BINGO: A Set of Object Oriented Tools for Hierarchical Pattern Recognition and Track Fitting

D. Chin[1], *A. Magerkurth*[2], *M. Marsh*[3], *M. Palmer*[3], *J. Thaler*[3]

[1] Department of Physics, Applied Physics and Astronomy, Binghamton University, USA
[2] Department of Physics, Cornell University, USA
[3] Department of Physics, University of Illinois at Urbana-Champaign, USA

Abstract

We describe an object oriented set of tools for carrying out integrated hierarchical pattern recognition in a tracking system composed of an arbitrary set of tracking devices. Two fundamental components are part of the method, a LayerSet object for pattern recognition operations and a TrackFilter object for manipulation of track candidates. The other major components in the package are: a management object that controls the package's overall operation; a generic input data object capable of holding information from any type of tracking element; and a track candidate object that can receive output from a LayerSet or TrackFilter algorithm. We have implemented a dictionary-based pattern recognition algorithm in this framework.

Keywords: Pattern Recognition, Tracking

1 Introduction

Several key steps define the exercise of pattern recognition in any real detector. First of all, hits in the detector must be converted to spatial information. Secondly, the hits must be processed by one or more algorithms in order to group individual hits into lists of hits which potentially belong together as track candidates. Thirdly, the track candidates typically need some level of preliminary fitting both to exclude bad possibilities and to provide starting track parameters before being sent off for final fitting. Lastly, the necessary bookkeeping must be carried out to keep the track candidates, unfitted and fitted, properly organized.

In developing the BINGO inner tracker for CLEO III, we closely examined the above issues with the goal of standardizing the various operations wherever possible and isolating any details of a specific detector to the user interface. This approach places the user in a position to concentrate primarily on implementing pattern recognition algorithms and studying their performance instead of dealing with detailed code infrastructure. The current CLEO III implementation uses this package as an inner tracker to improve our low momentum pattern recognition capabilities for a silicon microstrip device in combination with the inner drift chamber layers and the drift chamber cathodes [1]. For operation in the CLEO III software environment, we are able to directly interface to the package from C++. Since the CLEO II software environment is Fortran based, we have also provided an interface layer suitable for access from procedural languages. Thus the package is readily adaptable to the software environments of the majority of high energy physics experiments.

2 Package Overview

Five core objects make up our pattern recognition package:

- `Scungili` This is the *master* class which manages the operation of one or more track finders (implemented with the `LayerSet` class) and one or more track filters (implemented with the `TrackFilter` class) . It also manages the lists of track candidates and any scratch memory requested by a `LayerSet` or `TrackFilter`.
- `TrackHit` This class provides a standard interface to the data.
- `TrackCand` This class provides a standard internal representation of the list of hits belonging to a track and the track parameters and error matrix for the track if they exist. Both the `LayerSet` and `TrackFilter` classes can internally generate `TrackCands`. In addition, the user can also load `TrackCands` from an external source (*ie.,* from another tracking package) for comparison and merging with internally generated candidates if so desired.
- `LayerSet` This class carries out pattern recognition operations. The fundamental components of the `LayerSet` are a set of tracking layers on which to operate, a pattern recognition algorithm with which to examine the specified tracking layers, and a user-assigned ID. Each implemented `LayerSet` takes a list of `TrackHit` objects as input and generates a list of `TrackCand` objects as output.
- `TrackFilter` This class is explicitly designed to manipulate `TrackCand` objects as inputs and to generate updated `TrackCand` objects as outputs. Thus it can filter and fit preliminary pattern recognition candidates as well as merge multiple candidates into a more complete track.

These objects provide all of the infrastructure needed to implement a pattern recognition algorithm for a detector.

3 Managing Pattern Recognition and Track Candidate Creation with `Scungili`

`Scungili` is a utility for coordinating the simultaneous operation of several track finders and track filters. Implementing the package in a C++ environment is quite straightforward. The user must first create an instance of `Scungili` as well as instances of one or more `LayerSets` and `TrackFilters`. During job processing, the user's interface must insure that the relevant `Scungili` methods are activated at the correct time. The standard job flow entry points include: `beginJob()`, `beginRun()`, `event()`, `endJob()`, and `endRun()`. In addition, two other methods are defined which can be called at arbitrary points in the job: `status()` and `reset()`. Once any of the `Scungili` methods has been called, the corresponding methods for any enabled `LayerSet` or `TrackFilter` will automatically be called by Scungili with no further action from the user. The order of operation of `LayerSet` and `TrackFilter` methods is simply determined by the order in which they are enabled within Scungili, with all `LayerSet` operations preceding any `TrackFilter` operations.

