PAPER • OPEN ACCESS

ACTS: from ATLAS software towards a common track reconstruction software

To cite this article: C Gumpert *et al* 2017 *J. Phys.: Conf. Ser.* **898** 042011

View the article online for updates and enhancements.

ACTS: from ATLAS software towards a common track reconstruction software

C Gumpert[1], A Salzburger[1], M Kiehn[2], J Hrdinka[1,3] and N Calace[2] on behalf of the ATLAS Collaboration

[1] CERN, Geneva, Switzerland
[2] Section de Physique, Université de Genève, Geneva, Switzerland
[3] Vienna University of Technology, Vienna, Austria

E-mail: `christian.gumpert@cern.ch`

Abstract. Reconstruction of charged particles' trajectories is a crucial task for most particle physics experiments. The high instantaneous luminosity achieved at the LHC leads to a high number of proton-proton collisions per bunch crossing, which has put the track reconstruction software of the LHC experiments through a thorough test. Preserving track reconstruction performance under increasingly difficult experimental conditions, while keeping the usage of computational resources at a reasonable level, is an inherent problem for many HEP experiments. Exploiting concurrent algorithms and using multivariate techniques for track identification are the primary strategies to achieve that goal.

Starting from current ATLAS software, the ACTS project aims to encapsulate track reconstruction software into a generic, framework- and experiment-independent software package. It provides a set of high-level algorithms and data structures for performing track reconstruction tasks as well as fast track simulation. The software is developed with special emphasis on thread-safety to support parallel execution of the code and data structures are optimised for vectorisation to speed up linear algebra operations. The implementation is agnostic to the details of the detection technologies and magnetic field configuration which makes it applicable to many different experiments.

1. Introduction

The reconstruction of trajectories of charged particles, in the following referred to as *track reconstruction*, is one of the the most complex and CPU consuming parts of event reconstruction in high energy physics experiments. In particular for hadron colliders, track finding and fitting has to operate with very complex input, as the number of instantaneous hadron collisions, also called *pile-up*, is usually kept high in order to reach a high integrated luminosity. Most concepts of track reconstruction currently applied at the LHC have historically evolved, from global pattern recognition approaches to the combinatorial Kalman filtering [1–4]. The current software used by the LHC experiments is partly evolved from software written during their design phase in the late 1990s and deployed before the LHC startup in 2009. In the meantime, the event pile-up increased significantly and has put pressure on the CPU consumption of ATLAS [5] and CMS [6]. Both experiments have undergone recent software campaigns in order to optimise their event reconstruction software where a speedup factor of up to 5 could be achieved [7]. However, future scenarios such as the High-Luminosity LHC or the Future Circular Collider project will require a shift in paradigm of track reconstruction software in order to cope with event pile-up of up to 1000 instantaneous collisions. New concepts have to be developed and the code has to be prepared for modern CPU architectures that allow many parallel threads and long vector registers for internal parallelisation. The *A Common Tracking Sofware* (ACTS) project is an attempt to encapsulate the well-tested and high-performing software from the LHC experiments and prepare its modules for modern computing architectures and long term code maintenance. Its core design is based on the ATLAS common tracking software [8] that was established as a technology agnostic tracking software which was then specialised in the two ATLAS tracking systems, the Inner Tracker and the Muon System. This principle, i.e. fully decouple the tracking software from detector specifics, has been kept. ACTS also aims to be a research and development platform for algorithm development for future track reconstruction and is independent from any event processing framework. It is designed to have minimal external dependencies and all modules are tested for thread-safety and code compliance with the C++14 standard.

2. Tracking geometry

The ACTS reconstruction geometry follows the concepts of the ATLAS Tracking Geometry library [9]. Based on a `Surface` class, layers and volumes are built either as extensions of surfaces or as a compound of bounding surfaces, respectively. Surfaces exist in different flavours: planar surfaces with a local Cartesian coordinate system, disc surfaces with a local polar coordinate system, line-like surfaces for describing straw detectors and perigee representation as well as cylinder and cone surfaces with appropriate cylindrical coordinates. The shapes of surfaces are given by different shape classes. Tracking layers extend surfaces with containers of measurement surfaces in order to structure the readout elements. These layers are contained by volumes which are themselves built by confining surfaces. Volumes are *glued* together in the sense that bounding surfaces from neighbouring volumes attach completely, where possible bounding surfaces between volumes are shared. The bounding surfaces themselves are a templated extension of the surface classes, and they hold the possibility to point to inside or outside volume or volumes with respect to the surface's normal vector. This creates a predictive navigation through the tracking geometry, as every intersection at a given layer or volume boundary leads naturally to the next volume.

