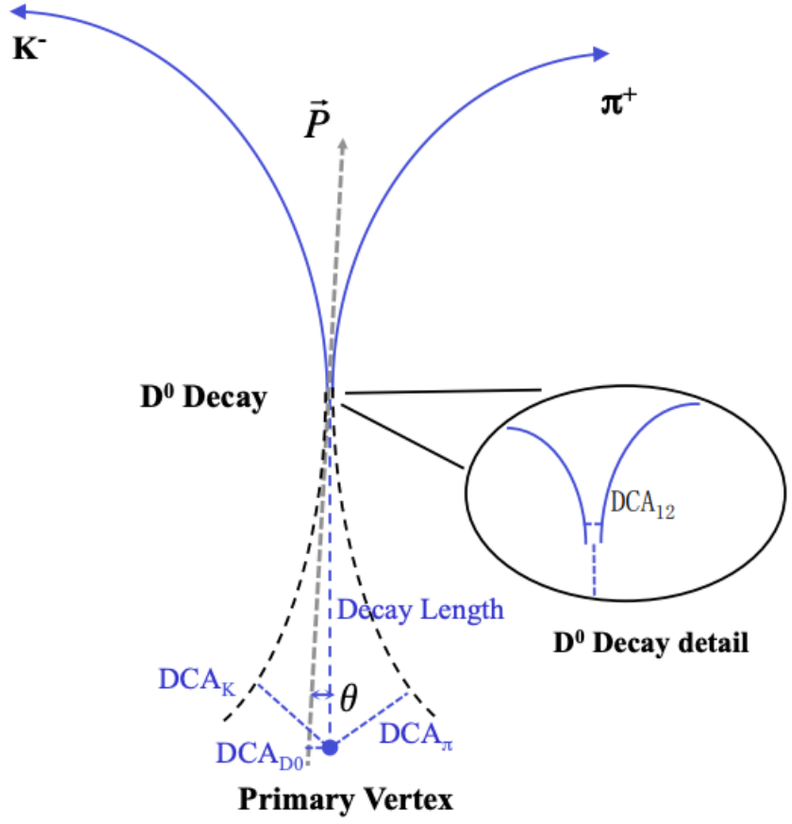
Helix and SecondaryVertices Factory in ElCrecon

Xin Dong



Thanks to:

Bishoy Dongwi, Lokesh Kumar, Rongrong Ma, Joe Osborn, Ashish Pandav, *Harsimran Singh*, *Khushi Singla*, Deepa Thomas, Connie Yang etc.

Previous presentation:
https://indico.bnl.gov/event/29424/contributions/112551/attachments/64322/110448/20250825_ePIC_Helix.pdf

Vertex Finders

ACTS Track Objects

ReconstructedParticles

Vertex Finder(s) (Primary)

(Secondary)

Acts-based AMVF Finder
- PR # 1915 - *Bishoy*

CentralTrackVertices

ACTS objs
PODIO output
EICrecon ready
Under development

2025/08/25 Proposal

User Analysis Vertex Finders (Helix/KFParticle)

a. Helix swimming (adopted from STAR)
   - helix utils to edm4eic - PR # 115
b. KFParticle
   - used by STAR/sPHENIX/ALICE/CBM etc.
   - under testing now, to be deployed later

Implementation Change

2025/10/13 Proposal

EICrecon

reco/SecondaryVertices (Helix/KFParticle)

a. reco/Helix.*
b. KFParticle standalone package working on integration

SecondaryVerticesXXX

EICrecon branch: pr/secondaryvertex-helix

- src/algorithms/reco/Helix.cc
- src/algorithms/reco/Helix.h

adopted from STAR code, adjusted for EICrecon

```
/// ReconstructParticle, b field
Helix(const edm4eic::ReconstructedParticle& p, const double b_field);
```