The interactions between `Scungili` and the other classes in the package are shown in Figure 1. In order to illustrate these interactions, we can consider the steps that take place at the `event()` level:

1. The **user** prepares a vector of `TrackHit` objects to pass to `Scungili`.
2. The **user** calls `Scungili::event()`.
3. **Scungili** calls each `LayerSet::event()` method and provides each `LayerSet` with the relevant list of `TrackHit` objects. **Scungili** stores any found `TrackCand` objects returned

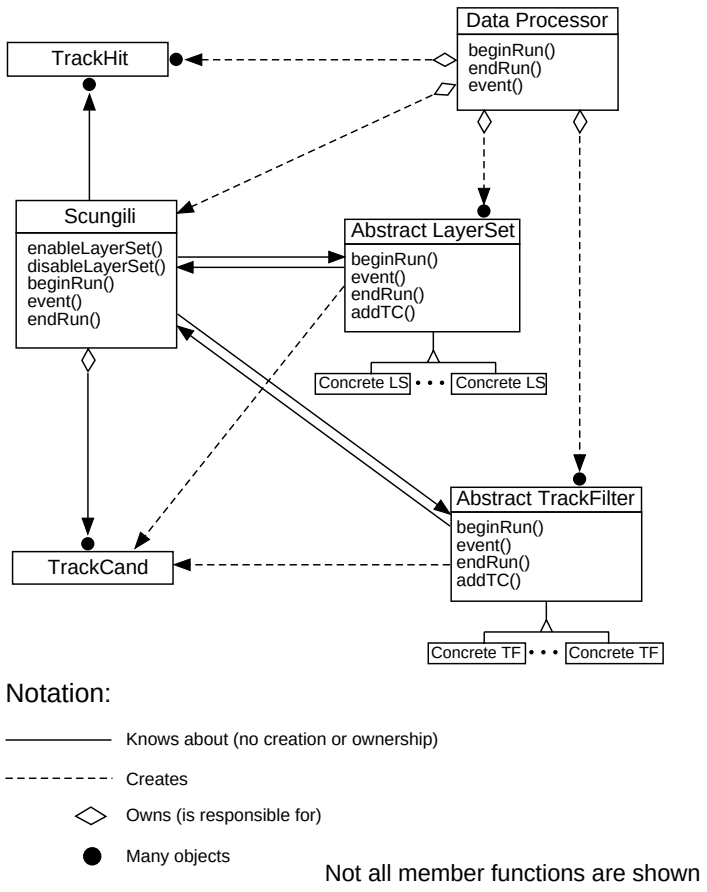

Figure 1: The class relationships within Scungili.

by a `LayerSet`.

4. **Scungili** calls each `TrackFilter::event()` method and provides access to the list of found `TrackCand` objects. **Scungili** stores any fitted `TrackCand` objects returned by a `TrackFilter`.
5. The **user** is now able to obtain a final list of fitted track seeds from `Scungili` for further processing.

Thus the code is structured to insure that the pattern recognition steps occur in the proper order. Also, the bookkeeping necessary to keep track of the pattern recognition results is automatically handled so that the **user** is completely freed from these details.

In order to accommodate access from procedural languages a C++ singleton can be used to create the necessary class structure for the package. A set of singleton functions which execute the Scungili methods can be directly called from Fortran or C. We employ this method when using this package to run in the CLEO II Fortran framework.

4 Pattern Recognition with the `LayerSet` Class

We have implemented the MARK III dictionary-based pattern recognition algorithm [2] as the primary type of `LayerSet` (the `BinGoLayerSet`) for the BINGO package. In this algorithm, each tracking layer of interest is divided into a group of bins. Up to 8 tracking layers at a time are examined for the presence of a pattern of hits corresponding to a physical track. Particular advantages of this algorithm for our purposes are: the binning technique readily accommodates any sort of detector geometry; the integer pattern recognition does not require fine-tuning cuts; and, the method is insensitive to detailed detector constants and performance issues.

Our implementation of the `BinGoLayerSet` class cleanly isolates the core algorithm from the details of any specific detector. All of the detector-specific information is distilled into a user-supplied binning function which provides a map from the ID of a detector element associated with a hit to the corresponding bin in its tracking layer. Standard methods are provided to generate track dictionaries using a Monte Carlo simulation of a given detector. Thus it is quite straightforward to implement this algorithm for arbitrary experimental configurations.

5 Track Candidate Creation, Filtering, and Manipulation with the `TrackFilter` Class

A range of functionalities can be imagined for the `TrackFilter` class. The first version that we have implemented applies simple, fast line and circle fit operations to `TrackCand` objects provided by the `LayerSet` class. These preliminary fits serve to filter noise hits from the pattern recognition candidates and resolve drift chamber hit ambiguities. The resulting fitted track candidates carry along track parameter and error matrix information from the fits which can be used in subsequent processing.

A second `TrackFilter` operation that we have implemented provides chi-square matching between `TrackCands` from different `LayerSets`. This allows us to join `BinGoLayerSet` pattern recognition candidates from different regions in the detector into more complete tracks that can then be fitted and placed in the general repository of seed tracks in the CLEO detector. We are also exploring the use of this algorithm to merge track candidate lists from our package with the candidates produced by the road-based algorithm which is employed for higher momentum tracking at CLEO.

6 Summary and Conclusions

In summary, we have designed a set of object oriented tools that allow us to coordinate pattern recognition within a group of user-specified `LayerSets`, to verify that the resulting track segments are reasonable by means of a preliminary fit, and to link track segments from different regions of a detector into more complete tracks which are then returned to the user for further processing. Care has been taken in both the management infrastructure and in the design of the primary pattern recognition package to isolate detector-specific information to a well-defined set of interface routines. In addition, the pattern recognition algorithm that has been chosen can readily handle a variety of detector geometries. Thus the package is easily adapted for use in other experimental configurations. It is also straightforward to introduce additional algorithms.

References

1 CLEO Collaboration, CLNS 94/1277, 1994.

2 J. Becker, *etal.*, *Nucl. Instrum. Methods* , **A235**,502(1985).