The reconstruction geometry can be built from various geometry models, such as the ATLAS GeoModel [10] description or the common DD4Hep modelling [11], see Section 6. To ensure a binding between the underlying detector modelling and the reconstruction geometry, a purely abstract `DetectorElementBase` class exists that needs to be implemented for each geometry conversion plugin. This detector element builds the link between the full detector geometry description and the simplified reconstruction geometry. It ensures that the sensitive detector elements are positioned correctly and that alignment effects are properly reflected. Figure 1 illustrates the relationship between detector elements in the detailed tracker geometry and the surface representation in the ACTS reconstruction geometry. Only sensitive detector elements are translated into corresponding ACTS surface objects. The remaining layer and volume structures are built automatically by the the ACTS geometry building tools. Finally, a mechanism to map the material from the detailed geometry to the simplified reconstruction geometry is in place.

3. Seed finding

The first step in track reconstruction from detector hits is the seed finding. From the set of all detector hits, suitable subsets that originate from the same particle track must be selected. These seeds yield an initial estimate of track parameters that can be used to propagate the track to other parts of the detector where additional hits will be searched for. Finding the initial seeds with high efficiency and purity is a difficult challenge especially for high pile-up conditions. Many

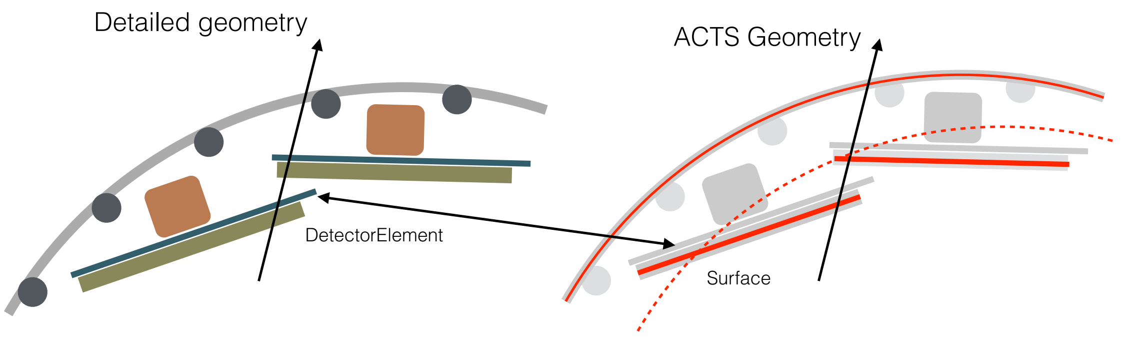

Figure 1. Illustration of the link of a detector element in a complex geometry model of a tracking detector to the surface representation in the ACTS reconstruction geometry. Only detector elements are directly translated, all other insensitive detector structures are simplified as layers and boundary surfaces.

possible combinations of hits must be considered and the combinatorial possibilities increase dramatically.

ACTS implements a simple combinatorial seed finder to select possible track seeds from a set of hits. For the typical number of tracks and measurements in a high energy physics collision, simply checking all possible combinations of hits is unfeasible. Additional constraints must be imposed to make the problem tangible. The ACTS seed finder assumes a homogeneous magnetic field, i.e. the particle trajectory can be described by a helix: a circle in the plane transverse to the beam direction and a linear movement along the beam direction. In addition, it takes advantage of the detector layout. Sensitive detector elements are assumed to be arranged in layers which can be ordered by the sequence a particle would traverse them. For each layer, the next layer along the nominal particle direction is well defined.

The input to the seed finder are three-dimensional `SpacePoints` instead of two-dimensional measurements. Depending on the detector type, a `SpacePoint` can be created directly from a single measurement, e.g. from silicon pixel sensors, or from a combination of multiple lower-dimensional measurements, e.g. from two silicon strip sensors. `SpacePoints` take an identifier as template argument which links back to the original measurement. The user can employ this to store a pointer to the relevant object in the event data model. The space points are then organised into layer containers. The resulting `TrackSeed` is templated on the number of hits and stores track parameter estimates and the list of input space points or identifiers.