- src/algorithms/reco/SecondaryVerticesHelix.cc
- src/algorithms/reco/SecondaryVerticesHelix.h
- src/algorithms/reco/SecondaryVerticesHelixConfig.h
- src/factories/reco/SecondaryVerticesHelix_factory.h

1. new SecondaryVertices factory, now based on Helix, can be expanded for others (e.g. KFParticle)

2. take ReconstructedParticle as input, run secondary finder (Ks included so far, can be expanded to include others)

3. output -> SecondaryVerticesHelix (edm4eic::Vertex) in PODIO
   - *proposed to add edm4eic::SecondaryVertex container (next slide)*

4. geometric cut variables controlled by SecondaryVerticesHelixConfig.h

- src/global/reco/reco.cc
- src/services/io/podio/JEventProcessorPODIO.cc

All codes have been compiled in EICrecon, run successfully and produce meaningful output.

SecondaryVerticesHelix Factory

```
void SecondaryVerticesHelix::process(const SecondaryVerticesHelix::Input& input,
                                     const SecondaryVerticesHelix::Output& output) const {
  const auto [rcvtx, rcparts] = input;
  auto [out_secondary_vertices] = output;

  auto& particleSvc = algorithms::ParticleSvc::instance();
//  edm4hep::Vector3f pVtxPos;
//  for(const auto& v : primVtx)
  const auto pVtxPos4f = (*rcvtx)[0].getPosition();
  // convert to cm
  edm4hep::Vector3f pVtxPos(pVtxPos4f.x*edm4eic::unit::mm/edm4e
                            pVtxPos4f.y*edm4eic::unit::mm/edm4e
                            pVtxPos4f.z*edm4eic::unit::mm/edm4e
  info("\t Primary vertex = ({},{},{})cm \t b field = {} tesla",
pVtxPos.z, m_cfg.b_field/dd4hep::tesla);

  std::vector<unsigned int> pi_index;
  std::vector<unsigned int> k_index;
  std::vector<unsigned int> p_index;
  for (unsigned int i = 0; const auto& p : *rcparts) {
    const auto pdg = p.getPDG();
    if(abs(pdg) == 211) pi_index.push_back(i);
    if(abs(pdg) == 321) k_index.push_back(i);
    if(abs(pdg) == 2212) p_index.push_back(i);
    ++i;
  }

  info("\t Array sizes: pions  = {}, kaons = {}, protons = {}",
k_index.size(), p_index.size());
```

```
for (unsigned int i1 = 0; i1 < pi_index.size(); ++i1) {
  for (unsigned int i2 = i1 + 1; i2 < pi_index.size(); ++i2) {
     const auto& p1 = (*rcparts)[i1];
     const auto& p2 = (*rcparts)[i2];

     if (p1.getCharge() + p2.getCharge() != 0) continue;

     Helix h1obj(p1, m_cfg.b_field);  Helix& h1 = h1obj;
     Helix h2obj(p2, m_cfg.b_field);  Helix& h2 = h2obj;

     // Helix function uses cm unit
     double dca1 = h1.distance(pVtxPos) * edm4eic::unit::cm;
     double dca2 = h2.distance(pVtxPos) * edm4eic::unit::cm;
     debug("\t dca1 = {}, dca2 = {}", dca1, dca2);
     if( dca1 < m_cfg.minDca1 || dca2 < m_cfg.minDca2 ) continue;

     std::pair<double, double> const ss = h1.pathLengths(h2);
     edm4hep::Vector3f h1AtDcaTo2 = h1.at(ss.first);
     edm4hep::Vector3f h2AtDcaTo1 = h2.at(ss.second);

     double dca12 = edm4hep::utils::magnitude(h1AtDcaTo2 - h2AtDcaTo1) *
edm4eic::unit::cm;
     if( dca12 > m_cfg.maxDca12 ) continue;
     edm4hep::Vector3f pairPos = 0.5*(h1AtDcaTo2 + h2AtDcaTo1);

     edm4hep::Vector3f h1MomAtDca = h1.momentumAt(ss.first, m_cfg.b_field);
     edm4hep::Vector3f h2MomAtDca = h2.momentumAt(ss.second, m_cfg.b_field);
     edm4hep::Vector3f pairMom = h1MomAtDca + h2MomAtDca;

     float e1 = std::hypot(edm4hep::utils::magnitude(h1MomAtDca),
particleSvc.particle(211).mass);
     float e2 = std::hypot(edm4hep::utils::magnitude(h2MomAtDca),
particleSvc.particle(211).mass);
     float pairE = e1+e2;

     edm4hep::Vector4f h1FourMom(h1MomAtDca.x, h1MomAtDca.y, h1MomAtI
     edm4hep::Vector4f h2FourMom(h2MomAtDca.x, h2MomAtDca.y, h2MomAtI

     double m_inv = std::hypot(pairE, -edm4hep::utils::magnitude(pai
     double angle = edm4hep::utils::angleBetween(pairMom, pairPos -
     if(cos(angle) < m_cfg.minCostheta ) continue;

     double beta = edm4hep::utils::magnitude(pairMom)/pairE;
     double time = edm4hep::utils::magnitude(pairPos - pVtxPos)/(beta*dd4hep::c_light);
     auto v0 = out_secondary_vertices->create();
     v0.setType(2); // 2 for secondary
     v0.setPosition({(float)(pairPos.x * edm4eic::unit::cm / edm4eic::unit::mm),
                     (float)(pairPos.y * edm4eic::unit::cm / edm4eic::unit::mm),
                     (float)(pairPos.z * edm4eic::unit::cm / edm4eic::unit::mm),
                     (float)time});
     v0.addToAssociatedParticles(p1);
     v0.addToAssociatedParticles(p2);

     info("One secondary vertex found at (x,y,z) = ({}, {}, {}) mm.",
               pairPos.x * edm4eic::unit::cm / edm4eic::unit::mm,
               pairPos.y * edm4eic::unit::cm / edm4eic::unit::mm,
               pairPos.x * edm4eic::unit::cm / edm4eic::unit::mm);

   } // end i2
  } // end i1

} // end process
```

