supported by

cooperations

Event Data Model

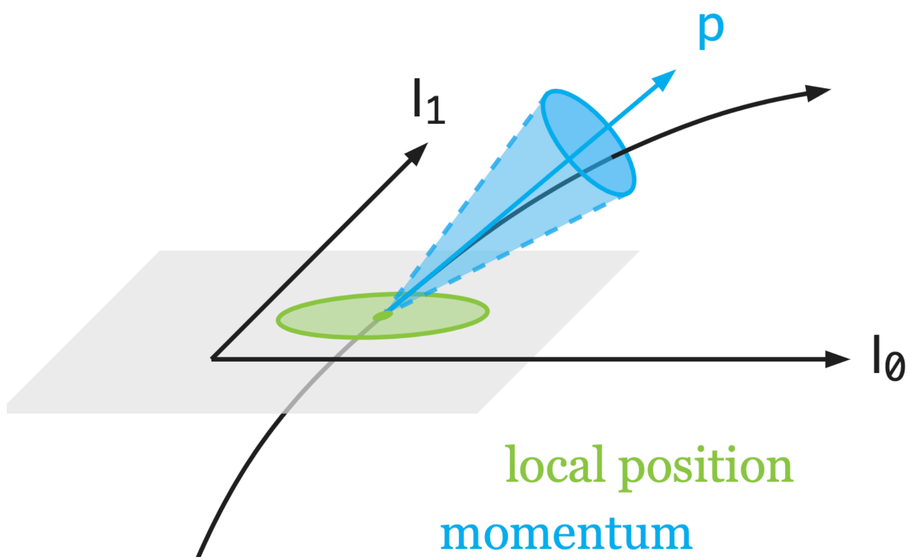
A. Salzburger (CERN) for the ACTS project

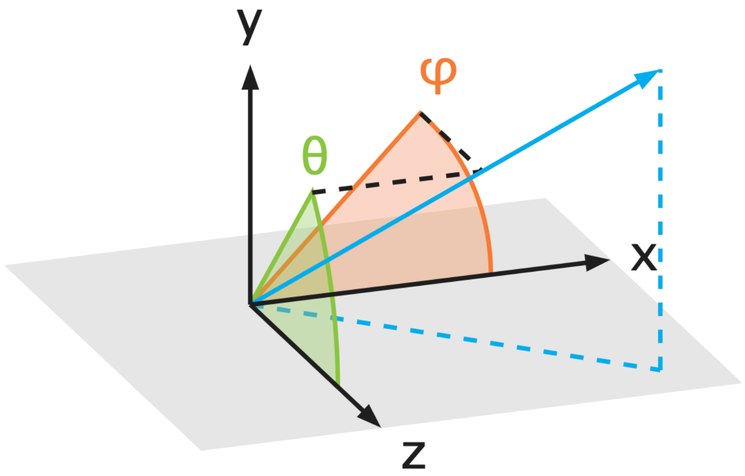
@SaltyBurger

Track parameterisation & measurements

(Bound) track parameterisation is defined:

local coordinates of the surface + global momentum





$$\vec{x} = (l_0, l_1, \phi, \theta, q/p, t)^T$$

$$C = \begin{bmatrix} \sigma^2(l_0) & \mathrm{cov}(l_0, l_1) & \mathrm{cov}(l_0, \phi) & \mathrm{cov}(l_0, \theta) & \mathrm{cov}(l_0, q/p) \\ . & \sigma^2(l_1) & \mathrm{cov}(l_1, \phi) & \mathrm{cov}(l_1, \theta) & \mathrm{cov}(l_1, q/p) \\ . & . & \sigma^2(\phi) & \mathrm{cov}(\phi, \theta) & \mathrm{cov}(\phi, q/p) \\ . & . & . & \sigma^2(\theta) & \mathrm{cov}(\theta, q/p) \\ . & . & . & . & \sigma^2(q/p) \end{bmatrix}$$

