"RecPack", a general reconstruction toolkit

A. Cervera-Villanueva[a], J.J. Gómez-Cadenas[b], J.A. Hernando[c]

***Abstract*— We present a general solution for the problem of reconstructing trajectories and vertexes. This solution has been realized in a C++ toolkit that could incorporate easily different methods for fitting, propagation, pattern recognition and simulation. The RecPack functionality is independent of the experimental setup, what allows to apply this toolkit to any dynamic system.**

***Index Terms*— reconstruction, kalman, track fitting.**

I. Introduction

In high energy physics (HEP), as in other fields, one frequently faces the problem of reconstructing (modeling) the evolution of a dynamic system from a set of experimental measurements. Most of reconstruction programs have common methods, which are however reimplemented for each specific case. For example, the Kalman Filter equations[1] , which are in general quite difficult to debug, are rewritten again and again. Other examples are the equations for propagation, random noise estimation, model conversion, etc. Similarly, the data structure (measurements, trajectories, vertexes, etc), which can be generalized as well, is not reused in most of the cases.

RecPack tries to avoid that by providing a general data structure and a set of common tools, which can be applied to any dynamic system. The package follows an "interface" strategy, that is, all the classes that could have a different implementation have their own interface, in such a way that the rest of the classes do not depend on such a specific implementation. This modular structure allows a great flexibility and generality.

This paper is structured as follows. The architectural design of the package is presented in section II. The geometry definition in section III. In section IV the model related methods are described. Sections V, VI, VII and VIII are devoted respectively to the fitting, navigation, matching and simulation capabilities of the package. Finally the RecPack versions and clients are presented in section IX.

II. Structure of the package

RecPack distinguish between data classes (passive) and machines (active). The tree of data classes is shown in Fig. 1. EVector and EMatrix are just a typedef of CLHEP's HepVector and HepMatrix respectively[2]. This establish the only RecPack external dependence. However, the user can replace the CLHEP classes by its favorite vector and matrix classes.

A structure that appears in several levels of the data model is the pair formed by a vector of parameters (EVector) and its covariance matrix (EMatrix). Thus, a new class call HyperVector has being introduced to hold this repeated structure. For example, experimental measurements (IMeasurement) can be always reduced to a HyperVector, and the same is true for the fitting parameters (IState).

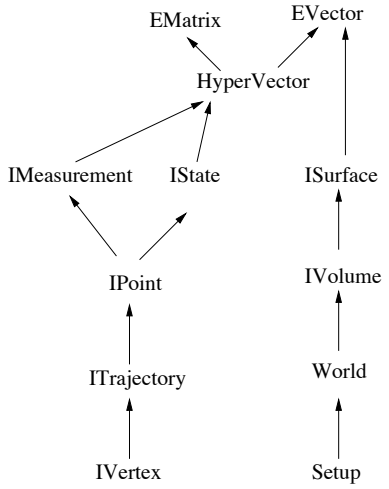

Fig. 1. Architectural design of the data classes.

A raw ITrajectory (before the fit) contains essentially a collection of IMeasurement's, while fitting results are stored in IState's. In the most general case the fitting parameters are local and therefore each IMeasurement must have an IState associated to it. Thus, an intermediate object, IPoint, has being introduced in order to accommodate the IMeasurement, the IState, and the quantities that relate both objects (the residual HyperVector and the local χ^2 of the fit.). Consequently, an ITrajectory can be seen as a collection of IPoint's, plus a set of global quantities as the total χ^2 of the fit, the number of degrees of freedom, etc. Finally, an IVertex is a collection of ITrajectory's.

The classes ISurface, IVolume, World and Setup, which are related with the definition of the experimental setup are treated in section III.

The data classes are not completely passive, they are allowed to perform internal operations. For example an IVolume has the function *bool is_inside(EVector&)*, which given a space point

a. Université the Genève, CH
b. Universitat de Valencia, Spain
c. CERN, Geneva, CH, and Universidade de Santiago de Compostela, Spain

[1]The Kalman Filter fitting method [1]-[2] is used in most of HEP experiments because its incremental strategy allows simultaneous pattern recognition and fitting, detection of outliers and an easy incorporation of random noise processes.

[2]These are vectors and matrices of double's with variable dimension.

returns true if the point is inside the volume and false otherwise.