The combinatorial seed finder proceeds as shown in Figure 2. Starting from hits on the first layer, additional hits on the second and third layer are searched within a window in the azimuth angle. The two hit doublets of these combinations are checked for consistency by comparing their polar angles. Both of these cuts are motivated by the assumed helical track model. The azimuth window determines the highest allowed curvature or lowest transverse momentum, while the polar angle should be consistent for true track candidates as a result of the linear longitudinal movement. For each such seed, initial track parameters can be estimated using purely geometrical circle and line fits. These can be used to constraint the compatibility with the known luminous interaction region.

Searching for possible hits in the given search windows must be optimised to allow fast seed finding. This usually takes advantage of the specific detector geometry, e.g. by using a binned data structure consistent with the detector layout to be able to select hits in a given window;

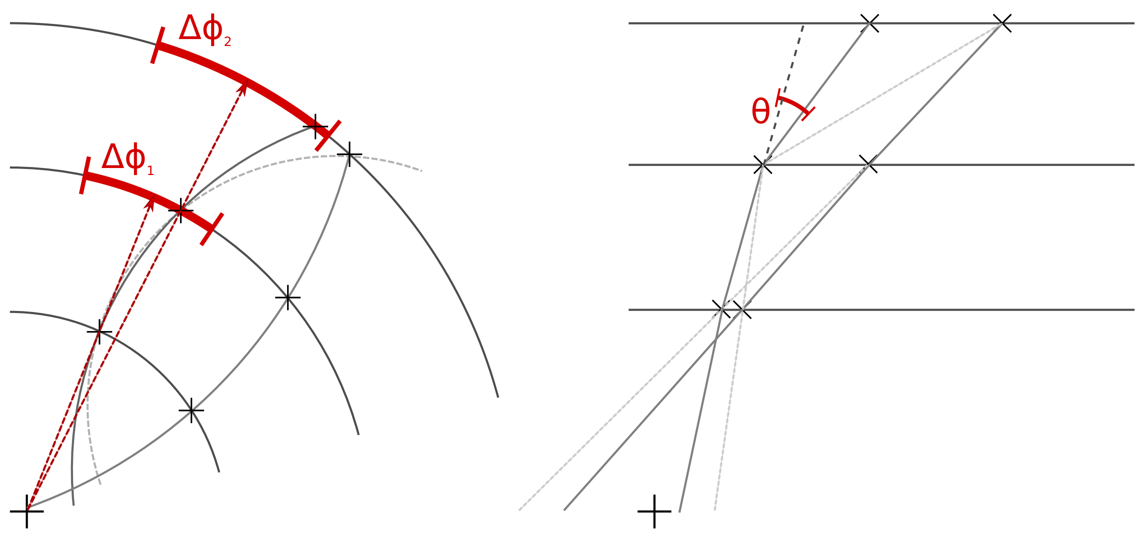

Figure 2. Illustration of the combinatorial seed finder for three-hit seeds in a barrel-like detector. Left: projection of the helix trajectory into the xy-plane transverse to the magnetic field direction. The two search windows $\Delta\phi_{\{1,2\}}$ on the second and third layer are defined with respect to the space point in the preceding layer. Right: hit combinations in the rz-plane along the magnetic field direction. The $\Delta\theta$ cut is defined as the difference in polar angle between the two hit doublets. Solid lines indicate true seeds, dashed lines indicate combinatorial background.

ACTS instead uses a detector-independent approach based on sorted vectors and assumptions on the layer shape. For barrel-like detectors a `BarrelSpacePoint` container stores the hits in a continuous vector sorted by azimuth angle. The elements within a certain window can then be determined using a binary search taking into account the periodicity of the angle. Work is ongoing to evaluate additional data structures and layouts.

