical in form to analysis modules described above. The distinction is that a standalone module does not operate on data coming from the data acquisition device. One of the intended and most common usages of standalone modules is as a slow control, taking data from monitoring sensors on the device rather than data, to report on the condition of the hardware. Standalone modules could also be used to *generate* data, if needed, acting as a programmed signal generator or random number generator.

An example of the structure of the framework utilising a standalone module can be seen in Fig. 2.3.

File reader plugins

A file reader is a type of plugin that reads a file from the disk and packs it into a data structure necessary for usage within DQM4hep. They are used primarily for offline monitoring or data processing. File readers can be made for any kind of file, provided the user understands the data structure. There are existing examples of file readers for data stored as binary, plain text, LCIO files, and ROOT TTrees.

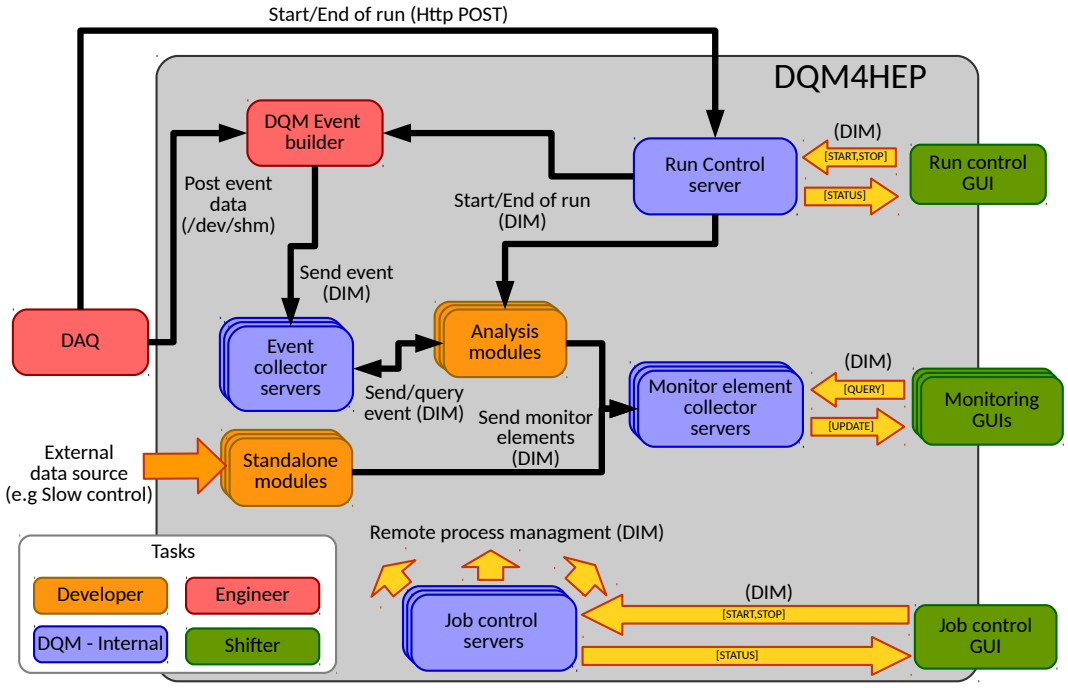

Figure 2.1: The global online architecture of DQM4hep. Each block is colour-coded to show which operator of the testbeam is responsible for the process.

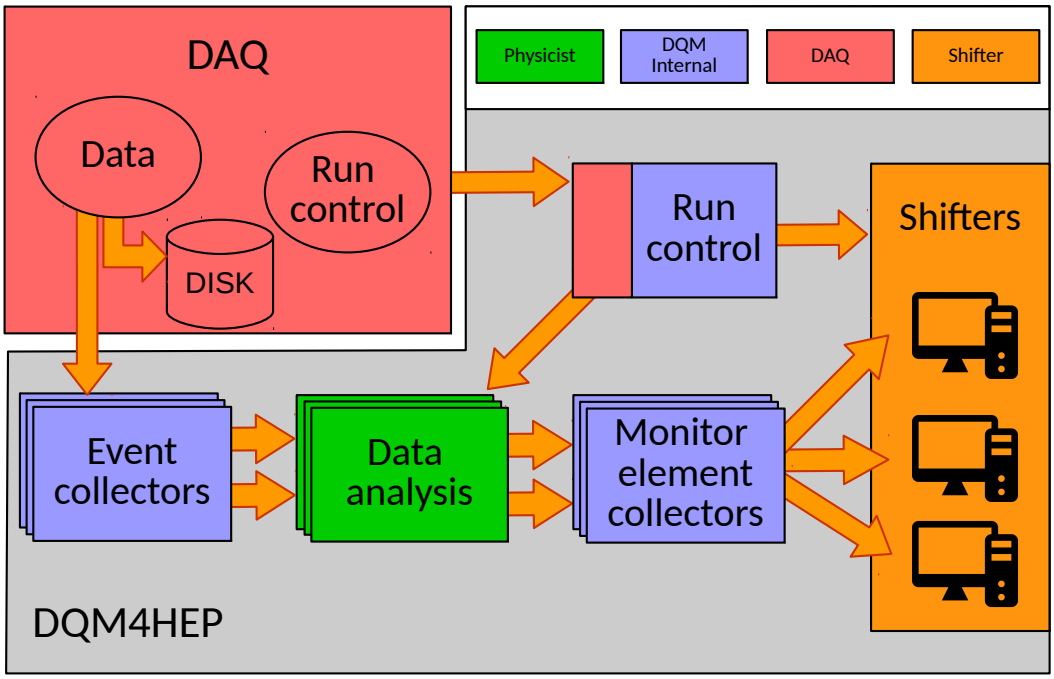

Figure 2.2: The structure of running DQM4hep online using an analysis module.

An example of the structure of the framework utilising a file reader plugin can be seen in Fig. 2.4.

File streamer plugins

A file streamer is a type of plugin that reads data from a stream and packs it into a data structure necessary for usage within DQM4hep. They are for receiving data from a data acquisition device for online monitoring. File streamers can be made for any kind of data stream, provided the user understands the data structure. File streamers are considered the "default" in DQM4hep.

2.2.2 Visualisation and graphical user interface

As of writing, the graphic user interface (GUI) and visualisation elements of the framework are still under active development for a new version. Therefore this topic will be split into two sections: one to describe the existing GUI, and one to discuss the motivations and goals for the new GUI under development.

Current GUI and visualisation

The current version of the GUI is built with Qt, a free and open-source toolkit and framework for creating graphic user interfaces and widgets that are independent of the operating system. The motivation for choosing Qt was that ROOT provides an option for integration between ROOT and Qt, allowing ROOT classes like `TCanvas` to be "embedded" into Qt widgets. This simplified the implementation of a GUI, allowing a graphical interface based on Qt to be written, then graphics from ROOT simply opened within the existing widgets and windows.

This interface is used in multiple places, including the run control process and the monitoring GUI. The monitoring GUI is built on a system of canvases. Each canvas can have multiple plots open, which can be resized, maximised, minimised, etc. and manipulated as normal for ROOT plots. The user can also create new canvases for more space to arrange plots.

In addition to this, there is an optional provision for a monitoring steering file, which contains presets of canvases, and the plots displayed on them. This is extremely useful when dealing with large datasets or large numbers of plots, as the plots required by the user can be opened automatically when the monitoring interface is run.

An example of the Qt-based monitoring GUI in use can 3be seen in Fig. 2.5.

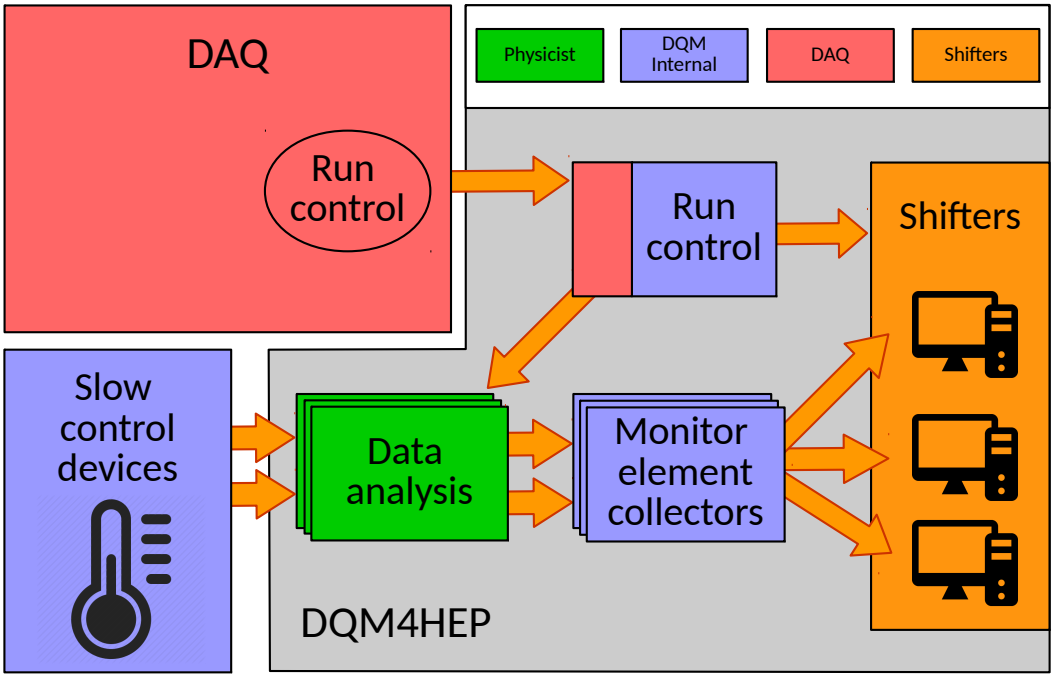

Figure 2.3: The structure of running DQM4hep online using a stand alone module.

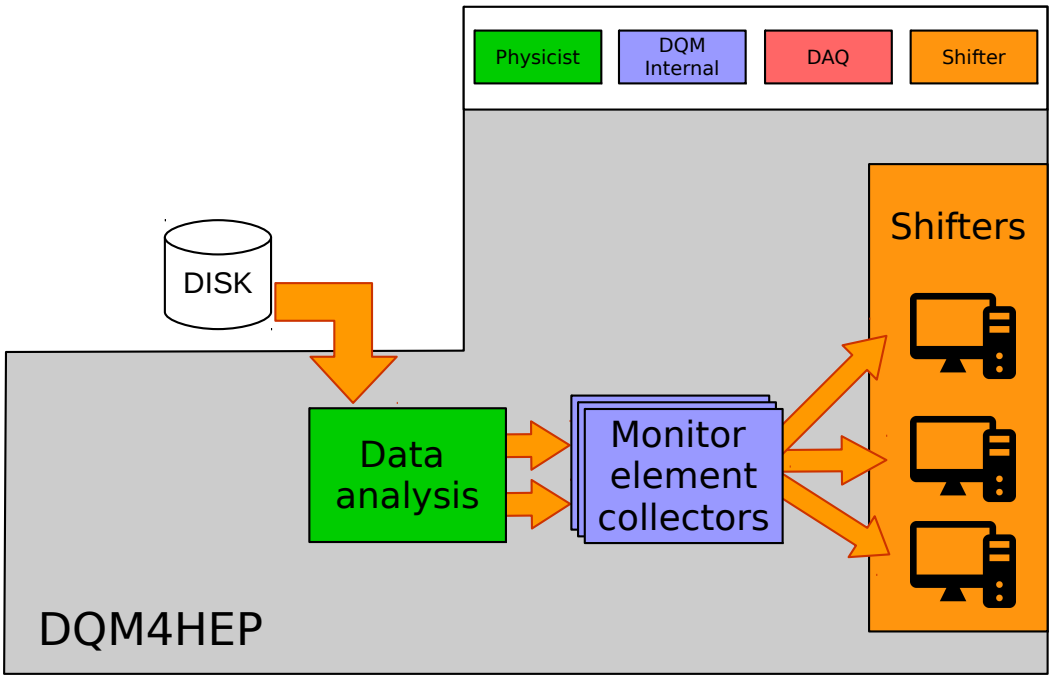

Figure 2.4: The structure of running DQM4hep offline using a file reader module.

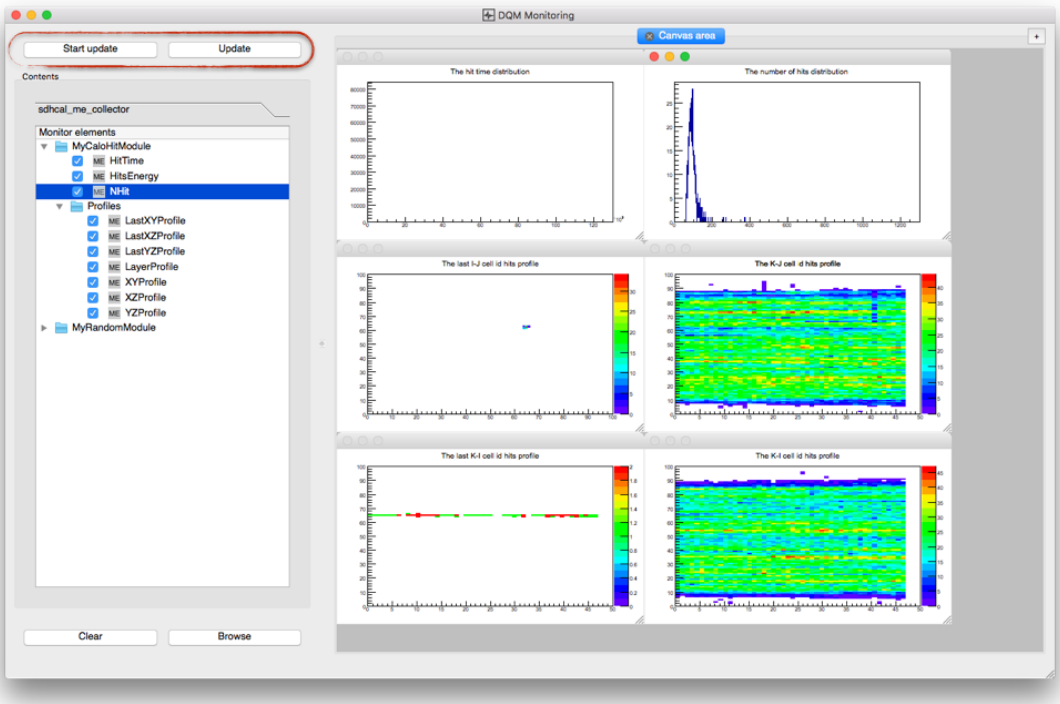

Figure 2.5: An example of the current Qt-based monitoring GUI in use.

New user interface and visualisation package

For the newer versions of DQM4hep, the decision was made to overhaul the GUI and visualisation packages, removing Qt from the framework and moving to a web-based interface.

The removal of Qt was motivated by two reasons. Firstly, the integration with ROOT provided some complications, since running DQM4hep's Qt-based GUI requires an installation of ROOT compiled with the `--enable-Qt` flag enabled. The majority of ROOT installations in remotely-accesible file systems based at CERN and DESY (which are heavily used for analysis and testbeams) were not compiled this way. Secondly, Qt was an additional dependency that must be installed prior to use, making the software more dependent upon the operating system, compiler tools, and environment of the machine, and thus less generic and easy to use.

The removal of the Qt GUI allows for greater freedom with development. The intended goal is to have a browser-based GUI, removing dependency on any external GUI libraries and allowing it to function on any device. This will also make it more user-friendly and convenient, as the interfaces for run control, networking, and data monitoring and quality display can be simply run in different tabs of a web browser.

As of writing, the web interface is under active development by Rémi Eté and is not yet complete. However, a mock-up of the web interface can be seen in Fig. 2.6.