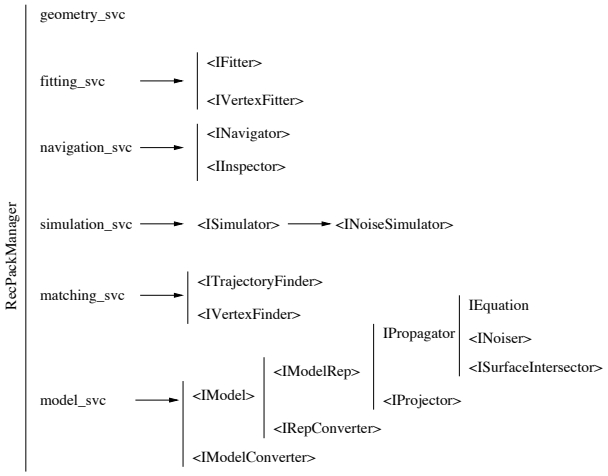

Fig. 2. Architectural design of the active classes (machines).

The active classes, also called machines (see Fig. 2), manipulate the data. All of them are pure interfaces (hence the I), which allow them to have different implementations. Machines are stored in services (_svc in Fig. 2), which not only actuate as containers for the machines, but also as managers. Indeed, the user interacts only with the RecPackManager, via its services, which provide user friendly methods. For example, fitting a trajectory by the least squares method to a straight line model would look like:

manager().fitting_svc().fit("Lsq", "straight_line", traj);

Some of the machines contain sub-machines (ISimulator, IModel, IModelRep, etc). If a machine has several sub-machines of the same type (i.e. IModel has several IModelRep), these are stored in associative containers (< ... > in the graph), which permits the access by key:

IModelRep& helix_ray =
manager().model_svc().model("helix").representation("ray");

In the following sections each of the services is treated individually.

III. Geometry

The RecPack geometry service provides the methods for the definition of the experimental setup (Setup), which is built through the addition of volumes (IVolume) and surfaces (ISurface). One can distinguish between basic surfaces, which have no borders, and finite surfaces, with a well defined size. On the other hand, volumes have by definition a concrete size. RecPack provides a set of predefined volume and surface types, but adding new types is straight forward. The available basic surfaces are plane, cylinder and sphere. Finite surfaces are derived from the basic ones:

- Plane: rectangle, ring.
- Cylinder: cylinder, cylinder sector.
- Sphere: sphere, sphere sector.

Three volume types are provided: box, tube[3] and sphere. Volumes may have properties, which can influence the evolution of the system (propagation). This is for example the case of a magnetic field, or the radiation length, for a charged particle.

Volumes and surfaces have a well defined position and axes inside the setup.

The definition of the experimental setup is quite straight forward from the user point of view. As an example, the following code defines a box called "tracker":

add_volume("tracker", "box", pos, axis1, axis2, size);

where pos, axis1, axis2 and size are EVector's. Similarly:

add_surface("wall", "rectangle", pos, axis1, axis2, size);
add_volume_to_volume("tracker", "my_tube", "tube", ...);
add_surface_to_volume("tracker", "my_ring", "ring", ...);

In this way, complicated setups as the one of Fig. 3 can be build.

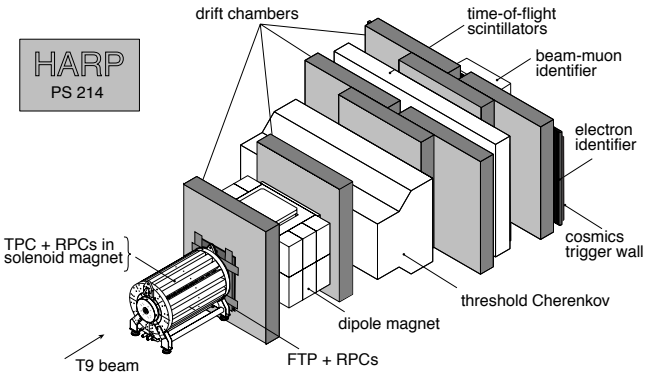

Fig. 3. Example of experimental setup corresponding to the HARP detector at CERN-PS [3].

