MICE Global Track Reconstruction Specification

MAUS Work Specification Document

Change History

Version	Changes	Author	Date
1	First draft	C. Rogers	06/01/2012
2	LaTeX draft with class hierarchy	P. Lane	14/02/2012
3	Section reorganization. Polynomial map and task details.	P. Lane	16/02/2012
4	Dot-derived class hierarchy diagram.	P. Lane	21/02/2012

Project Summary

Project Title	Global Reconstruction
Main Issue	Create a global track reconstruction tool.
Subtask Issues	
Project Lead	P. Lane
Project Supervisor(s)	A. Blondel C. Rogers
Associated Manpower	

Contents

MAUS Work Specification Document ii
Change History . . . ii
Project Summary . . . ii

Contents iii

I Overview of Work 1

1 Motivation and Overview 2
1.1 Project Goal . . . 2
1.2 Use Cases . . . 2

2 Specification 4

3 Implementation 6
3.1 Optics Model . . . 6
3.1.1 Polynomial Transfer Map . . . 6
3.1.2 Runge-Kutta Transfer Map . . . 6
3.1.3 Calculation of the Transfer Matrix . . . 7
3.2 Reconstruction . . . 8
3.2.1 Track and Error Propagation . . . 8
3.2.2 Score Function Minimisation . . . 9
3.2.3 Seed Finding . . . 10
3.3 Particle Identification . . . 10
3.3.1 Candidate Particles . . . 10
3.3.2 Mass as a Free Parameter . . . 10
3.3.3 Any other PID routines? . . . 10
3.4 Class Hierarchy . . . 10
3.4.1 Input Data Structure . . . 12
3.4.2 Output Data Structure . . . 12
3.5 Task Breakdown . . . 13
3.5.1 Data Structures . . . 13
3.5.2 Interfaces . . . 13
3.5.3 Polynomial Transfer Map . . . 13
3.5.4 Basic Minuit Track Fitter . . . 13
3.5.5 Basic Track Reconstructor . . . 13
3.5.6 Minuit Material Effects . . . 13
3.5.7 Differentiating Optics Model . . . 13
3.5.8 Integrating Optics Model . . . 13
3.5.9 Runge-Kutta Transfer Map . . . 14
3.5.10 Runge-Kutta Material Effects . . . 14
3.5.11 Kalman Filter . . . 14

3.5.12 TOF Track Fitter 14
3.6 Existing Code and Location of New Code 14
3.7 Potentially Conflicting Work 14

II MAUS Overview and Requirements 15
3.8 Issue Tracking 16
3.9 Code Version Control 16
3.10 Communication 16
3.11 Code Quality 16
3.11.1 Coding Style and Comments 17
3.11.2 Testing 17
3.11.3 Documentation 17
3.11.4 Code Review 17

III Effort and Timescale 18
3.12 Effort Available 19
3.13 Major Milestones 19

Part I

Overview of Work

1 Motivation and Overview

1.1 Project Goal

The MICE collaboration, seeks to:

- measure phase space variables position, time, momentum and energy at different planes along the beam axis of the experiment;
- identify the particle species of particles traversing the experiment.

One seeks to compare the two sets of variables to perform certain physics analyses such as calculating beam properties like emittance. In addition, one seeks to propagate these variables through the experiment, for example to perform detector alignment in the presence of fields or to measure the effect of physics processes in the cooling channel (e.g. multiple scattering at absorbers).

The position and momentum is measured at the trackers, while time is measured at fast time of flight detectors upstream and downstream of the experiment. Additionally some detectors have been placed into the accelerator specifically to determine particle species, namely the KL, EMR and Cerenkov detectors. The reconstruction of individual detectors is the task of the detector working groups, under joint supervision of the MAUS project manager and detector working group managers.

The global reconstruction task is to connect together the individual measurements of each detector into a global track. Typically, one would seek to make global tracks that conform as closely as possible to the measurements made by each individual detector.