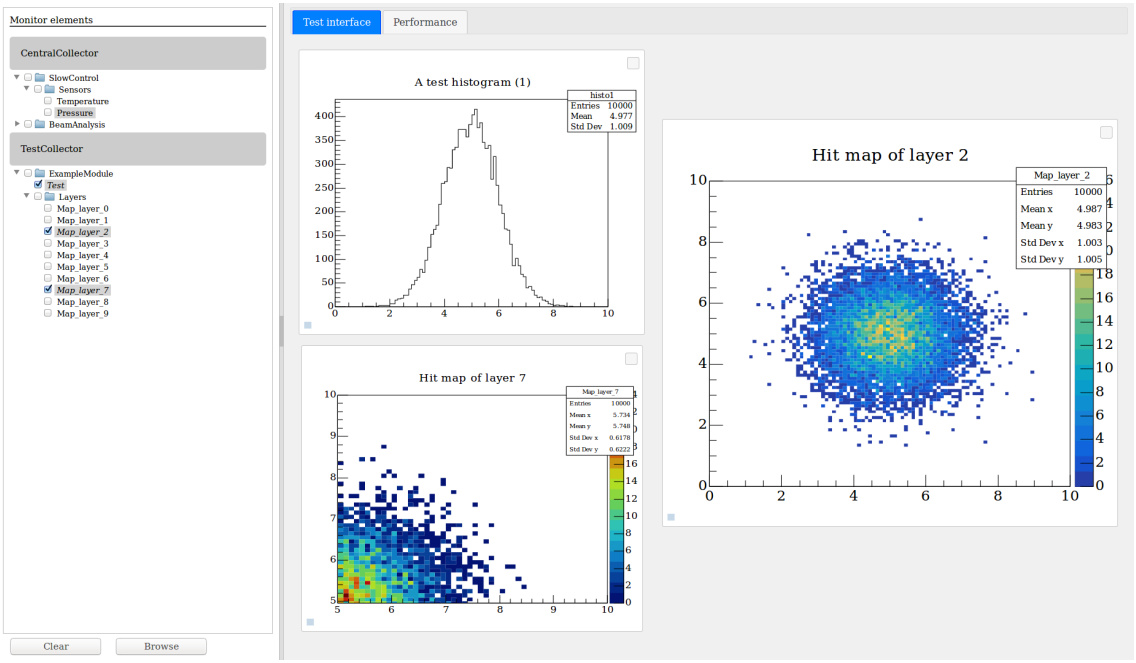

Figure 2.6: A preview of the planned web-based monitoring interface.

2.3 Analysis modules

An analysis module is a plugin that uses data to create monitor elements, which are the core object that drives online monitoring in DQM4hep. Analysis modules are thus a central piece of using DQM4hep as an online monitor.

Mechanically, analysis modules are plugins that read events from a file reader or file streamer plugin, perform some user-defined process to the data to produce a plot, graph, histogram or other ROOT object before emitting it to the rest of the framework as a monitor element.

The most basic analysis module will simply structure information coming from the data acquisition device into a human-readable plot, but analysis modules are able to do anything that can be done in C++ or ROOT, so can contain an arbitrary amount of processing. This means that analysis modules can also be used for online analysis or processing of data, making them very powerful tools for online monitoring.

2.3.1 Running an analysis module

The `dqm4hep-start-module` executable is used to run an analysis module, in combination with an XML steering file. The available arguments are as follows:

```
-h
--help
```

Displays usage information, then exits.

```
-f
--steering-file
```

(Required) Gives the path to the XML steering file that defines what analysis modules to run and their parameters. See the section below (need link) for more information on these steering files.

```
-t
--type
```

The type of module to run. This overwrites the module type in the steering file.

```
-n
--name
```

The name for this instance of the module. This overwrites the module name in the steering file.

```
-v
--verbosity
```

The verbosity of the logger. Options are `trace`, `debug`, `info`, `warning`, `error`, `critical`, and `off`. This is `warning` by default.

```
--
--ignore-rest
```

Ignores any arguments following this flag.

```
--version
```

Displays version information, then exits.

Steering files

An XML steering file is used to pass parameters to the analysis module, including the type of analysis module, the plots to create, and what other processes to connect to. An example steering file can be found in the `dqm4hep-example/tests/` directory.

Steering files are broadly made of four sections: the application settings, the archiver settings, the analysis module, and the monitor elements.

The application settings specify how to run the application itself, including which other file readers or streamers to connect to, which run control is being used, etc.

The archiver setting specify whether the results of running the analysis module are written to an archive, which is a ROOT file containing the monitor elements that the analysis module created.

The analysis module section sets the type of analysis module to run, and gives it a unique name to distinguish it from other instances of the same module. Both of these parameters can be set at the command line (see above).

The monitor element section specifies the monitor elements used in the file. Monitor elements is a general name for any kind of ROOT object, which may be a histogram, graph, plot, drawing, etc. The name, directory, and properties of monitor elements are set here, depending on what type of object the monitor element is.

2.3.2 Creating analysis modules

The DQM4hep pages on Github have a repository to accompany this documentation – the `dqm4hep-example` package is available here[reference this]. This package contains all of the tools necessary to immediately begin writing and compiling analysis modules for an existing installation of DQM4hep.

It is recommended to fork this repository before starting, so git and Github's version control features can be used to back up code.

In addition to writing the analysis module code, a compiled analysis module needs to be declared as a plugin for DQM4hep to be able to use it. The absolute path to the library file for new analysis modules needs to be appended to the `DQM4hep_PLUGIN_DLL` environment variable.

Writing analysis modules

Analysis modules must be written specifically for the type of data they recieve and for a certain analysis or set of analyses to perform. This means that the ideal person to write an analysis module is someone familiar with both the experiment's event structure and the goals of the monitoring.

Each analysis module is a single .cc file. E.g. `ExampleModule.cc` defines an analysis module called *ExampleModule* that can be found in the `dqm4hep-example/source/src/plugins`

directory.

The .cc file then has several sections that must be written, described in separate sections below.

Variable declaration

In this section of the file, the variables that must be persistent over the entire running of the module are declared. These variables will not go out of scope, so are usually reserved for the monitor elements themselves, or counters that must persist over the entire module. Monitor elements must be declared as the `online::OnlineElementPtr` type. None of the declared variables are initialised here; initialision is done in later functions.

readsettings

This function reads the settings from the XML steering file, meaning that anything that needs to be initialised from the steering file is done here. This notably includes all monitor elements. Typically the `core::OnlineElementPtr` will have been declared during the (preamble), then is assigned here based on the steering file using the `online::ModuleApi::getMonitorElement(` function.

initModule

This process is run once, when the analysis module is initialised. Anything that needs to be done only once at the beginning should be done here. The use of this function is limited, as most tasks that need to be reset are reset at the beginning of a run using `startOfRun()` below.

startOfRun

The `startOfRun` function handles code that should be executed only once per run, at the beginning. This is commonly used for counters that must persist over the entire run, for instance if a detector can give an error signal, a counter to store the number of error signals is initialised here so that it is persistent over the entire run, and the number of error signals can be totalled at the end of the run.

endOfRun

This is the end-of-run counterpart to `startOfRun()`. In general, this function will encapsulate logic that must deal with counters or procedures that were intialised or begun

in `startOfRun()`.

endModule

This function is called when the module ends, and is usually used for deleting or cleaning up any objects created during the `initModule()` function. Most normal use cases will not need this function, as cleanup of objects such as variables or monitor elements is handled by the framework.

process

This is the function where the analysis module performs the main processing of data. In this function, events are loaded into memory from the event stream or file reader, and made available for use. Data can then be processed using normal C++ methods, then filled into monitor elements to be sent to the monitor element collectors so they can be presented in the user interface, or stored by the archiver.

Plugin declaration

Analysis modules are DQM4hep plugins, so must be declared as a plugin using DQM4hep's facility for this so that the main executables can access them. At the end of the module, the following code must be included to declare the analysis module as a plugin:

```
DQM_PLUGIN_DECL(ModuleName, "ModuleName");
```

This should take place at the very end of the file but within the `dqm4hep` and `example` namespaces.

When an analysis module is declared as a plugin, the main DQM4hep executable must be given the directory of the library to load the plugin at runtime. This is done by appending the director of the library to the `DQM4hep_PLUGIN_DLL` environment variable:

```
export DQM4hep_PLUGIN_DLL=$DQM4hep_PLUGIN_DLL:/path/to/module/lib/
    libDQMExample.so
```

Once this is added, DQM4hep can access the libraries and run the analysis module.

2.3.3 SimpleModule – a worked example

To explain the process of creating and writing analysis modules in more detail, a worked example is presented. For this we imagine a simplified particle physics detector, and write an analysis module called `SimpleModule.cc` to monitor data from it. We also

write a steering file called `simple-test.xml` to run the analysis module. Each function of the analysis module will be described in depth, explaining in detail what the code is doing.

In this example, the built-in GenericEvent event type is used. For more information on the GenericEvent type, see[reference this]. We will not be using the `startOfCycle` and `endOfCycle` functions.

The full version of these files can be found in the dqm4hep-example repository, available here [reference this].

Detector

We can imagine a simplified particle physics detector as a square plane split into 36 tiles (6 on each side). When a hit occurs the detector reads out the location, ADC, and time of the hit. Each event will correspond to a single hit.

The data acquisition device sends us an event made of four integer values:

- xPos – the position of a hit on the x-axis of the detector, from 0 to 5
- yPos – the position of a hit on the y-axis of the detector, from 0 to 5
- ADC – the ADC of the hit, in arbitrary units, between 0 and 1500
- timeHit – the time the hit occured, in arbitrary units

Goals

Before writing the module, the variables and properties to monitor must be determined. For this detector and its analysis module, there are three goals:

- Spectrum histogram – a single histogram containing the ADC of each hit for the entire run, producing an energy spectrum.
- Hitmap – a hitmap showing the distribution of hits across the detector.
- Radiation damage – if we imagine that ADCs higher than 1000 are likely to cause radiation damage to the detector, then monitoring the number of hits in the entire run exceeding this threshold allows an estimation of how damaged the detector may be.

Preamble

Here the pointers for all monitor elements are initialised. One is required for the ADC spectrum, and one for the hitmap. They don't need to have their type declared here – this is done later. The standard style for DQM4hep is to prefix all monitor element pointers with `m_p` to distinguish them, as they are an important type.

Variables for handling the information about radiation damage are also created here, as they need to persist between events. These are declared here and will be initialised later.

```
private:
  online::OnlineElementPtr m_pSpectrum;
  online::OnlineElementPtr m_pHitmap;
  int radiationDamageADCThreshold;
  int damageHitsCounter;
```

readSettings

Here the monitor elements are assigned, reading in their information from the XML steering file. To do this the `online::ModuleApi::getMonitorElement()` function is used, which searches in the XML steering file for the corresponding information. This function has three arguments: the first is always `this`, the second is the directory the monitor element is in, and the third is a string of the name given to the monitor element in the steering file. These monitor elements are placed at the top directory, so the second argument is `/`.

```
void SimpleModule::initModule() {
  m_pSpectrum = online:ModuleApi::getMonitorElement(this, "/", "ADC_Spectrum
      ");
  m_pHitmap   = online:ModuleApi::getMonitorElement(this, "/", "Hitmap");
}
```

Any other variables that need to be initialised only when the module starts should be placed here. This is a good place to define the threshold for radiation damage:

```
  radiationDamageADCThreshold = 1000;
```

startOfRun

The only thing necessary to do at the start of each new run is to ensure that the counter for hits above the threshold for radiation damage is reset to zero:

```
void SimpleModule::startOfRun(core::Run &/*run*/) {
    damageHitsCounter = 0;
}
```

This ensures that even with multiple runs in the same file, the counter resets correctly.

endOfRun

Once a run has finished, the number of hits that might have caused radiation damage should be read out. There are many ways to do this, but the simplest is to send a message to the logger:

```
void SimpleModule::endOfRun(const core::Run &/*run*/) {
    dqm_info("Number of hits above radiation damage threshold: {0}",
        damageHitsCounter);
}
```

For more information on the `dqm _info()` function and it's syntax, see the section on the logging tools here [reference this].

process

The first thing to do is make sure that the `process()` function has access to the GenericEvent. This is done by making it an argument of the function, calling it `pEvent` to make it clear that this is a pointer to an event and not an event object. Basic error-checking is then done, to ensure that the current event exists:

```
void SimpleModule::process(core::EventPtr pEvent) {

  if (nullptr == pEvent) {
    dqm_warning("Event pointer is invalid - skipping this event");
    return;
  }
```

The `dqm_warning()` function publishes a message to the logger, with the `warning` level. See the section on the logging tools for more information [reference this].

Forcing this function to `return` ensures that the an analysis module doesn't attempt to access an event that does not exist. Otherwise, this would cause a segmentation fault and crash the analysis module. Since the `process()` function runs separately for each event, this has the effect of skipping to the next event.

Now the pointer of the event needs to be assigned so that the object itself can be accessed. A new variable of type `core::GenericEvent` is created, then the `getEvent()` function is used to assign it.

```
core::GenericEvent *pGenericEvent = pEvent->getEvent<core::GenericEvent>()
    ;
```

A variable is needed to store the information pulled from the event, then the information can be extracted using the `getValues()` function:

```
int xPos;
int yPos;
int ADC;
int timeHit;

pGenericEvent->getValues("xPos", xPos);
pGenericEvent->getValues("yPos", yPos);
pGenericEvent->getValues("ADC", ADC);
pGenericEvent->getValues("timeHit", timeHit);
```

The `getValues()` function takes two arguments: the first is a key in the form of a string, which identifies a piece of data within the GenericEvent. The second is the object to place the retrieved data into. In this case, the same names are used for simplicity. This works because the first variable is a string, used as a key to find information within the GenericEvent.

Now all the data is loaded into memory and available for use.

First the ADC is added to the `ADC_Spectrum` plot. To do this, the monitor element `m_pSpectrum` must be cast to the correct ROOT object type, using the `objectTo()` function, then use ROOT's `Fill()` function:

```
m_pSpectrum->objectTo<TH1I>()->Fill(ADC);
```

Then the same must be done for the hitmap. This time it needs to be cast to a TH2I object and filled with the x- and y-positions as well as the ADC:

```
m_pHitmap->objectTo<TH2I>()->Fill(xPos, yPos, ADC);
```

And lastly, a check is performed for whether the ADC was high enough to cause radiation damage, and if so increment the counter:

```
if (ADC >= radiationDamageADCThreshold) {
  damageHitsCounter++;
}
```

Steering file

Now the analysis module is complete, there needs to be a steering file to run it. This is started by making a file called `simple-test.xml`. Then the `<dqm4hep>` environment is opened, as this will contain everything else:

```
<dqm4hep>

    <!-- everything else will be in here -->

</dqm4hep>
```

There are then four sections to write: the application settings, the archiver settings, the analysis module, and the monitor elements.

Application settings

This section of the steering file controls the parameters that are needed to run the module itself. This part of the file is where the module is specified to be running online (receiving events from an event collector) or offline (receiving events from a file reader).

This analysis module will be run offline, so the important fields here are `EventReader`, which is which type of file reader to use to read events, and `EventFileName`, which specifies the path to the file to read.

If this were running online, the `RunControl`, `EventCollector`, `EventSource` and `MonitorElementCollector` would have to give the names of those processes. When running offline, these names can be set to anything, so they are given dummy names.

```
<settings mode="EventReader">
  <parameter name="EnableStatistics"> true </parameter>
  <parameter name="EventReader"> SimpleEventReader </parameter>
  <parameter name="EventFileName"> /foo/bar/simpleEventDatafile.root </
      parameter>
  <parameter name="CyclePeriod"> 1 </parameter>
  <parameter name="CycleCounter"> 0 </parameter>
  <parameter name="CycleTimeout"> 0 </parameter>
  <parameter name="RunControl"> DummyRunControl </parameter>
  <parameter name="EventCollector"> DummyEventCollector </parameter>
  <parameter name="EventSource"> DummyEventSource </parameter>
  <parameter name="MonitorElementCollector"> DummyMECollector </parameter>
</settings>
```

Archiver settings

This section controls the archiver, which creates an archive of all monitor elements in the form of a ROOT file when the analysis module exits. If the archiver isn't needed, it can be set to `enable="false"` and ignored.