An advanced feature of the geometry service is the possibility of having several worlds (see App. I). For this reason volumes and surfaces do not belong directly to the Setup. Instead, an intermediate object called World has being introduced (see Fig. 1).

IV. Model

The model service is the container and manager for model related equations: propagation, intersection with surfaces, projection, noise, model conversion, etc.

As shown in Fig. 2, an IModel contains a collection of model representations (IModelRep) and the corresponding converters between them (IRepConverter). A model representation

[3]A tube is defined by two concentric cylinders.

is determined by the definition of the state vector, $\vec{\mathbf{v}}$, that is, the set of independent parameters that describe the state of the system. For example, a 3D straight line may have several representations: $\vec{\mathbf{v}}_1 = (x, y, z, dx/dz, dy/dz)$, $\vec{\mathbf{v}}_2 = (r, \theta, \phi, dr/dz, d\theta/dz)$, etc. Each representation contains two major machines: IPropagator and IProjector.

A. IPropagator

The IPropagator propagates a state (which have a well defined position inside the Setup) to a given surface or final length. To do so, it delegates the intermediate calculations to smaller machines, which perform very concrete actions:

- <ISurfaceIntersector>: it is a collection of ISurfaceIntersector's, each of which corresponds to a different basic surface type (plane, cylinder, sphere). Its job is to calculate the path length from the actual position to the given surface.
- IEquation: it computes the position and direction of the state at a given length. This length can be provided by an ISurfaceIntersector, in the case of propagation to a surface, and externally, in the case of propagation to a length.
- <INoiser>: it is a collection of INoiser's. An INoiser computes the random noise covariance matrix for the given length and for a specific type of noise. Typical examples in HEP are multiple scattering and energy loss [4].

B. IProjector

An IProjector projects a state into a given measurement type according to the following equation:

$$\vec{\mathbf{m}}^{pred} = \vec{\mathbf{h}}(\vec{\mathbf{v}}), \quad \mathbf{C}_m^{pred} = \mathbf{H}\mathbf{C}_v\mathbf{H}^T, \tag{1}$$

where $\mathbf{H} = \partial\vec{\mathbf{h}}/\partial\vec{\mathbf{v}}$ is the projection matrix, $\vec{\mathbf{m}}^{pred}$ and $\vec{\mathbf{v}}$ are the predicted-measurement vector and state vector respectively, and $\mathbf{C}_m^{pred}$ and $\mathbf{C}_v$ their corresponding covariance matrices. Several measurement types could coexist with a single model representation. In a HEP particle detector, this is the case when several subdetectors that provide different type of measurements ("xy", "xyz"," $r\phi$", etc.), must be reconstructed with the same model representation (i.e. straight line with local parameters x, y, z, dx/dz, dy/dz). Therefore, each model representation must contain an extensible collection of IProjector's.

As we will see in forthcoming sections, this kind of projection is used by IFitter's (see Sec. V) and matching functions (see Sec. VII) to compute the residual HyperVector ($\vec{\mathbf{m}} - \vec{\mathbf{m}}^{pred}$, $\mathbf{C}_m + \mathbf{C}_m^{pred}$), and also by ISimulator's (see Sec. VIII) to create ideal measurements from states.

[4]Here we refer to energy loss fluctuations. Notice that energy loss it self introduces a systematic effect over the trajectory, which must be taken into account in the IEquation.

V. Fitting

Fitting algorithms are called fitters. There are two kind of fitters: trajectory fitters (IFitter) and vertex fitters (IVertexFitter). The fitting service has an extensible collection of IFitter's and IVertexFitter's. The user can either use one of the existing fitters or provide his own. Two IFitter's (least squares and kalman) and one IVertexFitter (kalman) are available.

An IFitter takes a raw trajectory (which contains measurements and empty states) and transforms it in a fitted trajectory, in which the states have meaningful contents. Similarly, the IVertexFitter takes a raw vertex (in this case the measurements are the fitted trajectories while the state associated to each trajectory is empty before the fit). In the case of a Kalman Filter fit (for trajectories or vertexes) a seed state must be provided.