Track parameterisation & measurements

(Free) track parameterisation is defined:

global position, time, global direction, charge/p,

```
/// Components of a free track parameters vector.
///
/// To be used to access components by named indices instead of just numbers.
/// This must be a regular `enum` and not a scoped `enum class` to allow
/// implicit conversion to an integer. The enum value are thus visible directly
/// in `namespace Acts` and are prefixed to avoid naming collisions.
enum FreeIndices : unsigned int {
  // Spatial position
  // The spatial position components must be stored as one continous block.
  eFreePos0 = 0u,
  eFreePos1 = eFreePos0 + 1u,
  eFreePos2 = eFreePos0 + 2u,
  // Time
  eFreeTime = 3u,
  // (Unit) direction
  // The direction components must be stored as one continous block.
  eFreeDir0 = 4u,
  eFreeDir1 = eFreeDir0 + 1u,
  eFreeDir2 = eFreeDir0 + 2u,
  // Global inverse-momentum-like parameter, i.e. q/p or 1/p
  // See BoundIndices for further information
  eFreeQOverP = 7u,
  // Last uninitialized value contains the total number of components
  eFreeSize,
};
```

Some free measurement prototype exists from Fabian Klimpel

Track parameterisation & measurements

(Bound) track parameterisation spans the maximum space of measurements, actual measurements are given as a subspace of the bound space

- this is done such that the measurement mapping functions turn simply into projection matrices

Parameter	l_0	l_1	phi	theta	q/p	t
Bound track parameters						
Pixel measurement						
Pixel measurement with time						
Strip measurement (along local x)						
Strip measurement (along local y)						
Drift time/circle measurement						
Track segment (straight line)						
…						

Track parameterisation & measurements

(Bound) track parameterisation spans the maximum space of measurements, actual measurements are given as a subspace of the bound space

- this is done such that the measurement mapping functions turn simply into projection matrices

```
/// Source link that connects to the underlying detector readout.
const SourceLink& sourceLink() const { return m_source; }

/// Number of measured parameters.
static constexpr size_t size() { return kSize; }

/// Check if a specific parameter is part of this measurement.
bool contains(indices_t i) const { return m_subspace.contains(i); }

/// Measured parameters values.
const ParametersVector& parameters() const { return m_params; }

/// Measured parameters covariance.
const CovarianceMatrix& covariance() const { return m_cov; }

/// Projection matrix from the full space into the measured subspace.
ProjectionMatrix projector() const {
  return m_subspace.template projector<Scalar>();
}
```

Track parameterisation

Bound track parameterisation is collider-centric

- Now surprising because it stems from ATLAS
- Not optimal (at least) for fix target experiments with chambers aligned along z (LDMX has had this problem)

Initially design idea was to make track parameterisation customizable

artifacts from that time

```
// The user can override the track parameters ordering. If the preprocessor
// variable is defined, it must point to a header file that contains the same
// enum definitions for bound and free track parameters as given below.
#ifdef ACTS_PARAMETER_DEFINITIONS_HEADER
#include ACTS_PARAMETER_DEFINITIONS_HEADER
#else

namespace Acts {

// Note:
// The named indices are use to access raw data vectors and matrices at the
// lowest level. Since the interpretation of some of the components, e.g. local
// position and the inverse-momentum-like component, depend on additional
// information the names have some ambiguity. This can only be resolved at a
// higher logical level and no attempt is made to resolve it here.

/// Components of a bound track parameters vector.
///
/// To be used to access components by named indices instead of just numbers.
/// This must be a regular `enum` and not a scoped `enum class` to allow
/// implicit conversion to an integer. The enum value are thus visible directly
/// in `namespace Acts` and are prefixed to avoid naming collisions.
enum BoundIndices : unsigned int {
  // Local position on the reference surface.
  // This is intentionally named different from the position components in
  // the other data vectors, to clarify that this is defined on a surface
  // while the others are defined in free space.
  eBoundLoc0 = 0,
  eBoundLoc1 = 1,
  // Direction angles
  eBoundPhi = 2,
  eBoundTheta = 3,
  // Global inverse-momentum-like parameter, i.e. q/p or 1/p
  // The naming is inconsistent for the case of neutral track parameters where
  // the value is interpreted as 1/p not as q/p. This is intentional to avoid
  // having multiple aliases for the same element and for lack of an acceptable
  // common name.
  eBoundQOverP = 4,
```

Track parameterisation - customisable?