4. Track parameter propagation

The trajectory of a charged particle in an arbitrary magnetic field can be described in the reference frame of a two-dimensional surface by five parameters: two local coordinates defining the current position of the particle, two angles giving the direction of the trajectory, and one parameter specifying its curvature. Calculating the trajectory from a given set of start parameters implies solving the equations of motion for the charged particle which are governed by the Lorentz force. Thus, the main task in track propagation is the integration of the equations of motion in conjunction with the transport of the track parameters' covariance matrix along the trajectory. In general, the choice of the free parameter to integrate the equations of motion depends on the geometry of the detector. As a consequence, an experiment-independent track reconstruction software package must not restrict the trajectory calculation to one particular parametrisation. Furthermore, the exact requirements on the behaviour of the `Propagator` class may vary from experiment to experiment. Typical examples are early abort conditions which stop the propagation in pathological situations. In order to meet the standard of a universal track reconstruction toolkit, the design of the `Propagator` class in ACTS

- is independent of the event data model for track parameters and surfaces on the user side,
- allows for the usage of custom defined abort conditions,
- supports execution of custom code after each integration step (observers),
- accepts custom implementations for solving the equations of motions.

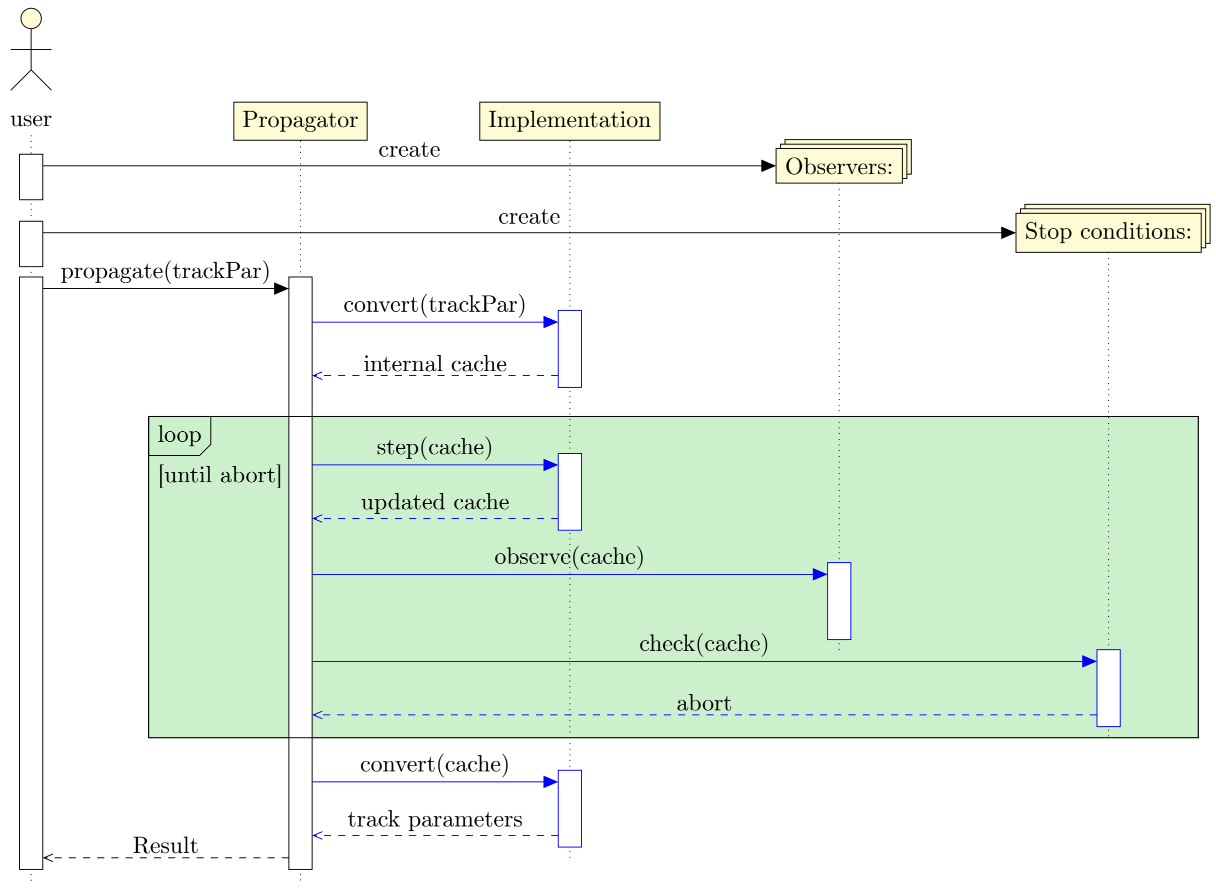

Figure 3. Sequence diagram for the propagation of track parameters. It is assumed that the `Propagator` and `Implementation` object are provided by the data processing framework used by the user. Otherwise, they can also be constructed by the user directly. Function calls in blue denote parts which can be replaced by custom implementations.