1.2 Use Cases

Here I include a few sample analyses (not exhaustive) that one might want to make of a global reconstruction algorithm:

- Generate phase space vectors at the input and output of MICE. Reject all particles that are not muons or do not traverse the experiment. Calculate the change in 4D emittance.
- Generate phase space vectors at the input and output of MICE. Reject all particles that are not muons or do not traverse the experiment. Calculate the change in longitudinal (time-energy) emittance.
- Calculate time and energy at the input and output of MICE. Look at the energy change vs time relationship for individual particles and verify that the energy change in RF cavities is correct.
- Generate phase space vectors at the input of MICE using only upstream detectors. Apply some cut to the data to reject particles that do not conform to an ideal beam distribution. Generate phase space vectors at the output of MICE and calculate the emittance change.

- Calculate a phase space vector at the input to MICE using only the upstream tracker. Propagate it to the output of MICE. Calculate a phase space vector at the output to MICE using only the downstream tracker. Compare the residuals to understand whether there is a bias in the reconstruction of events (caused by e.g. detector or magnet misalignment).
- Calculate phase space vectors at the input to MICE. Look at those that are not transmitted to the output of MICE. Seek to understand the aperture of the cooling channel.
- Calculate beam purity in the MICE beamline. Seek to optimise magnet settings for a pure muon beam.

And so on...

2 Specification

The global reconstruction shall generate a track consisting of a set of phase space vectors with the following information.

- x
- y
- z
- t
- px
- py
- pz
- energy
- Most favoured particle type (pid)

Additionally the global reconstruction shall return errors for the phase space vector

- The error matrix on the phase space variables u=(x,y,t,px,py,energy) that is the matrix of errors and correlations where each element of the matrix vij are of the form Cov(ui, uj) where Cov() is the covariance for the (multidimensional) error probability distribution function.
- Probability that the particle is an electron, muon, pion or something else *is that right? Or should we have errors on mass? Or something else?*

The global reconstruction routines shall work with any set of detectors, for example it should be possible to do only upstream, only downstream and combined analyses; and if a detector is removed for maintenance we should be able to work without.

The global reconstruction shall reconstruct tracks at user-defined positions along the beamline. The global reconstruction shall not have a preferred frame of reference.

- we will probably end up working in engineering coordinates with x-axis being the beam axis
- we might want to include dipoles in the track fitting routine

The global reconstruction shall reconstruct tracks passing through

- Material
- Solenoidal fields
- Quadrupole fields

- RF fields
- Dipoles

3 Implementation

3.1 Optics Model

The optics model is embodied in the transfer map. Two different models are foreseen: polynomial expansion and Runge-Kutta. Neither of these methods accounts for scattering in materials. An additional procedure must be used to deal with these non-linear momentum changes.

3.1.1 Polynomial Transfer Map

The polynomial transfer map is an expansion of each coordinate in the final phase space vector as a multivariate polynomial in terms of the initial phase space vector coordinates. The transfer map then comes in the form of a matrix of polynomial coefficients P_{ij}. To transform a phase space vector, it is first expanded into a vector of polynomial variable terms

$$\left(t, E, x, p_x, y, p_y\right) \rightarrow \left(1, t, E, x, p_x, y, p_y, t^2, tE, tx, tp_x, \ldots\right)$$

and then operated on by the coefficient (transfer) matrix:

$$\begin{pmatrix} t_f \\ E_f \\ x_f \\ p_{x_f} \\ y_f \\ p_{y_f} \end{pmatrix} = \begin{pmatrix} P_{11} & \cdots & \cdots & \cdots & \cdots & P_{1N} \\ P_{21} & \ddots & & & & \vdots \\ P_{31} & & \ddots & & & \vdots \\ P_{41} & & & \ddots & & \vdots \\ P_{51} & & & & \ddots & \vdots \\ P_{61} & \cdots & \cdots & \cdots & \cdots & P_{6N} \end{pmatrix} \begin{pmatrix} 1 \\ t_i \\ E_i \\ \vdots \\ t_i p_{x_i} \\ \vdots \end{pmatrix}.$$