Propagation engine could work with different local parameters

- Local -> free -> local

Would need a huge effort in also making the freeToBound & vv for parameters and Jacobins compile-time exchangeable

- templating is not an option, would turn the entire code stack into inlined code

Development Discussion

This would be quite an huge work to establish this …

```
// The user can override the track parameters ordering. If the preprocessor
// variable is defined, it must point to a header file that contains the same
// enum definitions for bound and free track parameters as given below.
#ifdef ACTS_PARAMETER_DEFINITIONS_HEADER
#include ACTS_PARAMETER_DEFINITIONS_HEADER
#else

namespace Acts {

// Note:
// The named indices are use to access raw data vectors and matrices at the
// lowest level. Since the interpretation of some of the components, e.g. local
// position and the inverse-momentum-like component, depend on additional
// information the names have some ambiguity. This can only be resolved at a
// higher logical level and no attempt is made to resolve it here.

/// Components of a bound track parameters vector.
///
/// To be used to access components by named indices instead of just numbers.
/// This must be a regular `enum` and not a scoped `enum class` to allow
/// implicit conversion to an integer. The enum value are thus visible directly
/// in `namespace Acts` and are prefixed to avoid naming collisions.
enum BoundIndices : unsigned int {
  // Local position on the reference surface.
  // This is intentionally named different from the position components in
  // the other data vectors, to clarify that this is defined on a surface
  // while the others are defined in free space.
  eBoundLoc0 = 0,
  eBoundLoc1 = 1,
  // Direction angles
  eBoundPhi = 2,
  eBoundTheta = 3,
  // Global inverse-momentum-like parameter, i.e. q/p or 1/p
  // The naming is inconsistent for the case of neutral track parameters where
  // the value is interpreted as 1/p not as q/p. This is intentional to avoid
  // having multiple aliases for the same element and for lack of an acceptable
  // common name.
  eBoundQOverP = 4,
```

Measurements and source links

The ACTS measurement class contains only the mathematical representation needs for track fitting (parameter set, covariances and surface)

- experiment specific, detector specifics can be packed into a source link object

```
/// Source link that connects to the underlying detector readout.
const SourceLink& sourceLink() const { return m_source; }
```

- Calibration code, etc. needs to unpack this and interpret the source link object
- As this is usually all from one common implementation, this unpacking can be practically done cost-free, e.g. with static cast

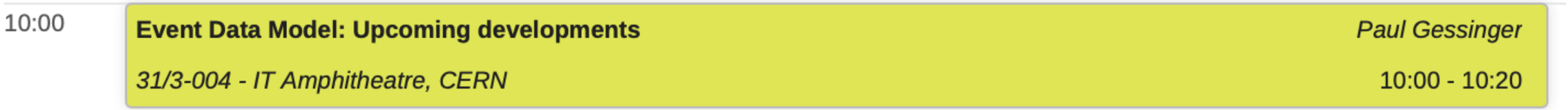
Development Proposal

We have an alignment demonstrator in the Examples, should also make a calibration demonstrator as well.

MultiTrajectory + Track class

Track finding and fitting share a common data model backend

- The multi trajectory allows to reuse track states, e.g. in a combinatorial track finding (CKF) (or also in the GSF) and stores tree descriptions of the track
- By reversing the valid track tips from CKF one can build the tracks
- In order to simplify the memory layout we initially overcommitted to full dimension (has shown to be very memory hungry)



Track object & Track summary objects are not yet defined

Development Proposal

Create a top level view object of MultiTrajectories that can be further used for ambiguity solving, vertex fitting and tracking performance evaluation.