This flexibility is achieved without introducing the overhead of virtual function calls by making extensive use of variadic template features in C++. Upon construction the `Propagator` instance needs to be provided with an `Implementation` object which is responsible for a step-wise integration of the equations of motion. In addition, the `Implementation` class must expose a method for converting the event data model used for track parameters into its internal representation (*cache*) and vice-versa. The design of the `Propagator` class in ACTS allows users to provide specialised implementations for solving the equations of motion which are optimised for their use cases. A default implementation based on the concepts of the ATLAS offline track reconstruction software is provided and outlined below. Depending on the use case, users may decide to pass custom *observer* and/or *stop conditions* to the `Propagator::propagate` call. The interaction between the different components involved in track propagation is shown as sequence diagram in Figure 3 .

The default implementation for integrating the equations of motion uses the curvilinear parametrisation as internal representation of the track parameters and s, denoting the arc length along the trajectory, is taken as integration parameter. Let $\mathbf{r} = (x, y, z)^{\mathrm{T}}$ denote the position in the global coordinate system, $\mathbf{T} = \mathrm{d}\mathbf{r}/\mathrm{d}s$ be the normalised tangent vector of the trajectory, and $\lambda = q/p$ be the ratio of the particles charge over its momentum, then the equations of motion read

$$\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}s^2} = \lambda\left(\mathbf{T}\times\mathbf{B}(\mathbf{r})\right) \ . \tag{1}$$

This second order differential equation is transformed into a set of first order ordinary differential equations,

$$\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}s} = \mathbf{T},\ \frac{\mathrm{d}\mathbf{T}}{\mathrm{d}s} = \lambda\left(\mathbf{T}\times\mathbf{B}(\mathbf{r})\right),\ \text{and}\ \frac{\mathrm{d}\lambda}{\mathrm{d}s} = 0\ , \tag{2}$$

which are solved using the adaptive *Runge-Kutta-Nyström* method detailed in [12]. An implementation incorporating energy losses (e.g. due to material interactions) will be included in the future. The transport of the covariance matrix along the trajectory is performed using the semi-analytical *Bugge-Myrheim* method explained in [13].

5. Track fitting

Track fitting is the task of estimating track parameters from a set of measurements, so-called *hits*, which encode information about the position and/or direction of a charged particle passing through the detector. Measurements are typically defined in a two-dimensional reference frame which corresponds to the surface of the active detector element. Thus, hits describe the measurement of one or two local coordinates and possibly additional information about the incident angle (e.g. from the shape of pixel clusters) together with their covariance matrix. In general, particles pass through different sub-detector systems which leads to measurements of varying dimensionality. This requires any track fitting algorithm to abstract from the dimensionality of individual measurements while still maintaining the capability to benefit from optimisation of algebra operations on fixed-sized vectors and matrices. In ACTS this is achieved by combining variadic template trait classes for measurements of different dimensions with the visitor pattern implemented in the `boost::variant` library. Using a template-based approach has the further advantage of providing compile-time checks for the correct usage of `Measurement` objects. That is, querying a `Measurement` object, which contains information only about local coordinates, for the value (or uncertainty) of the incident angles will yield a compile-time error. This approach reduces the number of required safety checks during runtime and thereby minimises the number of branch conditions.

At the time of writing, the ACTS package contains a simple implementation of a Kalman fitter using the *gain matrix* method. It implements the mathematical algorithm of Kalman filtering while remaining independent of the concrete implementation of the track extrapolation. Similarly to the `Propagator` class described in Section 4, the `KalmanFitter` interface is designed with flexibility in mind. It

- is independent of the event data model used for particle tracks,
- supports custom implementations for the extrapolation of track parameters,
- allows for on-the-fly recalibration of measurements using updated information from the partial track fit,
- allows for augmentation of the track fit result with additional information (e.g. the number of crossed sensitive detector elements which had no measurement).

Further fitters including a *Gaussian Sum Filter* as well as a multi-track fitter are planned for the future.

6. Plugins

ACTS provides several plugins for interfacing its core functionality with external software packages. Plugins are optional and must be enabled explicitly during the `cmake` configuration step. This behaviour gives the user very good control over the provided functionality, but also the required dependencies. In the following the currently provided plugins are described.