The beam envelopes can be transported by a similarity transform by a matrix formed from the first-order elements of the transfer matrix:

$$\begin{pmatrix} < tt > & \cdots & < tp_y > \\ \vdots & \ddots & \vdots \\ < p_y t > & \cdots & < p_y p_y > \end{pmatrix} = \begin{pmatrix} P_{12} & \cdots & P_{17} \\ \vdots & \ddots & \vdots \\ P_{62} & \cdots & P_{67} \end{pmatrix} \begin{pmatrix} < tt > & \cdots & < tp_y > \\ \vdots & \ddots & \vdots \\ < p_y t > & \cdots & < p_y p_y > \end{pmatrix} \begin{pmatrix} P_{12} & \cdots & P_{17} \\ \vdots & \ddots & \vdots \\ P_{62} & \cdots & P_{67} \end{pmatrix}^T$$

3.1.2 Runge-Kutta Transfer Map

The Runge-Kutta method calculates incremental vector deltas over uniform intervals along the beam axis. The deltas are then integrated to produce the final vector from the initial vector. Details of the algorithm can be found online or in the literature.

Additional care will need to be taken when transporting particles backwards through the lattice as Runge-Kutta incorrectly handles energy gain in materials. Considering that multiple scattering is also not handle automatically, it is apparent that only free-space transport will be handled by the Runge-Kutta algorithm. Transport through materials will be handled separately.

The beam envelope is transported by calculating a transfer matrix between two incremental steps of the vector and performing a similarity transformation as was described in *Polynomial Transfer Map*.

3.1.3 Calculation of the Transfer Matrix

Numerical Differentiation of Tracking

The transfer matrix can be calculated by numerically differentiating tracking output. Say we have a 2D transfer matrix in (x, p_x). We propagate tracks by integration of the usual equations of motion. Let us propagate the following three tracks

$$u_a = (x, p_x)$$
$$u_b = (x + dx, p_x)$$
$$u_c = (x, p_x + dp)$$

Here u_a is just the central track. Then using the defining relationship for a transfer matrix,

$$u_2 = M_{12} u_1$$

we can write

$$u_{b,2} = M_{12} u_{b,1}$$
$$u_{c,2} = M_{12} u_{c,1}$$

which is a simultaneous equation with unknowns as the terms of M, that can be solved using common simultaneous equation techniques. The algorithm generalises to an arbitrary dimension phase space.

Linear Least Squares (first order correction)

The second order correction can be made by propagating more tracks. Let's now propagate

$$u_b = (x + dx, p_x)$$
$$u_c = (x, p_x + dp)$$
$$u_d = (x - dx, p_x)$$
$$u_e = (x, p_x - dp)$$

Then instead of searching for the transfer matrix using a first order differentiation as above, we can find the tracks using a linear least squares fit. This should eliminate errors due to second order derivatives from the transfer matrix calculation, at a cost of doubling the number of tracks that must be transported.

Direct Integration

It is additionally possible to derive differential equations for the transfer matrix elements directly, for example, by taking the usual equations of motion and integrating them using standard techniques. This may be faster to run and will require fewer control parameters (a distance and a time).

3.2 Reconstruction

The fundamental algorithm that is proposed for track reconstruction is to take information yielded by each detector, together with estimated errors, and find the track that minimises the sum of the residuals (difference between the model prediction and the measurement from the detector) of each measurement, normalised to the errors on the measurement. In general, two algorithms are foreseen:

- An algorithm that propagates the tracks and associated error matrix between detectors
- A minimisation routine that minimises the errors given the tracks

Several different algorithms can be foreseen; one task of the Global Reconstruction is to determine a framework in which different algorithms can be implemented without major infrastructure changes.

3.2.1 Track and Error Propagation

In the paraxial approximation, tracks propagate according to an equation like

$$u_2 = M_{12}u_1$$

where u_1, is the phase space vector at some z position z_1, u_2 is the phase space vectors at some z position z_2 and M_{12} is the transfer matrix that propagates particles from z_1 to z_2 The error matrix propagates like

$$V_2 = M_{12}V_1M_{12}^T$$

where T superscript denotes the transpose matrix, V_1 denotes the error matrix at z_1, V_1 denotes the error matrix at z_1. (These relationships can be derived, see for example Chris Rogers thesis chapter 3, available on MICE website). We seek to minimise the sum of the χ^2 over the i measurements, defined by

$$\sum \chi_i^2 = \sum u_i \mathrm{E}_{i-1} u_i^T$$

where the i subscript indicates that the values are calculated at the position of the ith detector. E_i is the error matrix associated with a particular detector. Possibly can linearise this problem completely (i.e. express as linear sum of some seed value) to make linear least squares possible.