In this case the archiver is needed, so it is set to `true`. A filename for the archive needs to be given, e.g. `archive-run42.root`. The OpenMode is chosen to be `RECREATE` as the archive should be re-written every time the module is run, although `APPEND` would add events onto an existing file. `AllowOverwrite` is set to `true` so that old archives can be overwritten easily. As run numbers are not used in the analysis module, `AppendRunNumber` is set to `false`.

```
<archiver enable="true">
  <parameter name="FileName" value="archive-run42.root"/>
  <parameter name="OpenMode" value="RECREATE"/>
  <parameter name="AllowOverwrite" value="true"/>
  <parameter name="AppendRunNumber" value="false"/>
  <selectors>
    <selector regex=".*" select="true"/>
  </selectors>
</archiver>
```

Analysis module

This section contains a declaration of which analysis module to execute, and the name to give it. A running analysis module needs a unique name to distinguish it from other modules of the same type running in the same environment.

This steering file has to run SimpleModule. Only one instance is needed at a time, but the name has to be different than the module's base name, so:

```
<module type="SimpleModule" name="mySimpleModule"/>
```

Monitor elements

This section is where the monitor elements to use and the parameters of the ROOT objects are declared. This entire section will be within the `<storage>` and `<monitorElements>` environments:

```
<storage>
  <monitorElements>
```

```
    <!-- monitor elements will go here -->

  </monitorElements>
</storage>
```

To create a ROOT object, the `<bookElement>` environment is used to declare it's type, path, name, title, and any other parameters the ROOT object requires. For example, to create a histogram for the ADC spectrum, a TH1I is needed as the ADCs are integers. The range of the ADCs is from 0 to 1500, so the range can be set accordingly.

```
<bookElement type="TH1I" path="/" name="ADC_Spectrum" title="Spectrum of all
    ADCs" nBinsX="150" minX="0" maxX="1500">
</bookElement>
```

Similarly, the monitor element for the hitmap is defined:

```
<bookElement type="TH2D" path="/" name="Hitmap" title="Hitmap of the example
    detector" nBinsX="6" minX="0" maxX="5" nBinsY="6" minY="0" maxY="5">
</bookElement>
```

XML loops

In this example, loops aren't necessary, but defining monitor elements using a for-loop in XML is a useful feature, so it is discussed here. For example, if one of the goals were to create a histogram for each of the 36 tiles in the detector, a for-loop would allow thtis to be created with a smaller amount of code

To do this, the `<for>` environment is used, using `tileNumber` as the id. Then a template monitor element defintion is written out using `$ FORtileNumber` whenever the tile number should be inserted:

```
<for id="tileNumber" begin="0" end="35" increment="1">
  <bookElement type="TH1D" path="/" name="Tile$FOR{channelNum}" title="
      Spectrum for tile $FOR{channelNum}" nBinsX="150" minX="0" maxX="1500">
  </bookElement>
</for>
```

This would then create a series of monitor elements called `Tile0`, `Tile1`, `Tile2`, etc.

Building and running

To compile the analysis file, the normal build commands are issued from the `dqm4hep-example/build` directory:

```
cmake ..
make install
```

Running cmake is only needed for the first compilation after creating a new module – it isn't necessary when recompiling a module that has been compiled before.

Once the analysis module has compiled successfully, DQM4hep must be given the path to its libraries so that the main installation of DQM4hep can utilise it. This is done by appending the path to the libraries to the `DQM4hep_PLUGIN_DLL` environment variable:

```
export DQM4hep_PLUGIN_DLL=$DQM4hep_PLUGIN_DLL:/path/to/dqm4hep-example/lib/
    libDQMExample.so
```

In order to run, the network manager dim must also be running, but for offline use this is simple. In a new terminal window:

```
export DIM_DNS_NODE=localhost
dns
```

Then to run the analysis module, the `dqm4hep-start-module` executable is run, pointing it to the steering file using the `-f` argument. Since a logging output at the 'info' level was implemented, the argument `-v` must be given to manually set the logging level to `info` so that it can be seen in the output.

```
dqm4hep-start-module -f simple-test.xml -v info
```

The analysis module will then run. Once it has finished, the archiver will create the archive in the directory it was run in.

2.4 Data quality monitoring

Data quality monitoring (DQM) is a type of data monitoring where the data is tested using some form of statistical or mathematical process to produce a value corresponding to the "quality" of the dataset. This can take many forms, such as comparing an experimental dataset to reference data acquired from previous experiments, or requiring that the χ^2 or p-value of a dataset may need to pass a certain threshold to be considered valid.

The definition of the "quality" statistic will differ according to a variety of factors such as the type of data, the aim of an experiment, etc. Common examples are p-values, or binary pass-fail tests where data that passes has a quality of 1, and 0 otherwise.

One of the benefits of data quality monitoring is that it provides a more reproducible and robust set of checks on data-taking, allowing quantitative analysis of the performance

of a detector prototype. It can also be used as a way for shifters without detailed knowledge of the hardware, software, or physics to determine whether the detector is performing as intended during a testbeam when experts are not available, by using the quality statistics as a guide.

Previous versions of DQM4hep did not have infrastructure to support data quality monitoring, but this was added during refactoring in preparation for the next release version. Once this was in place, this permitted an array of quality tests to be developed, implemented, and tested.

A quality test (or qtest) processes a series of monitor elements (ROOT TObjects) according to a set of criteria defined in the test's code. This test produces a numerical result between 0 and 1, referred to as the "quality". Within the framework, quality tests are self-contained C++ code files, which hook into the framework's system for execution. Quality tests are run by using the executable `dqm4hep-run-qtests` and a steering file to define parameters, such as which files to load, which quality tests to execute, and what the passing and failing boundaries are for each quality test.

2.4.1 Quality tests

The quality tests that have been implemented in DQM4hep are described in detail below. Each test requires a certain type of object as an input and has it's own definition of what the "quality" statistic represents. Some quality tests also require a reference to compare against the input data, which is also described.

A summary of the quality tests can be found in Table 2.1.

Property within expected test

This is a quality test that takes either a TH1 or TGraph object, and finds some user-defined parameter. The parameter must be one of: mean, mean90, root mean square (RMS), root mean square 90 (RMS90), or median. It then checks whether either: that this parameter is within a user-specified range; or that it is above or below the user-specified threshold. If a range is being used, the result is the p-value of the property being within the specified range. If a threshold is being used, then the result is 1 if the property passes the threshold, 0 otherwise.

Exact reference comparison test

This is a quality test that takes any TObject, and compares it to a user-specified reference object (which must be of the same type). The result is 1 if the two objects are exactly identical, 0 otherwise.

Fit parameter in range test

This is a quality test that takes either a TH1, TGraph, or TGraph2D object and plots a user-defined function onto it, solving for one of the parameters of the function, then checks it against a user-defined range. The result is the p-value of the parameter being within the specified range.

Kolmogorov-Smirnov test

This is a quality test that takes either a TH1 or a TGraph object, and performs the Kolmogorov-Smirnov test between that object and a specified reference. The result is the p-value of the Kolmogorov-Smirnov test. The Kolmogorov-Smirnov test is intended for unbinned data, not histograms, but ROOT provides a function for performing the Kolmogorov-Smirnov test on histograms, so this is functionality is also included for the sake of completeness.

Pearson χ^2 test

This is a quality test that takes a TH1 object and performs the Pearson χ^2 test between that object and a specified reference. This test is analogous to the Kolmogorov-Smirnov test, but is designed specifically to work for binned histogram data. The result is the p-value output by the χ^2 test.

2.4.2 Running quality tests

Quality tests can be run using the `dqm4hep-run-qtests` executable, found in `dqm4hep-core/bin/`. This executable handles the running of the actual binaries for each qtest, as well as obtaining monitor elements from the ROOT file and the setting of parameters. This executable has one required arguments and several optional ones, detailed below.

Arguments

```
-h
--help
```

Quality test name	ROOT Objects	Required parameters	Optional paramters
PropertyWithinExpectedTest	TH1, TGraph,	Property, Method, plus others (see [ref])	
ExactRefCompareTest	Any TObject	None	CompareUnderflow CompareOverflow
FitParamInRangeTest	TH1, TGraph, TGraph2D	FitFormula, TestParameter, DeviationLower, DeviationUpper,	GuessParameters, FunctionRange, UseLogLikelihood, UsePearsonChi2, ImproveFitResult
KolmogorovTest	TH1, TGraph	None	UseUnderflow, UseOverflow
Chi2Test	TH1	None	ComparisonType, UseUnderflow, UseOverflow

Table 2.1: Table summarising all quality tests implemented in DQM4hep and their properties.

Displays usage information, then exits.

```
-i <string>
--input-qtest-file <string>
```

(Required) Gives the path to the XML steering file that defines what quality tests to run, their parameters, and what monitor elements to run them on. See the section below for more information on these steering files.

```
-c
--compress-json
```

Turns on compression for the JSON qtest report output file. Off by default.

```
-w
--write-monitor-elements
```

Turns on writing of monitor elements in the qtest report. Off by default.

```
-p <string>
--print-only <string>
```

Prints only the quality reports of the given flag. Options are `undefined`, `invalid`, `insuf_stat`, `success`, `warning`, `error`.

```
-e <string>
--exit-on <string>
```

Forces the program to exit if any qtest results in the given code. or greater. Options are `ignore`, `failure`, `warning`, `error`. This is `failure` by default.

```
-v <string>
--verbosity <string>
```

The verbosity of the logger. Options are `trace`, `debug`, `info`, `warning`, `error`, `critical`, and `off`. This is `warning` by default.

```
-q <string>
--qreport-file <string>
```

Gives the path of the qtest report output file (in JSON) format.

```
-o <string>
--root-output <string>
```

Gives the path of a ROOT output file to save the processed mmonitor elements.

```
--
--ignore_rest
```

Ignores any arguments following this flag.

```
--version
```

Displays version information, then exits.

Steering file

Steering files use XML to store all teh information needed to execute a qtest. An example steering file can be found in `dqm4hep-core/tests/test _samples.xml`.

There are two main sections: the `<qtests>` block and the `<monitorElements>` block, both of which must be within the `<dqm4hep>` XML tag.

The `<qtests>` block defines the qtests to execute along with their settings or parameters, without reference to what they will be run on. The structure and parameters of these is highly dependent upon the qtest being used – see the section for each qtest above.

The `<monitorElements>` block opens a file using the `<file>` tag, within which each monitor element is opened with `<fileElement>`. Inside this tag, all of the qtests to execute on this monitor element are given. In this example below, the qtests `ExampleTest1` and `ExampleTest2` are both performed on the monitor element `TestHistogram`:

```
<monitorElements>

  <file name="test_samples.root">
    <fileElement path="\TestDirectory" name="TestHistogram">
    <qtest name="ExampleTest1" />
    <qtest name="ExampleTest2" />
    </fileElement>
  </file>

</monitorElements>
```

Some kinds of qtests require reference objects to compare against, which must be declared in the `<references>` block. References have a `name` parameter which gives the path to the file used as a reference, and an `id` which is a short tag for referring to them later in the XML file. For example:

```
<references>
  <file id="mc-ref" name="montecarlo_reference_samples.root"/>
  <file id="ex-ref" name="experiment_reference_samples.root"/>
</references>
```

When a qtest that requires a reference is declared, the reference is given within the `<fileElement>` tag:

```
<fileElement path="\TestDirectory" name="TestHistogram">
  <reference id="MyReference"/>
  <qtest name="ExampleTest1"/>
</fileElement>
```

This performs the qtest `ExampleTest1` on the monitor element `TestHistogram`, looking for another ROOT object of the same name within the file `MyReference` points to. It is also possible to use a specific object in a file as the reference:

```
<fileElement path="\TestDirectory" name="TestHistogram">
  <reference id="MyReference" path="/path/to/the/reference/file" name="
      ReferenceHistogram"/>
  <qtest name="ExampleTest2"/>
</fileElement>
```

In this case, this performs `ExampleTest2` on the monitor element `TestHistogram`, using the object `ReferenceHistogram` as the reference.

2.4.3 Writing quality tests

Users can create their own quality tests if the included tests do not satisfy their requirements. Quality tests are a type of plugin – see the *plugin system* section of the DQM4hep documentation [reference this]for more information on plugins, including how to write and compile them.

The code for the built-in quality tests can be found in `dqm4hep-core/source/src/plugins/` and can be used as references or templates. Files for quality tests should be given a descriptive name in CamelCase, and end with `Test`, e.g. `ExactRefCompareTest.cc`.

The code for a quality test requires only a single .cc file, which has four functions: the constructor, the destructor, readSettings, and userRun. Each is discussed in a separate subsection below. If required, further functions can be implemented as needed. This is left for the user to decide.

Code should be written to catch common errors and throw appropriate exceptions, especially for errors that cause segmentation faults. This will help to avoid a qtest preventing other qtests from running should an error occur. Errors are reported using the `report.m_message()` function, and the error message will appear in the summary of the qtest after it has run.

Constructor and destructor

For the constructor and destructor, it's enough to copy existing code, changing the name of the qtest and the variables to be initialised. The program that runs qtests handles everything else. Care should be taken to initialise variables properly and to give the qtest a good description:

```
ExampleTest::ExampleTest(const std::string &qname)
    : QualityTest("ExampleTest", qname),
      m_someFloatParameter(0.f),
      m_someIntParameter(0)
{
  m_description = "A description of the test's functionality, as well as the
      meaning of the quality statistic it outputs.";
}
```

readSettings

The `readSettings` function initialises the variables of the qtest from the XML steering file, which is loaded into memory via `xmlHandle`. This function should be used to read in information from the XML file and validate it to make sure the test can be run. Variables are read in using a combination of pre-processor macros and XmlHelper. For example:

```
RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, XmlHelper::readParameter(xmlHandle
    , "PropertyName", m_property))
```

Note that the above example will fail if the parameter is not present in the XML file, so should only be used for parameters that are required. If a parameter is optional, use the `RETURN_RESULT_IF_AND_IF` macro instead. This allows the parameter to be returned if it is found, or does nothing if it is not. For example:

```
RETURN_RESULT_IF_AND_IF(STATUS_CODE_SUCCESS, STATUS_CODE_NOT_FOUND, !=,
    XmlHelper::readParameter(xmlHandle, "PropertyName", m_property))
```

For more information on status codes, pre-processor macros, and XML parsing with XmlHelper, see the core tools section of the documentation.

Parameters should be checked to make sure that the qtest can be run and that the result is meaningful. While `XmlHelper::readParameter` and similar functions can take an optional fourth argument for a validator delta function, users should make code clear and readable by using `if-else` statements. This is especially important when checking against more complicated criteria.

userRun

The `userRun` function defines the process of the qtest itself, using the monitor element. The result *must* be a float between 0.0 and 1.0 that represents the "quality" or "goodness" of the test. The meaning of this quality statistic depends on the test but is often a p-value. At absolute minimum, it should represent a pass-fail case, so that a passing qtest gives a quality of 1 and a failing qtest gives a quality of 0.

The monitor element must first be cast to an appropriate class. This should be done using the `objectTo` function. For example, if the monitor element is a TH1:

```
TH1* myHistogram = pMonitorElement->objectTo<TH1>();
```

After this, the object can be accessed using it's normal methods, and the qtest can be written using normal C++ code for ROOT objects.