Geometry plugins provide converters from the full detector description into the ACTS tracking geometry. Common to all geometry plugins is the implementation of a detector element and a layer builder which is part of the geometry building tools. Currently two geometry plugins are available: the TGeoPlugin and the DD4hepPlugin. The TGeoPlugin provides a translation of geometries built with the *ROOT Geometry Package* [14], which is a purely geometrical description. The DD4hepPlugin (Detector Description for High Energy Physics) provides a conversion from a geometry defined using *DD4hep* [11]. DD4hep is a package for specifying a full detector description, including geometry and detector specific information. It uses TGeo for the underlying geometry description and, therefore, the DD4hepPlugin depends on the TGeoPlugin.

The material mapping plugins provide the functionality to map the material of a detailed detector geometry onto the simplified tracking geometry. The user can provide material maps in any format and apply them using these plugins. During the mapping process the material distribution of the full detector description is projected onto the nearest layer in the simplified ACTS tracking geometry. The mapping is only done for layers which are marked as carrying support material. The marking of the layers can be done by the user, either directly during the geometry building, or by using the geometry plugins.

7. Summary

The ACTS project provides a framework- and experiment-independent toolkit for track reconstruction tasks. By adhering rigidly to *const-correctness*, ACTS algorithms are thread-safe and therefore well-suited for parallel event processing. Additionally, ACTS has a very short list of dependencies and lightweight code base which makes it easy to install and allows for fast compilation. This makes it a perfect environment for developing, testing, and prototyping new approaches to track reconstruction.

It includes a state-of-the-art implementation for the tracking geometry which features embedded navigation for a fast traversal through the detector geometry tree. A robust and configurable algorithm for finding track seeds in barrel-shaped detector geometries is provided. The versatile design of the track parameter propagation code ensures that this tool is widely applicable and highly customisable to meet the varying requirements by different users. The default implementation for integrating the equations of motion follows closely the implementation in the ATLAS offline track reconstruction software which is field-tested over many years yielding excellent performance and which has undergone several performance optimisation cycles. A track fitting class using a simple implementation of a Kalman filter is available as well. It supports features like in-situ re-calibration of measurements and on-the-fly hole counting which were proven to be useful in running LHC experiments. In order to facilitate the usage of ACTS in other frameworks, several plugins are provided to convert different geometry description formats or to interface with other external tools.

The ACTS project is actively developed and invites contribution from the community. Future developments will focus on the addition of more track fitting algorithms as well as specialised implementations for *packs* of tracks or track parameters to profit from the vectorisation capabilities of modern CPUs.

References

[1] Kalman R E 1960 *J. Basic Eng.* **82** 35 – 45
[2] Billoir P and Qian S 1990 *Nucl. Instr. Meth. Phys. Res. A* **294** 219 – 228

[3] Mankel R 1997 *Nucl. Instr. Meth. Phys. Res. A* **395** 169 – 184
[4] Frühwirth R 1987 *Nucl. Instr. Meth. Phys. Res. A* **262** 444 – 450
[5] The ATLAS Collaboration 2008 *J. Instrum.* **3** S08003
[6] The CMS Collaboration 2008 *J. Instrum.* **3** S08004
[7] Salzburger A 2015 *J. Phys.: Conf. Series* **664** 072042
[8] Cornelissen T, Elsing M, Gavrilenko I, Liebig W, Moyse E and Salzburger A 2008 *J. Phys.: Conf. Series* **119** 032014
[9] Salzburger A, Todorova S and Wolter M 2007 The ATLAS Tracking Geometry Description Tech. Rep. ATL-SOFT-PUB-2007-004 CERN Geneva
[10] Boudreau J and Tsulaia V 2005 *Computing in High Energy Physics and Nuclear Physics. Proc., Conf. CHEP'04, 2004 (Interlaken)* pp 353–356
[11] Frank M, Gaede F, Grefe C and Mato P 2014 *J. Phys.: Conf. Series* **513** 022010
[12] Lund E, Bugge L, Gavrilenko I and Strandlie A 2009 *J. Instrum.* **4** P04001
[13] Lund E, Bugge L, Gavrilenko I and Strandlie A 2009 *J. Instrum.* **4** P04016
[14] Brun R, Gheata A and Gheata M 2003 *Nucl. Instr. Meth. Phys. Res. A* **502** 676 – 680