The fitting functions provided by the fitting service have a user friendly interface. Let us consider the following example: we have a collection of 2D measurements produced by a charged particle in a magnetic field. These measurements have been already introduced in a raw ITrajectory (track) and now we want to fit it, first by least squares, and then use the result of this fit (track.first_state()) as a seed for a kalman filter fit. The necessary c++ code would be:

fitting_svc().fit("Lsq", "Helix", track);
fitting_svc().fit("Kalman", "Helix", track, track.first_state());

In HEP the Kalman Filter fit [1]-[2] is frequently used for track and vertex fitting. It has the advantage of being local (the fitting parameters variate along the trajectory) and incremental (measurements are treated sequentially), what allows simultaneous pattern recognition and fitting, and detection of outliers. In addition, it solves the problem of inverting large covariance matrices. Indeed, the largest covariance matrix to be inverted has the dimensions of the state vector (5 for a Helix model). However, in a least squares fit, the dimension is $N \times N$, being N the number of measurements. Finally, random noise processes, as multiple scattering, can be easily incorporated through a single covariance matrix.

VI. Navigation

The navigation service contains a collection of INavigator's. The user can use one of the existing INavigator's or provide a new one. The tasks of an INavigator are:

- Propagating to any surface within the setup.
- Propagating to a given length.
- Computing path lengths.

These tasks seem to be the same as the ones of an IPropagator (see Sec. IV). However, a clear distinction is made: the INavigator does the propagation in several steps while the IPropagator performs a unique step. In this sense, the INavigator needs one or several IPropagator's. Steps are needed when different volumes are traversed in a single propagation, when the volume properties are inhomogeneous, etc.

RecPack provides its own INavigator ("RecPackNavigator"). It distinguish between two types of propagation steps depending

on the properties of the actual volume: to a surface and to a length. If any of the volume properties is inhomogeneous the propagation occurs through dynamic stepping [5]. If all the properties are homogeneous there is no reason for small steps. In that case "RecPackNavigator" propagates to the next intersected surface.

After each step a list of IInspector's is called. IInspector's can be associated to volumes (called after step) or surfaces (called after intersection). An IInspector is a machine that produce a concrete action: set the properties of the entering volume, sum up intermediate path lengths, model conversion, set the length of the next step (dynamic stepping), etc. User defined IInspector's can be added to any surface or volume. For example, a CounterInspector could be added to a given surface in order to count the number of times this surface is traversed.

An important feature of the the "RecPackNavigator" is that the intersection with surfaces is done analitycally when it is possible (and numerically otherwise), what reduces considerably the computing time. The problem of intersecting volumes is allways reduced to the intersection with its outer walls.

VII. Matching

This generic name refers to the methods that are related with pattern recognition (PR) problems. In general, the purpose of PR algorithms is to distribute the existing measurements into trajectories, and these into vertexes. This is a common problem in HEP, but it can be generalized to any other field.

On can distinguish two types of PR algorithms: matching functions, which serve to estimate the probability of two objects of being related to each other (trajectory-trajectory, measurement-trajectory, trajectory-vertex, etc), and pattern recognition logics, which define the logical sequence in which such a relations are established. The first are always general, while the second may have a strong setup dependence. PR logics are introduced via two types of machines: ITrajectoryFinder and IVertexFinder, which build trajectories and vertexes respectively using the available matching functions and following a specific strategy.

Currently, the RecPack matching service provides the following functionality:

1) trajectory-measurement, trajectory-trajectory and trajectory-vertex matching functions.
2) Measurement, trajectory and vertex selection functions. For example, the function *best_matching_measurement(...)*, computes the best matching measurement in a given volume or surface, for the specified trajectory.

For the moment, PR logics are not implemented. In the future, one could try to identify common PR logics and include them in this service. For example, PR in a series of parallel planes which produce 2D measurements occurs always in a similar way. The same is true for a volume with 3D measurements (i.e. TPC), etc.

[5]Several steps of variable length according to the derivative of the inhomogeneous property.

VIII. Simulation

Some times reconstruction programs must operate over simulated measurements. However, in general one has to provide the classes and methods that allow the interface between simulation and reconstruction, which is not an easy task. The RecPack simulation service is able to generate simulated measurements with the data format required by the rest of the services, which can be very useful for obvious reasons.