Propagation through Material

The presence of material in the beamline introduces energy loss to the track and creates noise that increases errors.

Tracks propagate through material losing energy according to the Bethe Bloch relationship. Error matrices propagate through material in the thin material approximation as pure scattering and energy straggling.

From the Particle Data Group's July 2010 *Particle Physics Booklet*...

For a gaussian distribution of small scattering angles, the width of the gaussian is

$$\theta_0 = \frac{13.6\text{MeV}}{\beta_c p} z\sqrt{x/X_0}\left[1 + 0.038\ln(x/X_0)\right],$$

where p, β_c, and z are the momentum, velocity, and charge number of the incident particle; and x/X_0 is the thickness of the scattering medium in radiation lengths.

The mean energy deposit by an ionizing particle when energy transfers are restricted to $T \leq T_{\text{cut}} \leq T_{\max}$ is

$$-\left.\frac{dE}{dx}\right|_{T<T_{\text{cut}}} = K_z^2 \frac{Z}{A}\frac{1}{\beta^2}\left[\frac{1}{2}\ln\frac{2m_e c^2\beta^2\gamma^2 T_{\text{cut}}}{I^2} - \frac{\beta^2}{2}\left(1 + \frac{T_{\text{cut}}}{T_{\max}}\right) - \frac{\delta}{2}\right]$$

3.2.2 Score Function Minimisation

As discussed above, the reconstruction routine requires minimisation of the sum of the chi-squared values, $\sum \chi_i^2$. Various minimisation routines exist. It is foreseen that two routines will be investigated initially: Minuit and Kalman Filter.

Minuit Minimisation

Minuit is the algorithm, packaged with ROOT, that allows for solving of fairly general minimisation problems. It has a fairly tractable interface (see TMinuit documentation in ROOT) although it is probably not optimal for our problem. However, the majority of the optimisation has been done already as it is built-in to ROOT.

Kalman Filter Minimisation

The Kalman filter technique may be a more optimal solution. This algorithm has been proposed for use by the tracker group.

The Kalman filter would work in the context of track reconstruction by predicting subsequent particle event coordinates and calculating a weighted average between the prediction and the actual measurement from the detector. The transfer matrix is given, and the last weighted average is propagated back through the lattice to produce a final trajectory.*Is this last part right?*

TOF Iteration (***Better Name?***)

The TOF iteration takes advantage of the fact that the TOFs reconstruct longitudinal phase space quite accurately, and that longitudinal phase space is approximately independent of longitudinal phase space (cf Mark Rayner thesis). The TOF iteration

1. reconstructs momentum from measured time of flight

2. generates a transfer matrix for this momentum

3. uses the generated transfer matrix to calculate transverse phase space parameters
4. recalculates momentum using transverse phase space parameters

Other Minimisation Techniques

There is a rich literature in minimisation algorithms dating back to the era of Newton. Neural networks, simulated annealing, etc.

3.2.3 Seed Finding

All the minimisation routines discussed above are iterative routines. In most cases, the tracker reconstruction will deliver a seed for the global reconstruction. An additional time parameter will be required. This can be calculated analytically assuming no energy loss between the tracker and the nearest time of flight detector. If the tracker is not in use or reconstruction of tracks with no tracker hit is required, it may be sufficient to assume a paraxial particle.

3.3 Particle Identification

Particle identification can be treated in either of two ways. The simplest method is to specify a list of candidate particles and pick out the best fit given certain parameters. One may also be able to use mass as a free parameter in the track fitting.

3.3.1 Candidate Particles

This method performs track fitting with three candidate particle types: electron, muon, and pion. The most favoured particle type is that which fits the input variables in the best approximation.