It is useful to include a check for whether the monitor element exists and is the correct type, to prevent segmentation faults, using a comparison to `nullptr` and throwing an appropriate status code if the check fails. For example:

```
if (nullptr == pMonitorElement->objectTo<TH1>())
{
  report.m_message("Object does not exist or is of unrecognised type!");
  throw StatusCodeException(STATUS_CODE_INVALID_PTR);
}
```

The meaning of these status codes is documented under *Status codes and useful pre-processor macros* in the core tools section of the documentation.

Once the quality statistic of the test is known, it is output using `report.m_quality`. Any other information can be output using `report.m_message` – this is useful for including comments on the result.

2.5 Documentation and user manual

One of the biggest hurdles for the promotion and uptake of a new framework is the lack of understanding or familiarity with it's use. Many research teams will continue to use existing software solutions, which may be suboptimal or difficult to use, according to the principle of "better the devil you know than the devil you don't". The first step to overcoming this is to produce clear, readable and complete documentation across the entire range of features the framework has.

The DQM4hep framework has two sets of documentation with different intended readers and different aims, so these will be discussed separately.

2.5.1 Doxygen documentation

Doxygen is a tool for automatically generating documentation resources for C++ code, relying on marked sections of documentation written within the actual code itself. Doxygen is able to directly obtain the structure of code, objects, functions, etc. from the code, allowing it to automatically generate a complex and rich set of documentation that categorises and indexes objects based on their inheritance, namespace, etc.

Doxygen can also generate an HTML- or LaTeX-based document that can be used as a local reference guide or hosted online. This makes Doxygen a powerful tool for documenting the technical aspects of code, demonstrating hierarchies of functions and objects, and an extremely useful reference guide for large programs or frameworks.

While Doxygen documentation is extremely useful, it does have some limitations. Doxygen functions more as a technical reference for code, lacking any overviews or instructions due to it's automatic generation. This kind of documentation lacks a holistic element, and has no way for new users or those less familiar with the codebase to understand the overarching concepts. This can make it inaccessible for new users. The way this was addressed will be discussed in the next section.

DQM4hep has a Doxygen website hosted on the internet, which is available here [4].

2.5.2 User manual

The existing documentation featured the common elements of the framework – such as the plugin system, event interface, logger, and XML parser – explained in detail, with clear examples and straightforward advice on their use.

My contribution to this was to write in-depth explanations on the structure, usage, and creation of analysis modules and quality tests, the two parts that are most specific to end-users. The experience acquired using the framework and deploying it on testbeams for the first time, as well as integrating it with different detectors, provided a strong knowledge base to write the user manual intended for a user approaching the framework for the first time. These testbeams are described in more detail in Chapters 3 and 4.

The user manual can be found online here [3].

Analysis module guide

There is an in-depth explanation of each of the component functions of an analysis module, discussing which actions or data processing should be done in each, and their intended purposes. There is also an explanation of the XML steering files that are necessary to run an analysis module, and instructions on how the executable is run, along with it's arguments.

In addition to this there is a worked example of an analysis module from start to finish. A simplified particle physics detector and it's data format is defined, and then the reader is lead through the process of writing an analysis module step-by-step and function-by-function to obtain certain plots and results from the data. This is intended to give a more concrete demonstration of usage, as an easier to follow example. This section is also written in simple and plain English in order to be as accessible as possible.

Along with this documentation there is the `dqm4hep-example` Github repository [cite], which contains all of the code used in the analysis module user manuals, as well as the required cmake and make files to begin compiling projects right away. The intention of this repository is for new users to make a copy, write their own modules, and have everything needed for compiling and running them "out of the box" straight away. The user manual references this repository, so that users can see the files from the worked example in their full form.

Quality testing guide

There is extensive documentation of quality testing; each quality test is described in detail, including their purpose, output, required parameters, and optional parameters. In addition to this, there is a guide for how to run quality tests, including an explanation of running from the command line and an in-depth look at the structure of the XML steering files required.

Finally there is a section explaining how to write new quality tests. This includes a detailed explanation of the purpose of each function within a quality test file, what the required outputs are and how to utilise them, error testing, and advice on maintaining a style consistent with the rest of the framework.

2.6 Adaptation to other detectors

Due to the modularity and genericity of DQM4hep, the process of deploying it for a new detector is simple – the only parts of DQM4hep that need to be made for any specific use case are the analysis modules, standalone modules, streamer plugin, and file reader plugin. For all of these plugins, there are templates available in the codebase, as well as examples of in-use plugins for other detectors. A few special DQM4hep-specific functions are necessary for these plugins to hook into the framework properly, but apart from these all user-provided plugins are written in normal C++ code that also integrates ROOT, so should be familiar to most users.

A file streamer or file reader must be written by the user given a specific data structure. This requires knowledge of both the event data model of the data acquisition setup, as well as the structure of the data files. The ideal person to write this code is someone with detailed knowledge of the data acquisition software being used, and the data storage or streaming.

In general, only one of the file streamer or file reader plugins will be needed. Both of these plugins are similar in structure and differ only on where they get the data from – a file reader loads a file from disk, whereas a streamer loads it from the data acquisition system. If the data will be monitored offline or "nearly-online" by loading files from disk, then a file reader plugin must be written. If the data is to be monitored online, then a streamer plugin must be written.

Once the information is accessible from either the file reader or streamer, the framework handles passing this data to the analysis modules. Analysis modules are a type of plugin which take data that has been packaged into events by a file reader or streamer plugin and performs some analysis on it. The main action an analysis module must do is create a monitor element (a ROOT TObject) then emit it to the rest of the framework. Before this step an arbitrary amount of processing can be done, e.g. checking validation bits, thresholds, error-checking, and so on. Monitoring the data quality can be done from within analysis modules but this is not recommended as dedicated quality tests (see section above) are available.

For each analysis module that is being run, an XML steering file is required to provide the parameters and networking information to all the processes needed. A single steering file can call only a single analysis or standalone module, but multiple steering files can be run in parallel by the framework.

Examples of using DQM4hep with testbeams both within the AIDA-2020 community

and outside of it can be found in Chapters 3 and 4, respectively.

Chapter 3

AIDA-2020 testbeams

I love fools' experiments.
I am always making them.

Charles Darwin

One of the most important aspects of testing and developing DQM4hep was to ensure that it was as generic as it was intended to be, and this meant deploying and using the framework on physics testbeams. DQM4hep was originally developed during testbeams of the Silicon Tungsten Electronic Calorimeter (SiWECAL), and its early testing phases were predominantly based on this detector, so it was apparent that it could be used in the originally-intended setting. However, in trying to develop it as a generic monitor, and to satisfy the requirements of a generic data monitoring and quality monitoring tool for AIDA-2020, it was essential that it was tested on other detectors of different types to demonstrate its generic nature.

In this chapter, the deployment, testing and usage of DQM4hep on testbeams within the AIDA-2020 collaboration will be discussed in detail. For a testbeam where DQM4hep was deployed outside of the AIDA-2020 community, see Chapter 4. Three testbeams will be described in detail, as they represent the different stages of using DQM4hep as a data monitoring tool, from first implementation and deployment, to further developments, and finally as a mature tool integrated into the workflow of a testbeam.

The CALICE-AHCAL prototype

The Analogue Hadronic Calorimeter (or CALICE-AHCAL) is a sampling calorimeter formed of steel absorber plates and plastic scintillator tiles, read out by silicon photomultipliers (SiPMs) as active material [5]. One of the important features of the AHCAL is that the prototypes were designed to be made using techniques suitable for mass production,

such as injection-moulding and automated foil-wrapping of the scintillator tiles, and pick-and-place assembly of the layers and their electronics. It also uses power pulsing – rapidly cycling power so that the electronics are active only when the beam is present, according to a known beam structure. This helps to reduce power consumption and heat production, making cooling the layers easier. Some assembled layers of the AHCAL prototype can be seen in Fig. 3.1.

Figure 3.1: Two varieties of AHCAL layers.

CALICE testbeams

The CALICE testbeams were done with the *Forschung mit Lepton Collidern* (FLC)[1] group based at DESY in Hamburg, working on the AHCAL prototype. Regular testbeams were held at the DESY II synchrotron at DESY in Hamburg, Germany and at the Super Proton Synchrotron (SPS) at CERN in Geneva, Switzerland. The goals for these testbeams varied over time but common focuses for the hardware were power-pulsing tests, and commissioning and calibration of new detector boards to test variations and changes to the manufacturing process. The testbeams were often used as testbeds for data acquisition electronics and software, such as the EUDAQ data acquisition software, the AIDA-2020 Beam Interface (BIF), and DQM4hep.

DQM4hep was used as an online monitoring and data quality monitoring tool for AHCAL tesbeams beginning in May 2016, and in further testbeams between 2016 and 2018. The majority of these testbeams occurred at the DESY II facility, but two took place at the CERN SPS in May 2017 and June 2018.

[1] EN: Research with Lepton Colliders

3.1 May 2016 testbeam at DESY II

The first deployment of DQM4hep on an AHCAL testbeam was at DESY II during May 2016. The testbeam was to be two weeks in duration, following a one-week setup and preparation period. Besides testing the deployment and usage of DQM4hep, the goals of this testbeam where to test MIP calibration of a new AHCAL base unit (HBU), to test the power pulsing feature, and to perform TDC calibrations. In addition to these goals for the AHCAL, a device called a Beam Interface (BIF), another part of the ongoing work of AIDA-2020 Work Package 5, was being tested [reference this].

Before and during the testbeam, the majority of the development for AHCAL-specific analysis modules was undertaken. Prior to this, DQM4hep had only been used on Si-WECAL beams, and was untested for other detectors.

File reader and streamer plugins for the LCIO data format were already available in a DQM4hep package called `dqm4ilc`[2]. This meant that the plugins necessary for the framework to open, access and serialise the LCIO data format were already available.

3.1.1 Data format

The data for the AHCAL is in the Linear Collider Input/Output (LCIO) format, using an object type called LCGenericObject, which is a generic format for use when the existing data formats are not suitable. It comprises two parts: the block of data itself, held in 14-bit numbers; and a header containing user-defined parameters, in this case a timestamp, a typename for the object, and a description of the data contained in the object.

The structure of a single event in LCGenericObject format can be seen below, which is the result of using the `dumpevent` tool to dump the contents of an LCIO event to the command line:

```
--------------- print out of LCGenericObject collection ---------------

  flag:  0x0
 parameter DAQquality [int]: 1,
 parameter DataDescription [string]: i:CycleNr:i:BunchXID;i:EvtNr;i:
     ChipID;i:NChannels:i:TDC14bit[NC];i:ADC14bit[NC],
 parameter Timestamp [string]: Tue, 09 Feb 2016 18:20:43 +0100,
 parameter TypeName [string]: CaliceObject,
```

[2]This package is now deprecated.

```
[   id   ] i:Type,i:EventCnt,i:TS_Low,i:TS_High - isFixedSize: false
-------------------------------------------------------
[00000004]  i:0; i:15; i:15; i:0; i:36; i:12423; i:12422; i:12421;
    i:12420; i:12419; i:12418; i:12417; i:12416; i:12415; i:12414;
    i:12413; i:12412; i:12411; i:12410; i:12409; i:12408; i:12407;
    i:12406; i:12405; i:12404; i:12403; i:12402; i:12401; i:12400;
    i:12399; i:12398; i:12397; i:12396; i:12395; i:12394; i:12393;
    i:12392; i:12391; i:12390; i:12389; i:12388; i:12459; i:12458;
    i:12457; i:12456; i:12455; i:12454; i:12453; i:12452; i:12451;
    i:12450; i:12449; i:12448; i:12447; i:12446; i:12445; i:12444;
    i:12443; i:12442; i:12441; i:12440; i:12439; i:12438; i:12437;
    i:12436; i:12435; i:12434; i:12433; i:12432; i:12431; i:12430;
    i:12429; i:12428; i:12427; i:12426; i:12425; i:12424;
-------------------------------------------------------
```

In this case, the `TDC14bit[NC]` and `ADC14bit[NC]` are arrays, each holding a number of elements equal to the `NChannels` variable, in this case 36. Each element of these arrays corresponds to a single physical scintillator tile within the detector, and identifies which chip it belongs to using `ChipID`. The `ADC14bit` and `TDC14bit` arrays contain binary data, which is represented above converted directly to decimal. An additional conversion from this format to specific information such as validation bits, hit bits, etc. was also needed.

3.1.2 Results

Over the course of the preparation week, the foundations were laid for the analysis module. This involved gaining a familiarity with the data structure, loading data from previous testbeams into DQM4hep offline, and attempting to read it in basic ways. The first module was not ready for the beginning of the testbeam proper, but a few days afterwards we were able to produce plots in DQM4hep from "nearly-online" testbeam data.

The first analysis module developed was the `AHCALRawModule`. The majority of the processing in this module was decoding of the data from the binary format and extracting the information from it. After this, validation bits and hit bits in the data were checked to classify data as 'good' or 'bad' hits. Then the actual ADCs and TDCs were filled into their respective histograms.

Figure 3.2: The T22 beam area at the DESY II synchroton, looking towards the beam aperture.

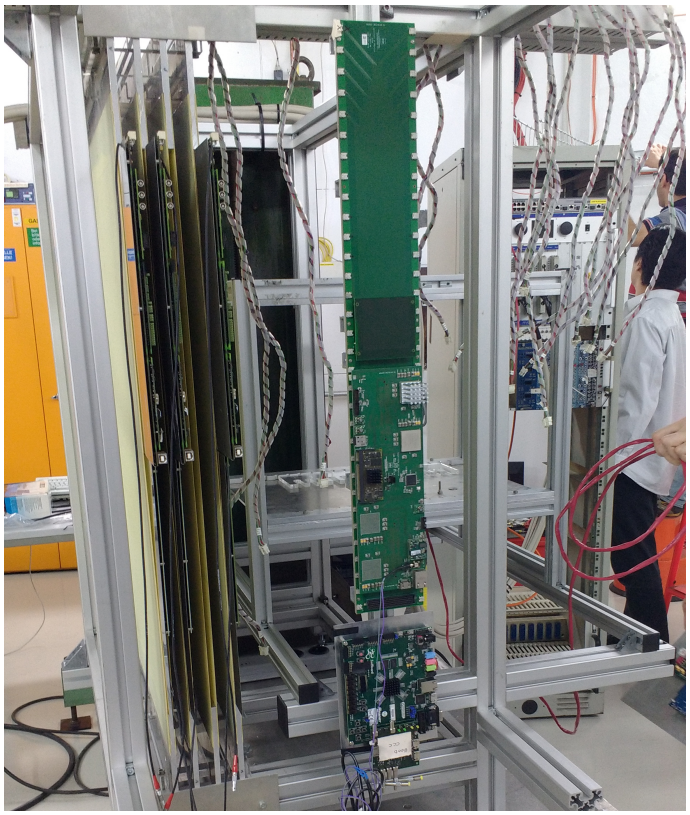

Figure 3.3: Close-up of the layers and electronics assembled in the support structure in the beam area. During operation, the detectors were covered with black fabric to block ambient light from impinging upon the SiPMs.

The first module acted as a proof-of-concept, and once this was done further work started on creating more modules with a wider variety of features and plots to provide better coverage for online monitoring. We created and refined two separate modules during this testbeam – `AHCALRawModuleChannel` and `AHCALRawModuleGlobal`.

The channel module created a per-spectrum channel of all ADCs, integrated over the whole run. It was able to load a number of individual channels, though due to the memory requirement, it could not track all channels simultaneously. To work around this, we implemented a facility for the module to read which channels to monitor from the XML steering file, so that these could be defined at runtime. An example of some of the per-channel spectra created using this module can be seen in Fig. 3.4

The global module produced a 2D histogram containing cells for each channel, coloured for the ADC in that channel. This didn't produce a hitmap as the geometry information was not available in this plot, but did allow easy identification of dead channels and channels that were in the beamspot. An example of this can be seen in Fig. 3.5.