As usual, this service has an extendible collection of ISimulator's. RecPack provides its own ISimulator ("RecPackSimulator"). The user must declare the active volumes and surfaces (the ones that produce measurements). and specify the measurement type in each of them. Active surfaces produce a measurement when they are intersected, while active volumes produced measurements through dynamic stepping (see Sec. VI)

The "RecPackSimulator" uses the propagation equations of the model service via the INavigator, which for a given simulation seed defines the ideal trajectory inside the setup. Then, a special IIspector ("CreateMeasurementInspector") creates ideal measurements in the active volumes or surfaces by calling the IProjector corresponding to the measurement type in that volume or surface. Ideal measurements are created according to equation 1. Finally, propagation noise and experimental errors are introduced by the INoiseGenerator's (multiple scattering, resolution, etc).

This simple simulator does not attempt to be a full simulator (i.e. Geant4). Instead, its main purpose is to serve as a debugging tool. For example, it is very useful to validate new IModel's, IFitter's, volume types, INoiser's, etc. It can be also used for analysis as a fast simulator.

Existing simulation toolkits, as Geant4, could be integrated into RecPack by implementing the ISimulator interface and the IInspector's that generate measurements in the different subdetectors. Such a inspectors should be able to access the Geant4 information and then use it to create specific measurements.

The measurements produced by an ISimulator are stored in a std::vector, which can be directly passed to the relevant RecPack services for pattern recognition, fitting, etc.

IX. RecPack versions and clients

RecPack was born in the HARP experiment at CERN-PS [3] (see Fig. 4). The initial version (RecPack-1) is being also used to study the MICE performance [4]. A reorganization of the code (RecPack0) has been done recently in order to gain in flexibility and generality. The new version is being used by the SciBar detector, which is part of the K2K experiment [5], for LHCb trigger studies [6] and for the design of a new experiment called HERO [7].

X. Conclusions

In summary, RecPack is a modular and extensible reconstruction toolkit, which provides the basic data structure and most of the common methods needed by any reconstruction program: matching, fitting and navigation. It also has the tools to perform a quick interface with simulation packages. The different RecPack services provide user friendly interface to control the the package functionality.

Appendix I
Advance geometry features

The elements of a setup can be distributed in several parallel worlds (World), which may have different dimensions. For example, a HEP particle detector needs a 3D world to study the trajectory of particles in our 3D space (Fig. 3). Is in this world where the various subdetectors are defined. However, one could define a bidimensional parallel world to study the energy loss by a charged particle as a function of the path length traveled. A non-HEP example of 2D world could be the evolution of the temperature in a given place as a function of the time (Fig. 4).

Worlds may also overlap each other. A typical case in which one could be interested in using this feature is when a magnetic field exceeds the borders of a volume. In this case, instead of defining the field as a property of the physical volume is preferable to define it in a overlapped world.

In the case of several worlds, each world has a mother volume, which defines the world's borders. A N-dimensional world will contain volumes and surfaces of dimension N and N-1 respectively. In the temperature example, a volume could be a well defined period and temperature ranges, while a surface could be a concrete day (1D).

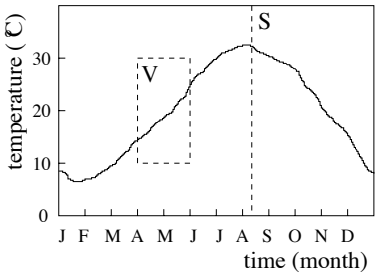

Fig. 4. Example of 2D world. V is a 2D volume while S is a 1D surface.

Acknowledgment

We would like to thank Gersende Prior for her help with the Kalman Filter vertex fit. The contribution of Malcolm Ellis, as the main RecPack user, has being essential for reporting a non negligible amount of bugs.

References

[1] R.E. Kalman, J. Basic Eng. 82 (1960) 35
R.E. Kalman, R.S. Bucy, J. Basic Eng. 83 (1961) 95
[2] R. Fruhwirth, M. Regler, Nuc. Ins. and Meth. A 241 (1985) 115.
[3] The HARP Collaboration. *http* : *//harp.web.cern.ch/harp/*
[4] The MICE Collaboration. *http* : *//hep*04*.phys.iit.edu/cooldemo/*
[5] S.H. Ahn. et *all*. The K2K Collaboration. Phys. Lett. B511 (2001) 178-184
[6] The LHCb Collaboration. *http* : *//lhcb.web.cern.ch/lhcb/*
[7] No references available yet.