A second round of fitting adding results such as light yield in the Cerenkovs may then improve the pid.

3.3.2 Mass as a Free Parameter

It may also be possible to use mass as a free parameter in the track finding. The algorithms described above are sufficiently general that they can support track finding with any mass. Charge will be assumed to be +1 or -1 according to the beamline polarity.

3.3.3 Any other PID routines?

3.4 Class Hierarchy

Figure 1.1 shows the proposed, main class hierarchy for track reconstruction. The interfaces OpticsModel, TransferMap, and TrajectoryFitter provide the required implementation flexibility which is detailed in figure 1.2. The particular optics model and trajectory fitting algorithm can be specified in the configuration file. The appropriate class instances will then be instantiated.

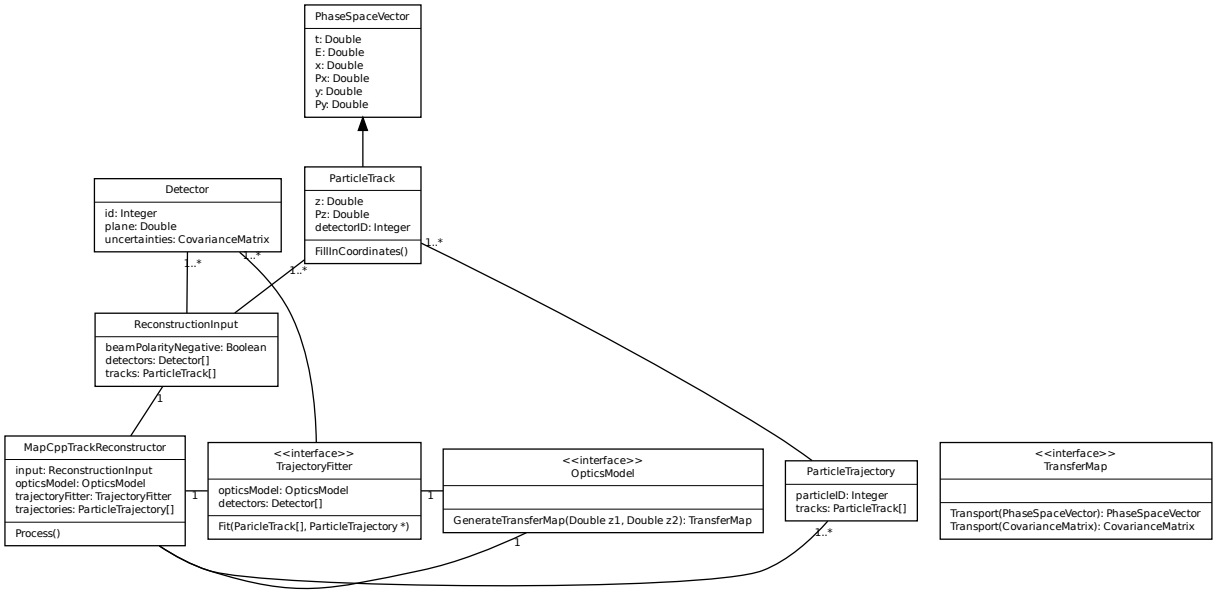

Figure 3.1: Main track reconstruction class hierarchy

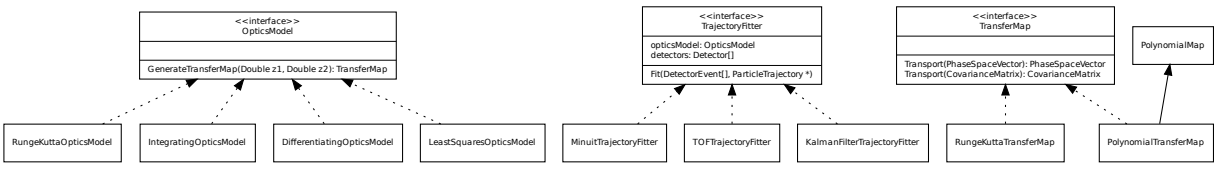

Figure 3.2: Track reconstruction interface details

3.4.1 Input Data Structure

The inputs to reconstruction are embodied in the ReconstructionInput class. This will likely be constructed by the reconstruction map component from seperate data structures supplied by the various detectors. The following abstract description of this class is a guide for the detector groups in order that they may include all of the required information in their output data structures.