Overall, once the monitoring had been set up and initial bugs and problems fixed, it became a routine tool of the testbeam. This was made easier by the usage of XML steering files for the monitoring interface canvases, allowing groups of plots and histograms to be automatically opened when the monitoring interface was started. This meant that with little effort, all information necessary for monitoring was easily available.

An example of the monitoring interface in use can be seen in Fig. 3.6.

3.2 July 2016 testbeam at DESY II

The next usage of DQM4hep was a testbeam during July 2016, also at the DESY II facility. The testbeam was to be one week long, with a one-week preparation period. The primary goal for DQM4hep in this testbeam was to establish hitmaps of the calorimeter. This would give a visual representation of the layers, allowing identification of the beamspot, allowing dead or miscalibrated channels to be identified visually.

Creating a hitmap for the AHCAL was nontrivial, as the information coming from the data acquisition device and stored in LCIO format did not encode location. Each channel was instead identified by its "electronics number" – a combination of the ChipID of the board the channel was located on, and the number of the channel on that board.

Each layer was formed of four boards (each one with a ChipID), each of which contained sixteen layers. The orientation of the boards, which boards were in a layer, and the order of the layers in the stack were all changeable, so an additional requirement for the

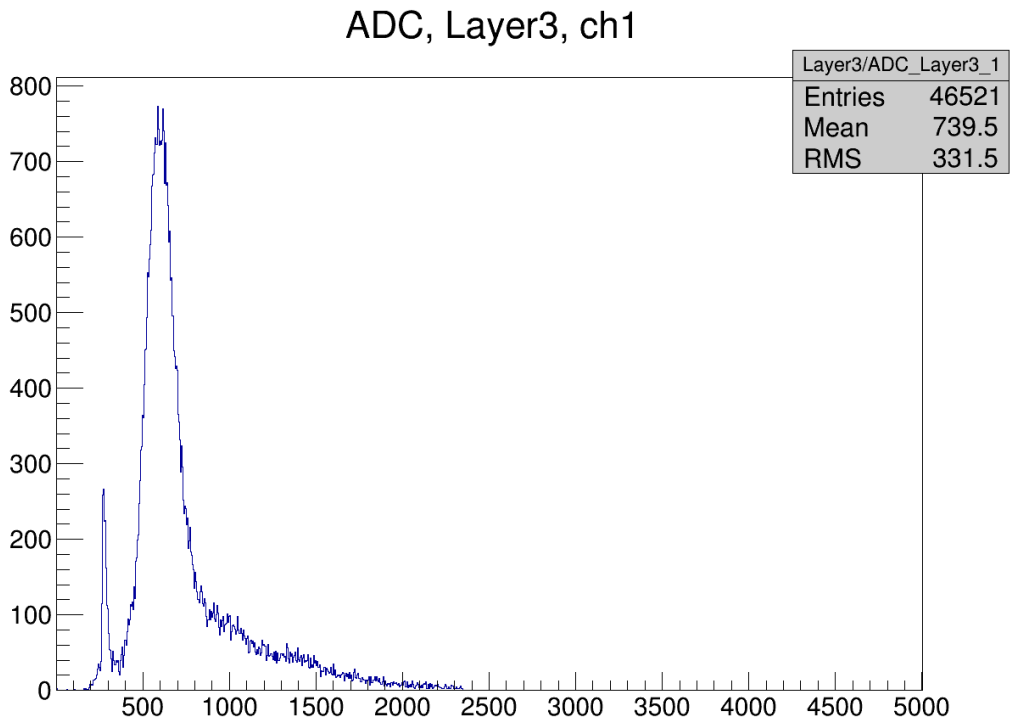

Figure 3.4: Histogram produced by `AHCALRawModuleChannel` showing an ADC spectrum of a single channel for one run. The pedestal can be seen at approximately 300 and the MIP peak at around 650.

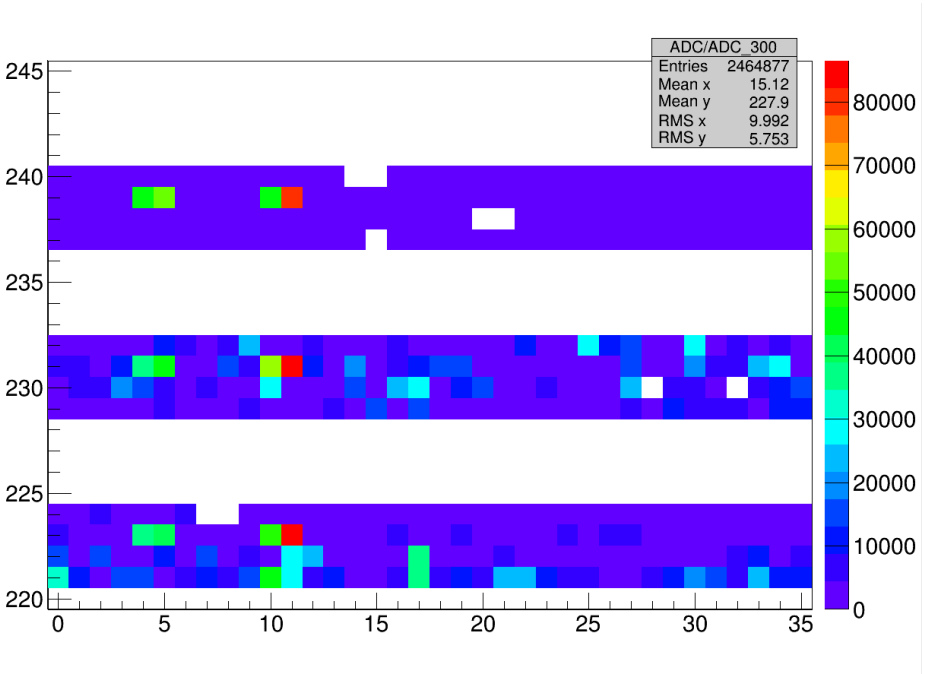

Figure 3.5: Histogram produced by `AHCALRawModuleGlobal` showing all ADCs exceeding 300 over a single run. Dead or nonresponsive channels are seen as white squares. The horizontal gaps are due to the fact that some ChipIDs were not present.

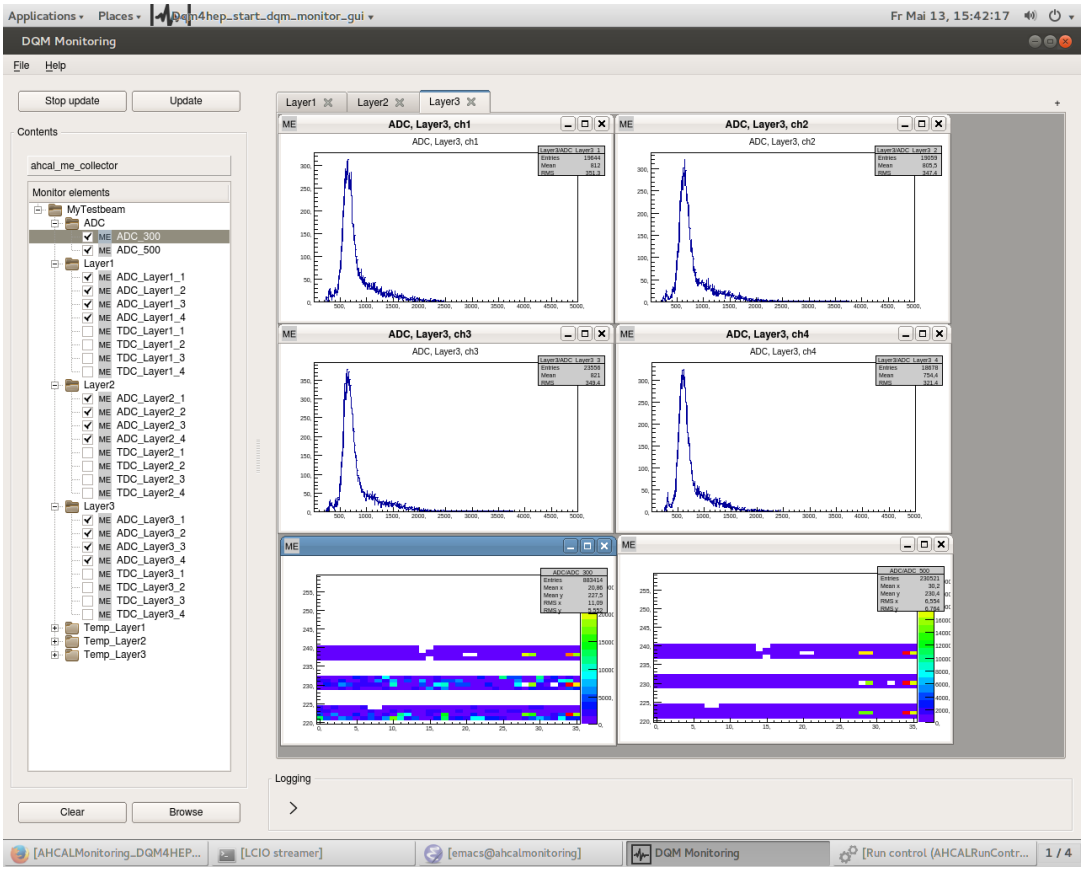

Figure 3.6: A full screencapture from the May 2016 testbeam showing the monitoring interface in use during power pulsing tests, displaying a MIP scan.

hitmap function was that it could take an external geometry file that could be changed or automatically generated.

DQM4hep has internal functions for parsing XML data, so an XML file was chosen as the format to store the geometry data. By making this an XML file, it avoided hard-coding the geometry into the framework itself, allowing the geometry data to be changed at runtime. XML also has the benefit of being human-readable. The AHCAL team already used an internal format for the geometry file for offline processing after testbeams, which was constructed and converted into different formats via a shell script, so this could be replicated for the DQM4hep XML file.

For analysis modules needing geometry, the XML file was given as a required parameter for the steering file, which then built a C++ map of the correspondence between electronics number and (i, j, k) co-ordinates of each channel in memory during initialisation. Then a function called `electronicsToIJK` was written that took the electronics number as the argument and returned the position of the channel in geometric co-ordinates. A further function was written, called `IJKToElectronics`, that performed the opposite operation,

but this was not used.

Using these new functions, another analysis module called `AHCALHitmap` was written that created a two dimensional histogram, with each bin representing a channel on the x and y axes for a single layer. This histogram was then filled with with the ADCs of that channel for the whole event, producing a hitmap.

The analysis modules for creating hitmaps were prepared ahead of the testbeam, and used extensively. They were used to identify channels that were dead or nonresponsive, as well as to confirm already-known nonresponsive channels.

3.3 May 2017 testbeam at CERN SPS

During May 2017, testbeam time at the CERN Super Proton Synchrotron (SPS) facility was used for further tests for the AHCAL. One of the goals for this testbeam was to evaluate the performance of the power pulsing feature in magnetic fields up to 1.5T. The process of manufacturing the detector layers and boards was being automated, and a larger number of layers were available for this testbeam, so it also presented a way to test using a larger number of channels than before.

Part of the programme for the testbeam was to use the electron and muon beams available at the SPS for calibrating the newly-produced layers, as well as doing an energy scan with a pion beam.

The monitoring with DQM4hep in this testbeam included analysis modules that monitored the individual channels of all 40 large layers, as well as producing hitmaps of several types, such as unweighted, ADC-weighted, and near-pedestal. Standalone modules monitoring the temperature of the detector hardware was also used.

At this point in the testbeam process, the online monitoring system with DQM4hep had matured, partially due to the data format of the detector having been fixed for some time. Experience with running, using, and modifying the analysis modules had been disseminated throughout the team, and team members wrote their own analysis modules for producing plots.

Because of this, during the testbeam DQM4hep was used as intended – as a tool for shifters to use to diagnose and troubleshoot problems with the beams or detectors. A specific expert on DQM4hep was not necessary, as enough people operating the testbeam understood DQM4hep to be able to modify analysis modules on the fly according to their needs.

For example, [correlation plots].

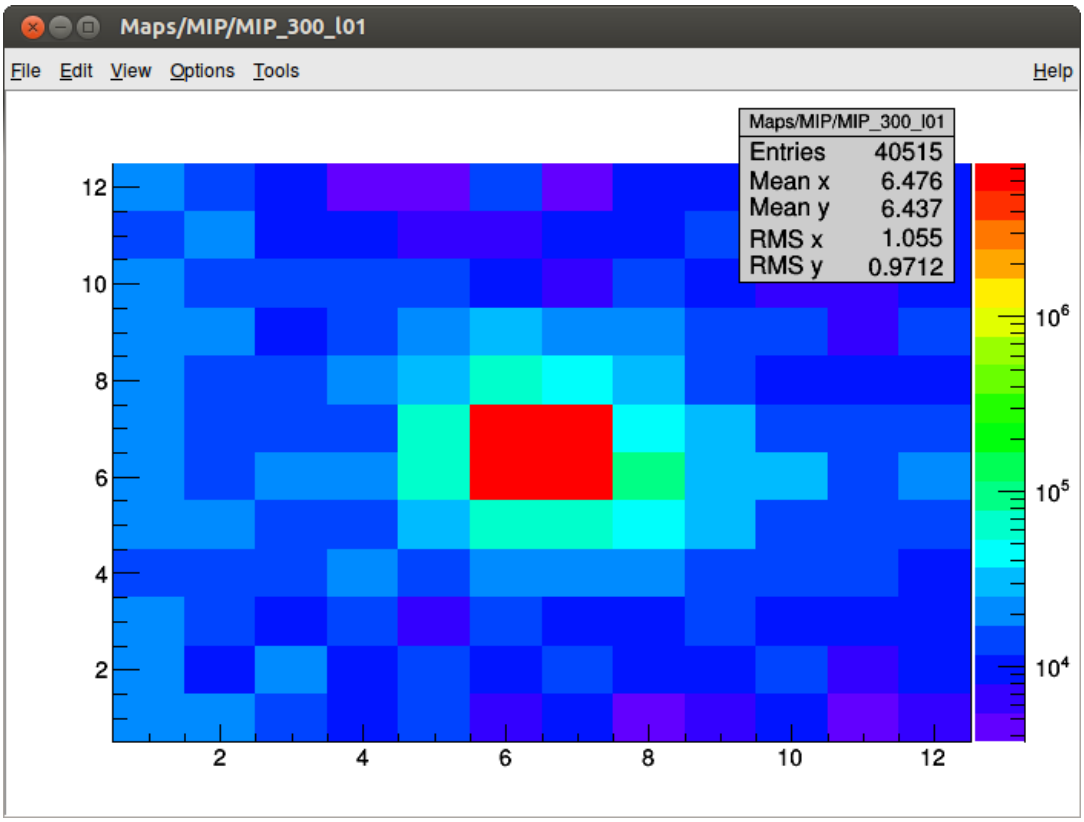

Figure 3.7: A hitmap for a single layer during the July 2016 testbeam, showing all channels with ADCs higher than 300. The beamspot is clearly visible in the centre.