An instance of the reconstruction input data structure requires three pieces of information:

1. a flag indicating the polarity of the particle beam,
2. a list of detector information data structures, and
3. a list of particle track data structures.

Each instance of the detector information data structure is comprised of

1. a unique detector ID in the form of an *unsigned integer greater than zero*,
2. a distance along the beam axis where the center of the detector lies, and
3. a matrix of covariances indicating the uncertainties of the detector.

The items of the particle track data structure that are used for inputs into reconstruction are

1. the 6D phase space coordinates of the track (t, E, x, Px, y, Py), and
2. the ID of the detector which measured the track.

This data structure also contains the z and Pz coordinates as well an uncertainty matrix. The z and Pz coordinates are filled in automatically and are provided as an output of reconstruction. For measured particle tracks the uncertainty matrix is filled in using the one given for the detector where the track was measured.

3.4.2 Output Data Structure

The outputs of reconstruction are a list of particle trajectory data structures. Each trajectory contains

1. the PDG particle ID for the type of particle associated with the trajectory, and
2. a list of particle track data structures that comprise the discrete trajectory.

Each particle track data structure contains the following items:

1. the 6D phase space coordinates of the track (t, E, x, Px, y, Py),
2. the calculated z and Pz coordinates,
3. the uncertainties on the phase space coordinates in the form of a covariance matrix.
4. the ID of the detector which measured the track, or zero if this track was calculated.

3.5 Task Breakdown

3.5.1 Data Structures

Define input and output data structures. It is crucial to get this flushed out early on so that the detector groups know what is needed. Referring to the class diagram, this would involve implementing PhaseSpaceVector (exists already), ParticleTrack, Detector, and ReconstructionInput for the inputs. ParticleTrajectory must be implemented for the outputs.

3.5.2 Interfaces

Defining the OpticsModel and TransferMap interfaces.

3.5.3 Polynomial Transfer Map

Convert existing PolynomialVector into three separate classes: PolynomialMap, PolynomialVector, and LeastSquaresOpticsModel. LeastSquaresOpticsModel would become an implementation of the OpticsModel interface. The PolynomialTransferMap class will be the first TransferMap implementation, and will also be a subclass of PolynomialMap.

3.5.4 Basic Minuit Track Fitter

The TrackFitter implementation MinuitTrackFitter will be created and tested with just fields (no materials).

3.5.5 Basic Track Reconstructor

This will likely be created in tandem with the Minuit Track Fitter to serve as an interface for testing the reconstruction. This task is to create a "map" in the Python Map/Reduce flow structure that can be inserted into the chain of execution of a MAUS simulation or live run.

3.5.6 Minuit Material Effects

Materials will be added to the transfer matrix based transport algorithm.

3.5.7 Differentiating Optics Model

The DifferentiatingOpticsModel class will be created that conforms to the OpticsModel interface. it will be tested and compared to the least squares optics model.

3.5.8 Integrating Optics Model

The IntegratingOpticsModel class will be created that conforms to the OpticsModel interface. It will be tested and compared to the differentiating and least-squares optics models.

3.5.9 Runge-Kutta Transfer Map

The RungeKuttaOpticsModel class will be created that conforms to the OpticsModel interface. This may be as simple as simple creating a RungeKuttaTransferMap instance the model does not require a matrix representation. The RungeKuttaTransferMap class will be implemented, conforming to the TransferMap interface. Again, no material effects will be included at this stage.

3.5.10 Runge-Kutta Material Effects

Materials will be incorporated into the Runge-Kutta transport algorithm.

3.5.11 Kalman Filter

The TrackFitter implementation KalmanFilterTrackFitter will be created, tested, and compared with the Minuit track fitter.