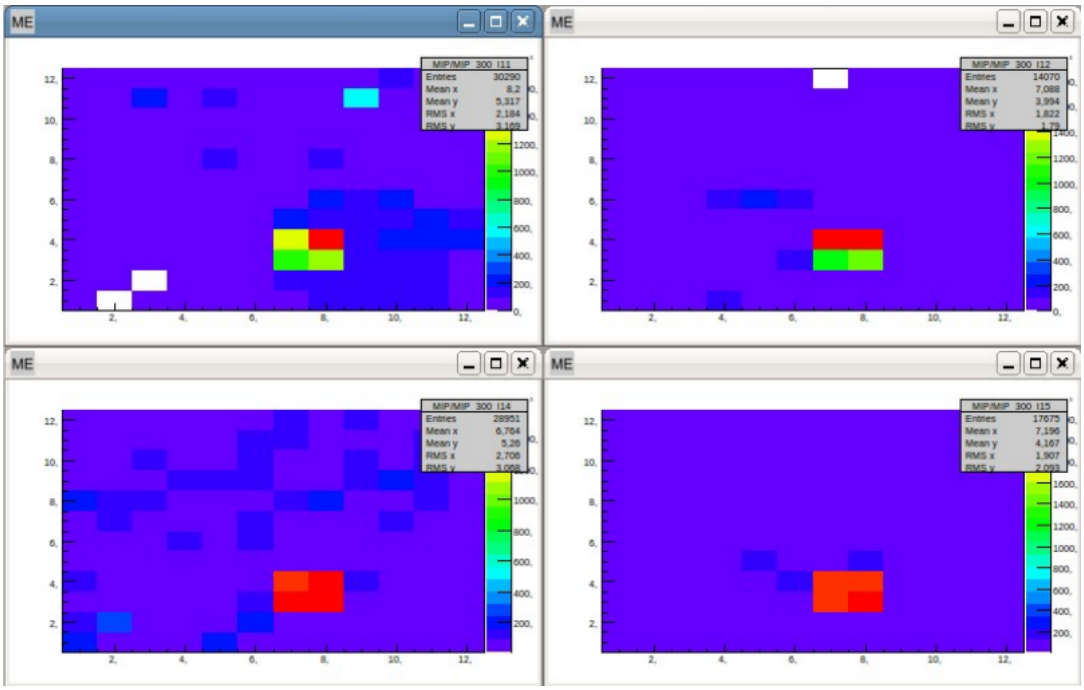

Figure 3.8: A collection of hitmaps for four layers in the stack during the July 2016 testbeam. The beamspot is visible, as are several dead or unresponsive channels in the top-left and top-right hitmaps.

Figure 3.9: The CALICE-AHCAL being unloaded during installation at the CERN SPS during the May 2017 testbeam.

Figure 3.10: The CALICE-AHCAL being moved into position by the cranes in the CERN SPS testbeam area.

Figure 3.11: An overview of the beam hall at the CERN SPS, with the solenoid the CALICE-AHCAL was placed within visible in the centre-right of frame.

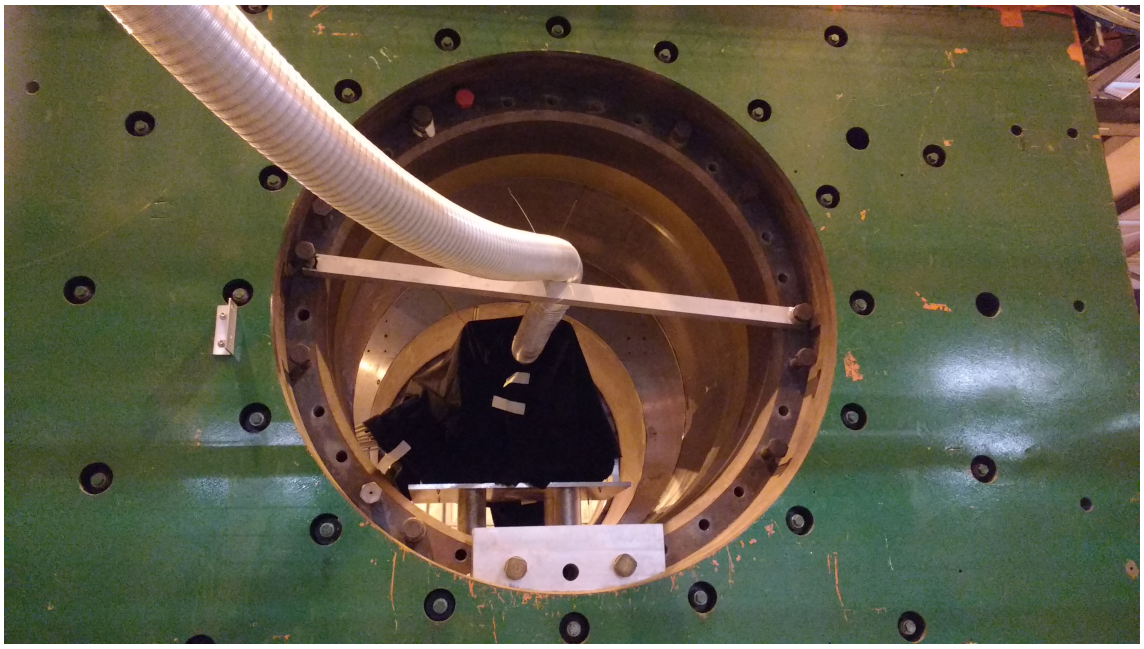

Figure 3.12: The CALICE-AHCAL placed inside the solenoid, covered with black fabric to block ambient light from the SiPMs.

Chapter 4

IDEA testbeams

[new quote split
over two lines]

[author]

While DQM4hep was intended from the beginning as a generic tool, it was developed largely within the AIDA-2020 collaboration, which also promoted a variety of standards and guidelines for data acquisition devices, data formats, etc. In addition to this, it was also only tested on calorimeter-type detectors within the CALICE collaboration. To be sure that DQM4hep was truly generic – capable of adapting to *any* detector – it was necessary to test it on a wider variety of detectors outside of the AIDA-2020 and CALICE collaborations.

An ideal opportunity arose for this in the form of the IDEA testbeam. This testbeam was run by a collaboration of Italian and American universities working on four different particle physics detectors of different types, intending to run a single combined testbeam of all four instruments at the CERN SPS in September 2018. DQM4hep was offered as a possible unified monitoring solution that could integrate information from all four detectors into the same tool. This provided convenience for the teams operating the testbeam and detectors, and an extremely valuable opportunity to test DQM4hep out of its established operating range.

4.1 Introduction

The combined testbeam took place between in September 2018 at the CERN SPS beamline facility. The testbeam itself lasted a week, from the 5th-12th September, with a preparation period of one week before this for installation, calibration, etc. [...]

Detectors at the combined testbeam

The combined testbeam comprised four separate dectors: a calorimeter, a muon detector and preshower, a drift chamber, and a silicon photomultiplier. One of the biggest challenges involved in the testbeam was operating these four different detectors [...]

RD52 calorimeter

The Dual Readout Method (DREAM) calorimeter, also called the RD52 calorimeter, is a dual-readout calorimeter built, developed and tested by the RD52 collaboration by researchers based in universities at Gagliari, Cosenza, Pavia, Pisa, Rome, Iowa State and Texas Tech.

The aim of the calorimeter is to use both the Čerenkov and scintillator techniques simultaneously (hence "dual-readout") to improve calorimetry, especially using calorimeters to measure four-vectors of jets and single hadrons, which suffer a reduced precision compared to electrons and photons. By comparing signals from Čerenkov light and scintillator light, the electromagnetic shower fraction can be measured on an event-by-event basis, eliminating the effects of fluctuations.

Muon chamber and preshower

[...]

Silicon photomultiplier GEM

[...]

Drift chamber

[...]

Ancillary detectors

4.2 Monitoring

[...]

Existing monitoring within the DREAM collaboration could produce accurate histograms from raw data using ROOT, creating plots of the energy spectra of each detector

channel per event, along with [...]. This facility was reproduced in DQM4hep quickly using for-loops in both the C++ code and XML steering files, allowing this to be done with comparatively little code.

4.2.1 File readers

Writing file readers for the different detectors meant first understanding the structure of their data and file types. Once data has arrived from the data acquisition device, each of the detectors wrote to a different 'raw' format. However, the RD52 calorimeter, drift chamber, and muon and preshower all produced ROOT ntuple files as part of their data acquisition process, in addition to different file formats. It was decided to use these ROOT ntuples as the file format to read into DQM4hep, as due to support for ROOT within the framework, reading data from ROOT ntuples is simpler.

For each of these three instruments a file reader was developed that walked through the ROOT trees, extracting data from the leaves event-by-event, then converted it to DQM4hep's inbuilt `GenericEvent` format. Choosing to make them into GenericEvents meant that there was no need to implement an additional event type, simplifying the connection between the file readers and analysis modules. The GenericEvent type was also easily suited to the type of data, as the majority of the data were series of numbers, which were easily converted into C++ vectors to store in the GenericEvent.

An additional motivation for using the ROOT ntuple as the data source was that some of the file type, notably the drift chamber, had not finalised their data structure at the beginning of the testbeam. Using the higher-level structure provided by a ROOT file would be easier to change in the future than reading in data in a binary, hex, text or CSV format.

For the silicon photomultiplier GEM data, the "raw" data format was a text file, containing an XML header followed by a large amount of data in Comma-Separated Values (CSV). This file could be loaded directly into DQM4hep, the XML header separated and parsed with DQM4hep's internal XML parsing libraries, and the remaining data parsed. The comma-separated values could be easily parsed using the `dqm4hep::core::tokenize` function, which takes a string, a delimiter, and a vector, and parses the string into values separated by the delimiter, loading them into the vector. This made extracting the GEM data extremely simple, even in this format. It also allowed the parameters in the XML header, which included run numbers and physical information of the detector, to be passed into DQM4hep and the analysis module, using the `core::Run` object type.

4.2.2 Analysis modules

During the course of the testbeam, four analysis modules were developed, one for each detector. To begin with, these were 'dummy' modules – analysis modules that receive events then do nothing. Dummy modules like this are required to run file readers offline, but are simple to create as they can be produced from a template with minimal changes.

Once the file reader for the RD52 calorimeter was complete, the dummy analysis module for this detector was then changed into a functional module, `RD52MainModule`, to produce plots from the data. The RD52 was chosen as it was the most data-rich of the detectors in the testbeam, and also contained the ancillary detectors, which allow particle selection efficiencies to be studied.

During the course of the testbeam, the `RD52MainModule` was developed to read in and arrange data from the ROOT ntuples and arrange them into distinct vectors for the ADCs, TDCs, pedestals, and ancillaries. Following this, pedestal subtraction of all ADC and TDC channels was completed. Then the malfunctioning tiles and electronics that meant data had to be re-routed through other available channels was integrated so that the ADC information in the analysis module represented the full state of the detector was completed.

There was insufficient time at the testbeam to develop the monitoring more complex than this, but further development continued offline with saved data from the testbeam to continue the proof-of-concept. This will be discussed in more detail in the following section.

4.3 Results

Monitoring within DQM4hep was able to replicate all the existing monitoring solutions, combining the monitoring tools for all four detector systems into one framework. [...] However due to time constraints, more complex monitoring such as online processing was not implemented during the testbeam. [...]

Following the testbeam, it was decided that DQM4hep could be used to perform some offline analysis functions in addition to the work at the testbeam, as a further proof of its ability to do this at testbeams. [...]

Several analysis goals were outlined to attempt to implement in DQM4hep. These were performing the tower ADC calibration, the ADC to energy calibration, and particle selection efficiencies.

4.3.1 Tower ADC calibration

[...]

Due to a large number of tasks for setting up the testbeam, performing a calibration of the individual towers' high voltages was not possible, so the calorimeter ADCs were not calibrated to each other [?]. In order to fix this, [...]

Two sets of calibration runs were performed. The first set covered Towers 1-29 using an 80 GeV secondary beam (π and e^-), with the beam pointed at each tower in sequence for 29 runs. The second set used a 20 GeV electon beam, covering Towers 30-36 plus Tower 15. Tower 15 recieved runs in both sets in order to calibrate the two sets to each other.

[...] The run numbers corresponding to each tower is given in Table 4.1.

Tower	**Run No.**	**Tower**	**Run No.**
1	12545	16	12526
2	12556	17	12567
3	12558	18	12633
4	12560	19	12591
5	12601	20	12612
6	12638	21	12530
7	12598	22	12528
8	12514	23	12569
9	12518	24	12639
10	12521	25	12610
11	12600	26	12609
12	12636	27	12607
13	12539	28	12604
14	12628	29	12602
15	12512		

Table 4.1: Table of the run numbers and corresponding tower numbers for the calibration runs.

The process for calibrating the towers was to make a histogram of the ADCs registered in a tower, only in the run where the beam was centered on them, in order to find the mean and standard deviation. These histograms were then combined to show the overall

responses of the entire set of calibration runs, which visualises the differences between the Towers. These can be seen in Fig. 4.1.

Since Tower 15 was present in both runs, this was chosen as the reference, and all the other towers were given a coefficient that leveled their mean ADC to the same value as Tower 15, in both sets of calibration runs. The ADC values for each tower are then multiplied by their calibration coefficient. [...] [Difference between scintillator and Cherenkov] [...] The calibration coefficients are shown on Table 4.2.

Tower	**Coefficient**	**Tower**	**Coefficient**
1	x	16	x
2	x	17	x
3	x	18	x
4	x	19	x
5	x	20	x
6	x	21	x
7	x	22	x
8	x	23	x
9	x	24	x
10	x	25	x
11	x	26	x
12	x	27	x
13	x	28	x
14	x	29	x
15	x		

Table 4.2: Table of tower numbers and their calibration coefficients.

4.3.2 ADC to energy calibration

[...] It was known from prior testbeams that the response of the calorimeter was linear with ADC, therefore the energy calibration could be extrapolated from a single datapoint. [...]

4.3.3 Particle selection efficiencies

An important result for the calorimeter was determining the efficiency with which it could determine the beam composition and select for certain types of particles based on

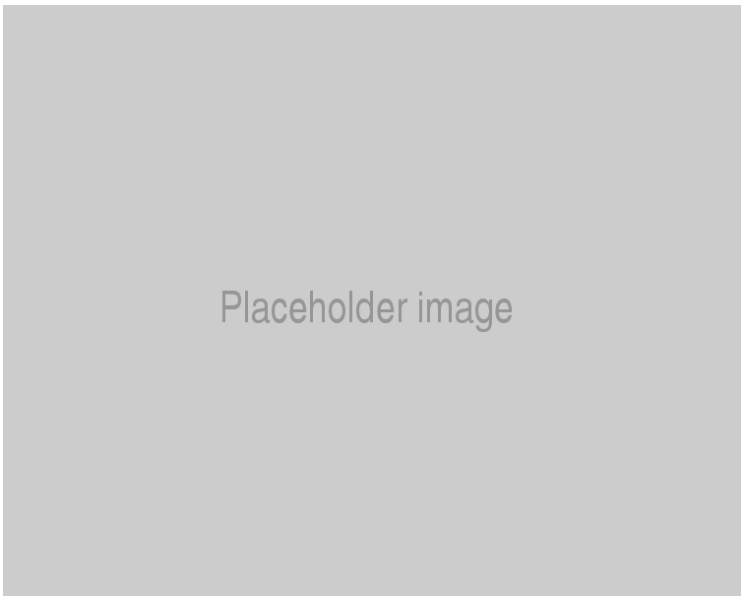

Figure 4.1: The plot of ADCs for each tower before calibration.

kinematic properties. The calorimeter was designed with a set of ancillary detectors to help discriminate between particle of different types. These provide a first selection of different particles, allowing us to use kinematic properties from the calorimeter itself to perform a second particle selection, and then compare the two.

The two ancillary detectors used for particle selection are the muon trigger and ...]

[...]

Using the calorimeter

[...]

One of the first steps is using the RD52 calorimeter data to find the energy ratio R for each event:

$$R = \frac{E_1}{\sum_{i=1}^{n} E_i}$$

where E_i is the energy of the i^{th} most energetic channel in the event and n is a nonzero integer. The choice of n [...]. Once the ratio R is calculated, a plot can be made of E_{total} vs. R for an entire run that shows separation of electrons from muons and pions – see Fig. 4.3.