3.5.12 TOF Track Fitter

The TrackFitter implementation TOFTrackFitter will be created, tested, and compared with the Minuit and Kalman filter track fitters.

3.6 Existing Code and Location of New Code

3.7 Potentially Conflicting Work

Part II

MAUS Overview and Requirements

MAUS is an analysis tool designed for the reconstruction and analysis of the Muon Ionisation Cooling Experiment. The MAUS project uses a number of tools to manage code and the development process. Issue trackers are used to monitor development of new features and log any issues arising such as bugs. Code is stored using Version Control System (VCS). Communication between the development team is achieved using mailing lists, regular phone conferences etc. Code quality is assured by a set of style requirements, testing requirements, documentation of code and regular code reviews. These tools are documented here for convenience, but additional documentation can be found on the MAUS website http://micewww.pp.rl.ac.uk/projects/maus/wiki/MAUSDevs. Where there is a conflict, the MAUS website takes precedence as this is more likely to be updated.

3.8 Issue Tracking

The Redmine issue tracking system is used to monitor development of new features and log any issues arising such as bugs. Any information pertaining to the features in this document shall be stored in the relevant issues on that system. Redmine issues can be found by following the Issues link on the MAUS website.

3.9 Code Version Control

Code is controlled using the Bazaar Distributed Version Control System (bzr). Before writing code, developers shall checkout a copy of the maus trunk (lp:maus). Developers shall then make their own development branch on launchpad. Code shall be pushed regularly to this development branch (typically nightly). Every release, the MAUS trunk shall be merged with the development branch to ensure that development is performed against the latest MAUS version. Further guidance on bzr usage can be found on the MAUS wiki.

3.10 Communication

Communication is performed using mailing lists and fortnightly phone meetings. All developers should be subscribed to the following mailing lists: mice-software maus-devs maus-users Where possible, developers shall attend the fortnightly phone meetings, as announced on the mice-software mailing lists.

3.11 Code Quality

Code quality is ensured by a number of tools. Code must be commented and conform to certain style requirements. All code must have appropriate regression tests. Documentation must be created or updated as appropriate. Code may be reviewed by one or more developers before submission. The workflow that ensures code quality is controlled by the workflow field in the issue tracker. As code moves through the workflow, this field should be updated.

3.11.1 Coding Style and Comments

Code shall be written in the appropriate style. For C++, we follow the google style guide; for python we enforce the standard python style guide. Scripts that automatically check for code style are provided and are executed upon execution of the unit tests. All public functions shall be commented. C++ functions should have a description in the header file. Python functions should be commented using python standard docstrings. Comments should include the purpose of the function, the definition of any input parameters and the values that can be returned. Python comments should specify possible types for all input and output parameters. Comments should be formatted for the Doxygen tool.

3.11.2 Testing

Unit tests shall be provided by the developer for all code they develop at the 90% line coverage level. We aim to provide about 90% line coverage for our unit tests. Tests for modules $maps, reduces, inputs, outputs$ should be written in python and kept in the folder for that module. Tests for common code $src/common\_cpp, src/common_py$ should be written in the same language as the code to be tested and kept in tests/cpp_unit or tests/py_unit respectively. All C++ code should be compatible with the google testing framework. All python code should inherit from the python unittest framework. The MAUS test server is used to replicate environments that the code is required to be run in such as the MICE control room and standard high energy physics clusters. All tests must pass not only in the developers local environment but also in the test server environment.

3.11.3 Documentation

High level documentation should be written in latex and placed in the docdoc_src area. All user interfaces should be documented (either input or output) in latex.

3.11.4 Code Review

All code by developers new to the project shall be reviewed by the project supervisors. Additional reviewers, or code reviews, may be requested by the project supervisor. The code should be reviewed by the project supervisor when it is ready to merge and all tests pass on the test server. Code review is a way of: finding bugs spreading knowledge of the code to other developers ensuring high quality code by: checking that code is adequately tested checking that the code is documented properly checking that the code has the correct style When code is ready for review, please set the workflow field on the issue tracker to the appropriate value.

Part III

Effort and Timescale

3.12 Effort Available

3.13 Major Milestones