An appropriate cut can be used to select electron events. [Information about the cut]. Adding in the information from the RD52's muon trigger, muons and pions can then also be separated. Using both the cut and the muon trigger, we can thus produce spectra for each individual type of particle in the run.

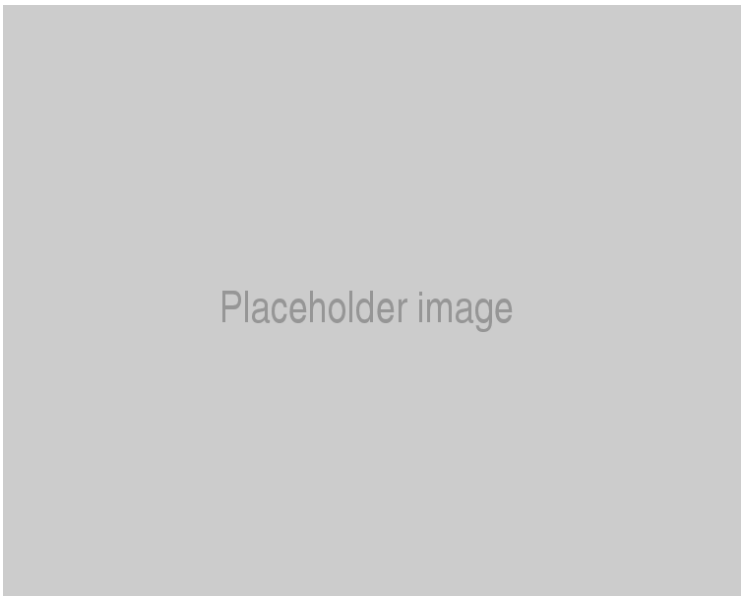

Figure 4.2: Comparison of the calibration plots for the ADCs before and after calibration.

[...]

The main goal of the data analysis is to characterise the response of the detector, and to measure the selection efficiencies of the detectors to various particle types. Once this is done, this will also give us a detailed account of the beam composition during each of the runs, which can be used for further work.

In order to do this, several ways to select for different particle types are necessary. The first way was using the preshower detector and muon trigger, which are both designed to discriminate between electrons and muons (respectively), with a high selection efficiency. These are used to create "reference" samples, [...]

The second way is to perform a kinematic selection using variables from the calorimeter. This is done using the E vs. R plot (normalised for beam energy) to select a region corresponding to a certain particle type. For example, the plot below shows this plot for Run [xxx] (X GeV hadrons) with the regions corresponding to hadrons and electrons highlighted. These regions overlap, meaning that attempting to select for hadrons using an ellipse around that region will also result in a non-insignificant number of electrons also being selected.

In order to perform a pure selection using the calorimeter, an extremely tight cut was used, focusing on the red spot at the centre of the hadron region. This ensures that the majority of events that pass the cut are hadrons, giving a high-purity selection. The purity of this selection can be assessed by using the appropriate preshower or muon trigger, excluding all non-hadron particles, and comparing the two numbers. If N_K is the number

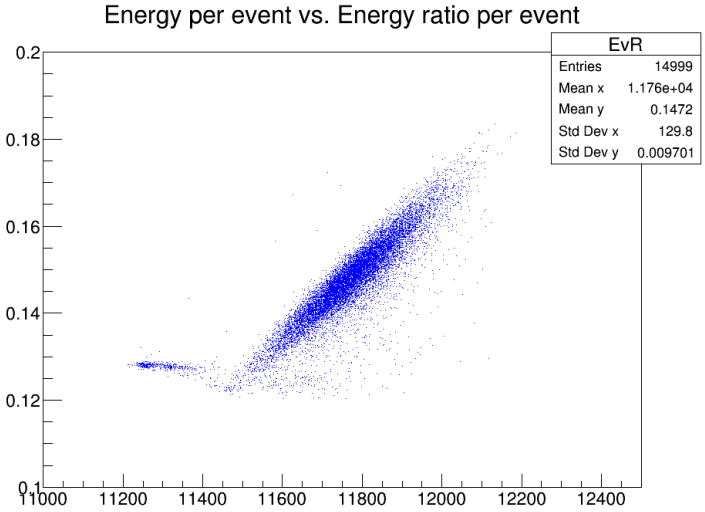

Figure 4.3: Plot of E against R for the secondary hadron beam with $n = 10$ (Run 12709).

of particles passing the selection with only the kinematic cut, and N_{K+T} is the number of particles passing *both* the kinematic cut and the triggers, then the selection efficiency for hadrons ϵ_{hadron} is given by:

$$\epsilon_{hadron} = \frac{N_{K+T}}{N_K}$$

This was then done with the same process for electrons and muons, to obtain the individual selection efficiencies.

Chapter 5

Physics studies for the Compact Linear Collider

Somewhere, something incredible is
waiting to be known.

Carl Sagan

One of the primary goals of future lepton colliders is to become "Higgs factories" – machines that can produce large numbers of Higgs bosons in a variety of final states, allowing the Higgs sector of the Standard Model to be probed with unprecedented accuracy and coverage.

One of the best way to examine the Higgs sector is via the production of Higgs bosons in association with top quarks. The coupling between these two particles is the strongest Higgs coupling in the SM, making it one of the easiest ways to examine the properties of the Higgs boson and the Higgs sector. This is the top-Higgs Yukawa coupling y_t.

An important process used to study the coupling between the Higgs and the top is the tth process:

$$e^+e^- \to t\bar{t}h$$

The production of two top quarks in association with a Higgs makes this an ideal probe for this coupling. The top quark is only permitted to decay via the $t \to W^+b$ process, but the decays of the W boson vary. W bosons can decay *hadronically* into a pair of one up-type quark and one down-type quark (e.g. $u\bar{d}$ or $c\bar{s}$). Or they can decay *leptonically* into a charged lepton and a neutrino of the same flavour (e.g. $e^-\overline{\nu}_e$ or $\mu^+\nu_\mu$). The total branching ratio of hadronic decays is 68%, and the total branching ratio of the leptonic decays is 32%. We denote a quark-antiquark pair as $q\overline{q}$, and a lepton-neutrino pair as $\ell\nu$.

Given that we focus only on events where the Higgs decays via the process $H \to b\bar{b}$, there are then three possible final states:

$$e^+e^- \to t\bar{t}h \to q\bar{q}q\bar{q}b\bar{b}$$

$$e^+e^- \to t\bar{t}h \to \ell\nu q\bar{q}b\bar{b}$$

$$e^+e^- \to t\bar{t}h \to \ell\nu\ell\nu b\bar{b}$$

Due to the branching ratios, the hadronic and semi-leptonic final states are the most common, and the leptonic final state occurs less than ten percent of the time. This means that the most common final states all heavily rely upon jets, meaning that in order to perform accurate analysis of this process, jet energy resolution and jet reconstruction is of primary importance. As discussed in Chapter 1, the detector concepts for future lepton colliders are designed to focus on jet energy resolution and jet reconstruction, and utilise particle flow to augment this.

This chapter describes an analysis performed to attain the uncertainty on the measurement of the top-Higgs Yukawa coupling at the proposed Compact Linear Collider (CLIC) experiment at a centre-of-mass energy of $\sqrt{s} = 1.4$ TeV, using the CLIC_SiD detector concept. This analysis focuses specifically on the hadronic channel, and when taken in combination with a similar analysis of the semi-leptonic channel performed by Yixuan Zhang[reference this]allows calculation of the precision on the top-Higgs Yukawa coupling.

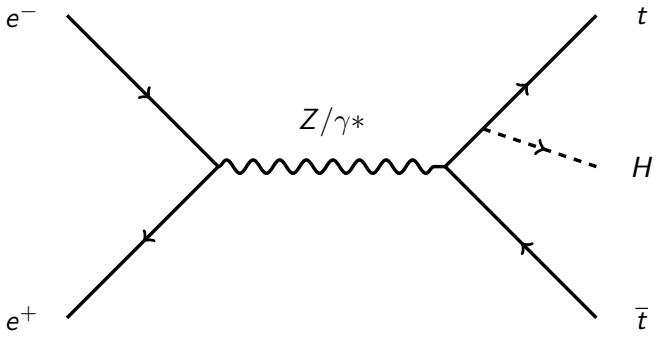

Figure 5.1: Feynman diagram of the tth event.

5.1 Physics generation and samples

The samples used for the study had been generated previously using ILCDIRAC. The physics generation was done predominantly in WHIZARD 1.95, using PYTHIA 6.422 for

the simulation of hadronisation and fragmentation. Due to constraints of WHIZARD, samples with final states with eight fermions were generated in PhysSim, also using PYTHIA. The mass of the Higgs boson was set at 125.0 GeV, and the mass of the top quark at 174.0 GeV.

All samples were simulated at $\sqrt{s} = 1.4$ TeV using unpolarised beams, assuming an integrated luminosity of 1.5 ab^{-1}. See Table 5.1 for a summary of all of the samples used.

Following the generation of the underlying physics events, the interactions of the particles with the detector must be simulated. This was done using the Geant4 toolkit, specifically the Mokka tool. Simulation of the signals that result from the interactions in the detector was done in Marlin.

At the end of this process, the Monte Carlo dataset is in a format identical to how information from the constructed detector would look. At this stage, the analysis can proceed as if the data were real. However, in addition to the simulated detector response, there is also the Monte Carlo truth information, which will allow correlation of the simulated detector response data with the "true" particles in the simulation itself. However, this is not used in this analysis.

ProdID	Process	Cross-section (fb)	Sample weight	Events in 1.5 ab^{-1}
2435	$t\bar{t}h$, 6 jets, $h \to b\bar{b}$	0.431	0.03	647
2441	$t\bar{t}h$, 4 jets, $h \to b\bar{b}$	0.415	0.03	623
2429	$t\bar{t}h$, 2 jets, $h \to b\bar{b}$	0.100	0.006	150
2438	$t\bar{t}h$, 6 jets, $h \not\to b\bar{b}$	0.315	0.02	473
2444	$t\bar{t}h$, 4 jets, $h \not\to b\bar{b}$	0.303	0.02	455
2432	$t\bar{t}h$, 2 jets, $h \not\to b\bar{b}$	0.073	0.004	110
2450	$t\bar{t}Z$, 6 jets	1.895	0.1	2843
2453	$t\bar{t}Z$, 4 jets	1.825	0.1	2738
2447	$t\bar{t}Z$, 2 jets	0.439	0.03	659
2423	$t\bar{t}b\bar{b}$, 6 jets	0.549	0.03	824
2426	$t\bar{t}b\bar{b}$, 4 jets	0.529	0.03	794
2420	$t\bar{t}b\bar{b}$, 2 jets	0.127	0.008	191
2417	$t\bar{t}$	125.8	1.5	203700

Table 5.1: Table of all signal and background samples used for this analysis. The first two rows are the signal samples, the others are backgrounds.

5.1.1 Detector model

The samples for this study were simulated using the CLIC_SiD detector model, built virtually within GEANT4. This is the version of the Silicon Detector (SiD) as described in the Compact Linear Collider Conceptual Design Report (CDR).

The CLIC_SiD uses an all-silicon construction, with a design optimised for the needs of particle flow. It uses high-resolution vertex detectors and trackers, both based on silicon pixels and strips. The ECAL and HCAL use predominantly tungsten as the absorbing material, with steel used instead in the endcaps. These detectors are placed within a 5T solenoidal magnetic field, with an iron flux return yoke that is instrumented to act as a muon detector and tail catcher.

A more in-depth description of the SiD detector concept can be found in Chapter 1.

5.2 Analysis strategy

The tth event has three possible final states, defined by the decay channel of the W bosons produced by the top quarks. The decay of a top quark will always produce a W boson and a bottom quark, while the W boson can decay one of two ways. There are thus three possible combinations that can be seen in Table 5.2.

Decay Channel	Final State	No. of leptons	Branching Ratio
Fully hadronic	$q\bar{q}q\bar{q}b\bar{b}$	0	46%
Semi-leptonic	$\ell\nu q\bar{q}b\bar{b}$	1	45%
Fully leptonic	$\ell\nu\ell\nu b\bar{b}$	2	9%

Table 5.2

Given the low branching ratio compared to the other channels, the fully leptonic state is not used in this analysis. Extended Feynman diagrams of the fully hadronic and semi-leptonic final states are shown in Figs. 5.3 and 5.4

There is also another contribution to the tth final state in the form of *higgstrahlung*. This process is actually an $e^+e^- \to t\bar{t}$ event where the intermediate $Z/\gamma*$ radiates a Higgs boson *before* producing the $t\bar{t}$ pair. This means the final state is tth but no direct interaction between the top quark and Higgs has occurred (see Fig. 5.2). This means that the higgstrahlung process is not sensitive to the top-Higgs Yukawa coupling, and these events must be taken into account when calculating the total cross-section of the tth process

when trying to determine the coupling.

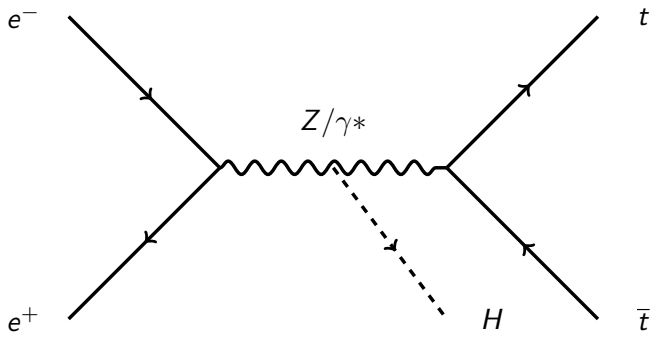

Figure 5.2: Feynman diagram of the higgstrahlung process.

5.2.1 Lepton identification

The first step of the analysis procedure is to distinguish between the hadronic and semi-leptonic final states. This is done by finding isolated leptons in the final state. If there are no final-state leptons it is classified as a hadronic event; if there is a single lepton in the final state it is classified as a semi-leptonic event. If there are two leptons in the final state it is classified as a leptonic event and discarded.

Finding the leptons that have been produced by the decay of the W boson is relatively simple, as leptons produced in this decay have a much higher energy than leptons produced in jets. In general, these leptons have energies in the 50-350 GeV range, which can be found from the reconstructed track energy. However there are additional properties of these leptons that can be used to improve their selection.

Leptons produced within jets will detected within the same region as the other particles from the same jet, in a high-occupancy region of the detector. Therefore, an isolated lepton detected in a low-occupancy region of the detector is more likely to have originated from the decay of a W boson, rather than from a jet.

The impact parameter (IP) is the distance between the primary vertex that produced a particle, and the closest approach of the particle's track. Electrons and muons produced from W boson decays have much lower IPs than other particles or tau decay products. The impact parameter can be considered in either the longitudinal (Z_0) or radial (d_0) components, which can be combined for a 3D IP:

$$R_0 = \sqrt{Z_0^2 + d_0^2} \tag{5.1}$$

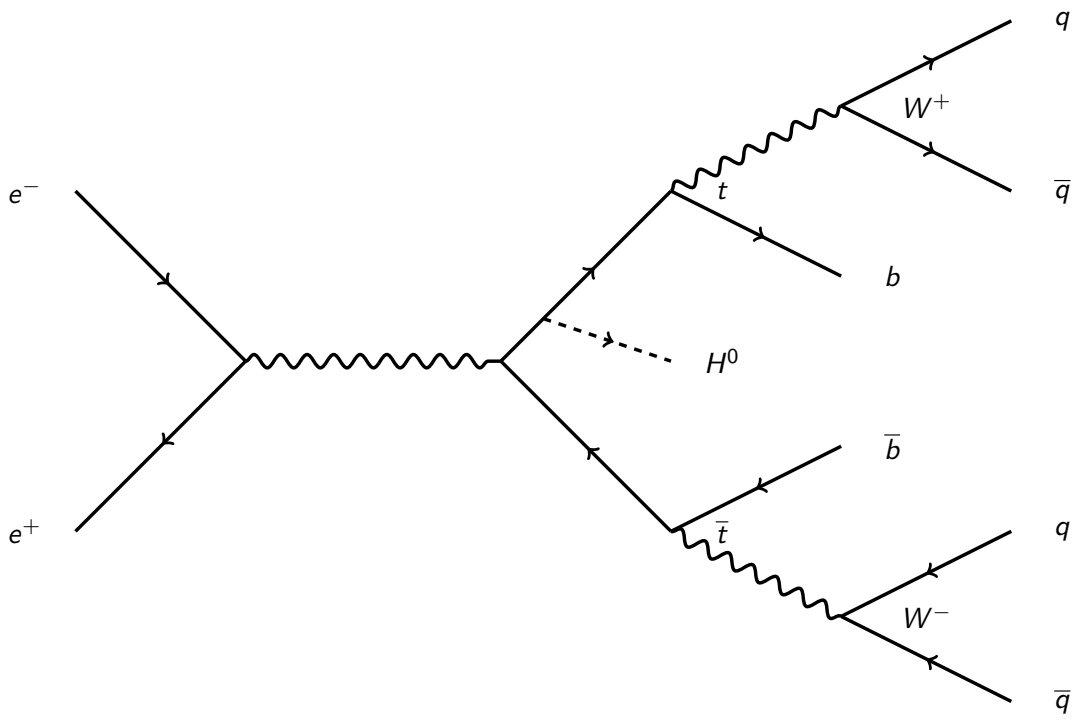

Figure 5.3: Extended Feynman diagram of the tth event, showing the fully-hadronic decay channel with the final state $q\overline{q}q\overline{q}b\overline{b}$, where q and $\overline{q}$ indicate a quark-antiquark pair.

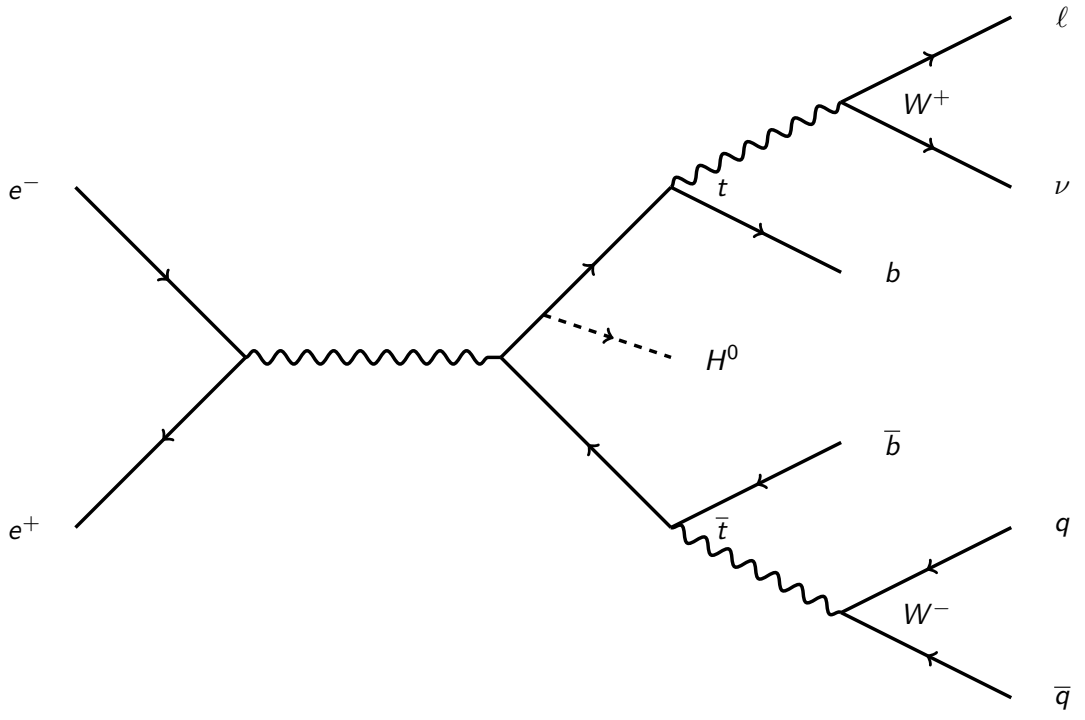

Figure 5.4: Extended Feynman diagram of the tth event, showing the semi-leptonic decay channel with the final state $\ell\nu q\overline{q}b\overline{b}$, where ℓ and ν indicate a lepton-neutrino pair.

Another property useful for distinguishing electrons and muons decaying from W bosons is the profile of their energy deposition in the calorimeters. We can define a ratio of the energy deposited in the electromagnetic calorimeter to the total energy deposited in the calorimeters:

$$R_{CAL} = \frac{E_{CAL}}{E_{ECAL} + E_{HCAL}} \tag{5.2}$$

Since electrons are contained within the ECAL, their value of R_{CAL} will peak at 1. Muons deposit their energy throughout both calorimeters, so will peak at approximately 0.2. Taus cannot be identified this way, as their decay products do not deposit predictable amounts of energy in the calorimeters.

IsolatedLeptonFinder

To perform the selections to find isolated leptons, the `IsolatedLeptonFinder` processor within MARLIN was used. This lepton finder processor uses criteria based on the properties described above to identify isolated leptons:

- Track energy greater than 15 GeV
- Each of d_0, Z_0 and R_0 greater than 0.05 mm
- R_{CAL} greater than 0.5, or in the range 0.05-0.3

The first cut on track energy already retains 96.5% of the reconstructed electrons and muons, while accepting only 9.6% of other particles. With the additional criteria, this becomes 87.3% isolated leptons and 0.4% other particles.

5.2.2 Tau identification

Taus cannot be identified directly due their extremely short lifetime (2.9×10^{-13}s), and must instead be identified from their decay products, which can vary significantly. Taus most commonly decay hadronically into a number of charged or neutral pions, or leptonically into a lepton-neutrino pair; in each case always also producing an undetectable tau neutrino.

Identification of taus was done using the `TauFinder` processor within MARLIN, which was adapted for this analysis. The criteria were determined using truth-matching the simulation data to find the best strategy to identify taus.

The processor looks at particles that may have decayed from taus, and forms a tau candidate based on their properties. First it creates a seed track with a p_T greater than 10 GeV, adding all particles within a 0.04 rad cone of the seed track to the tau candidate if every particle has $p_T > 2$ GeV and R_0 within the range 0.01-0.05mm. The reconstructed tau candidate must then be formed of an odd number of charged tracks, with an invariant mass of less than 1.5 GeV. This mass is slightly lower than the true tau invariant mass to account for missing energy in the form of the tau neutrinos. Then an isolation ring is defined in the region 0.04-0.24 rad around the tau candidate seed track. In order to be considered a tau, there must be fewer than five particles in the isolation ring, with a total energy of less than 5 GeV.

5.2.3 Reconstruction of Higgs, top and W boson candidates

Before jet clustering, any isolated leptons are removed from the event. The event is then clustered using the k_t algorithm with $R = 1.0$. The jet clustering algorithm is run in exclusive mode, forcing the event into a specific number of jets – in the case of the hadronic analysis, eight jets. Once this is complete, the jet particles are then reclustered using the Durham algorithm .

Following this, the properties of the jets are used to reconstruct candidates for the Higgs boson, top quarks, and W bosons. This is done by doing a calculation of the invariant masses of the tops, W bosons, and Higgs, resulting in a χ^2 value. By computing this χ^2 value for all possible combinations of jets and finding the combination that results in the minimum, the jets can be matched to which particles produced them.

$$\chi^2 = \frac{(m_{12} - m_W)^2}{\sigma_W^2} + \frac{(m_{123} - m_t)^2}{\sigma_t^2} + \frac{(m_{45} - m_W)^2}{\sigma_W^2} + \frac{(m_{456} - m_t)^2}{\sigma_t^2} + \frac{(m_{78} - m_H)^2}{\sigma_H^2} \quad (5.3)$$

where the numbered subscripts indicate the indices of the jets used to form the invariant masses m. A similar equation is used for the semi-leptonic channel, using only the 6 jets present in that process. The χ^2 values are extremely useful for selection, as events that are not tth, or are tth events without the fully hadronic decay channel, will have poor χ^2 values, as the jets the event was forced into do not represent physical jets.

5.2.4 Flavour tagging

The majority of the backgrounds for this analysis do not have four b-jets in the final state. Therefore reliable b-tagging is an important source of discriminating power.

For this analysis, b-tagging is done in the LCFIPlus package, using simulated samples of $e^+e^- \to qqqqqq$ where all quarks have the same flavour. The high number of jets in the final state means that the kinematic properties of the jets are similar to those in the analysis samples. Four Boosted Decision Trees (BDT) are trained for b-tagging, and four for c-tagging, trained using the properties of the jets. These BDTs then give b-tag and c-tag probabilities for each jet, which are used for event selection in the next step.

5.2.5 Event selection

The final event selection is done using Boosted Decision Trees (BDT), implemented in the Toolkit for Multi-Variate Analysis (TMVA). The BDTs for the hadronic channel are trained separately to the semi-leptonic channel. No pre-selection is used in this analysis. For the hadronic channel, the following variables are used:

- Reconstructed Higgs mass
- Number of reconstructed particles in the event
- Visible jet energy
- Missing transverse momentum (p_T)
- χ^2 of the jet grouping
- Event shape variables – thrust, sphericity, aplanarity
- Four highest b-tag values and their corresponding c-tags
- The cosine of the decay angle of the $h \to b\bar{b}$, and the cosine of the angles between the Higgs and each top quark
- The distance between the two closest jets, as defined by the Durham jet clustering algorithm
- The energy of the four lowest-energy jets
- The cosine of the angle of the two jets closest to the beam-axis

The BDTs use the sample split into two randomly. The first half is used for training the BDTs, and the other becomes the samples used by the BDT.

5.3 Results

In order to extract the top-Higgs Yukawa coupling y_t, the total cross-section of the tth process must be calculated. Using the output of the BDTs, the precision on the cross-section of the tth process using just the hadronic channel is 9.6%. A similar analysis by Yixuan Zhang found the precision for the semi-leptonic channel to be 11.1%. Thus the combined uncertainty for the cross-section of both decay channels is:

$$\Delta\sigma = 7.30\%$$

When extracting the top-Higgs Yukawa coupling y_t from the total cross-section, the contribution of the higgstrahlung diagram (Fig. 5.2) must be taken into account, as this is a tth final state that has no vertex between the top and the Higgs and thus is not sensitive to the coupling.

In order to get the uncertainty on the coupling, the uncertainty on the cross-section has to be multiplied by a factor that was calculated using next-to-leading order (NLO) corrections to QCD, the initial state radiation, and beamstrahlung[reference this]. This factor is found to be 0.503, slightly higher than the 0.5 that would be expected if the higgstrahlung diagram were not included.

Thus the uncertainty on the top-Higgs Yukawa coupling is:

$$\frac{\Delta g_{tth}}{g_{tth}} = 0.503\frac{\Delta\sigma(t\bar{t}H)}{\sigma(t\bar{t}H)} = 3.86\%$$

These results were contributed to a paper that summarised the top physics potential for CLIC at $\sqrt{s} = 1.4$ TeV, which was submitted to CERN's European Strategy Update in 2019. See [reference this].

Chapter 6

Discussion and Conclusions

What we know is really very, very little
compared to what we still have to know.

Fabiola Gianotti

Bibliography

[1] Assessment of the revised plan of international linear collider project (executive summary). http://www.scj.go.jp/ja/info/kohyo/pdf/kohyo-24-k273-en.pdf, 2018. Accessed: 2019-02-07. 8

[2] R Ete, T Coates, and A Pingault. Dqm4hep: a generic data quality monitoring framework for hep. Technical report, 2018. 20

[3] Dqm4hep user manual. https://dqm4hep.readthedocs.io/en/latest/. Accessed: 2019-02-07. 47

[4] Dqm4hep doxygen pages. https://dqm4hep.github.io/dqm4hep-doxygen/doxygen/dqm4hep/master/index.html. Accessed: 2019-02-07. 47

[5] Felix Sefkow and Frank Simon. arxiv: A highly granular sipm-on-tile calorimeter prototype. Technical report, 2018. 51

Acronyms

ADC Analogue to Digital Conversion. 31, 32, 34, 37, 54, 56, 57, 59, 60, 66–70

AHCAL Analogue Hadronic Calorimeter. 9, 51–53, 56, 59, 61, 62

ATLAS A Toroidal LHC Apparatus. 5

BDT Boosted Decision Trees. 6, 80, 81

BIF Beam Interface. 52, 53

BSM Beyond the Standard Model. 1, 2, 4, 6

CALICE Calorimeter for Linear Collider Experiment. 51, 52, 61–63

CDR Conceptual Design Report. 13, 14, 75

CEPC Circular Electron Positron Collider. 2, 14, 15

CERN *Conseil européen pour la recherche nucléaire*. 7, 12, 13, 17, 19, 25, 52, 59, 61–63, 81

CLIC Compact Linear Collider. 1, 12, 13, 73, 75, 81

CSV Comma-Separated Values. 65

DAQ Data Acquisition. 17

DESY *Deutches Elektronen-Synchrotron*. 7, 17, 25, 52

DQM Data Quality Monitoring. 18, 19, 38

DQM4hep Data Quality Monitoring for High-Energy Physics. 19–26, 28, 30, 32, 38, 39, 41, 44, 47, 49, 51–54, 56, 58, 59, 63, 65, 66

DREAM Dual Readout Method. 64

ECAL Electromagnetic Calorimeter. 9, 11, 75, 78

EDM Event Data Model. 20

EUDAQ EU [?] Data Acquisition. 52

FCC Future Circular Collider. 2, 13, 14

FLC *Forschung mit Lepton Collidern*. 52

GEM Gas Electron Multiplier. 65

GUI Graphical User Interface. 23, 25

HCAL Hadronic Calorimeter. 9, 11, 75

HL-LHC High Luminosity Large Hadron Collider. 1

IDEA ?IDEA?. 63

ILC International Linear Collider. 1, 7–9, 11, 13, 15

ILD International Large Detector. 9–11, 14

IP Impact Parameter. 76

IP Interaction Point. 7

JINR Joint Institute for Nuclear Research. 7

JSON Javascript Object Notation. 42

LCIO Linear Collider Input/Output. 21, 53

LEP Large Electron Positron Collider. 13

LHC Large Hadron Collider. 1, 3, 4, 6, 7, 12, 13

MAPS Monolithic Active Pixel Sensor. 9

MIP Minimum Ionising Particle. 53, 57, 58

PFO Particle Flow Object. 8

QCD Quantum Chromodynamics. 5, 6, 81

qtest quality test. 39, 40, 42–46

RF radio frequency. 12

RPC resistive plate chambers. 9

SCRF Superconducting Radio Frequency. 7

SDHCAL Semi-Digital Hadronic Calorimeter. 9

SiD Silicon Detector. 9, 11, 75

SiPM Silicon Photomultipliers. 9, 51, 55, 62

SiWECAL Silicon Tungsten Electronic Calorimeter. 51, 53

SM Standard Model. 1–3, 5, 6, 72

SPPC Super Proton Proton Collider. 14

SPS Super Proton Synchrotron. 17, 52, 59, 61–63

SUSY Supersymmetry. 6

TDC Time to Digital Conversion. 53, 54, 66

TDR Technical Design Report. 7

TMVA Toolkit for Multi-Variate Analysis. 80

TPC Time Projection Chamber. 9, 14

XML Extensible Markup Language. 27, 32, 37, 42, 43, 45, 48, 49, 56, 58, 65