SecondaryVertex to edm4eic

1. Current edm4eic::vertex object doesn't contain many topological variables for secondary vertices.
2. Though one can in principle re-calculate all these based on the associated ReconstructedParticle, it will be much more convenient to save them during reconstruction and down stream analysis can directly use them for physics analysis and also this will avoid repeated calculations.

Propose to add edm4eic::SecondaryVertex container

```
edm4eic::Vertex:
  Description: "EIC vertex"
  Author: "J. Osborn"
  Members:
    - int32_t             type          // Type flag, to identify wha
    - float               chi2          // Chi-squared of the vertex
    - int                 ndf           // NDF of the vertex fit
    - edm4hep::Vector4f   position      // position [mm] + time t0 [n
    ## this is named "covMatrix" in EDM4hep, renamed for consistency
    - edm4eic::Cov4f      positionError // Covariance matrix of the p
  OneToManyRelations:
    - edm4eic::ReconstructedParticle associatedParticles // particles
```

```
edm4hep::Vertex:
  Description: "Vertex"
  Author: "EDM4hep authors"
  Members:
    - uint32_t            type          // flagword that defines the type of
    - float               chi2          // chi-squared of the vertex fit
    - int32_t             ndf           // number of degrees of freedom of th
    - edm4hep::Vector3f   position [mm]     //  position of the vertex
    - edm4hep::CovMatrix3f covMatrix [mm^2]  // covariance matrix of the posi
    - int32_t                 algorithmType // type code for the algorithm th
  VectorMembers:
    - float               parameters    // additional parameters related to this vertex
  OneToManyRelations:
    - edm4hep::ReconstructedParticle particles // particles that have been used to form this vertex, aka the d
```

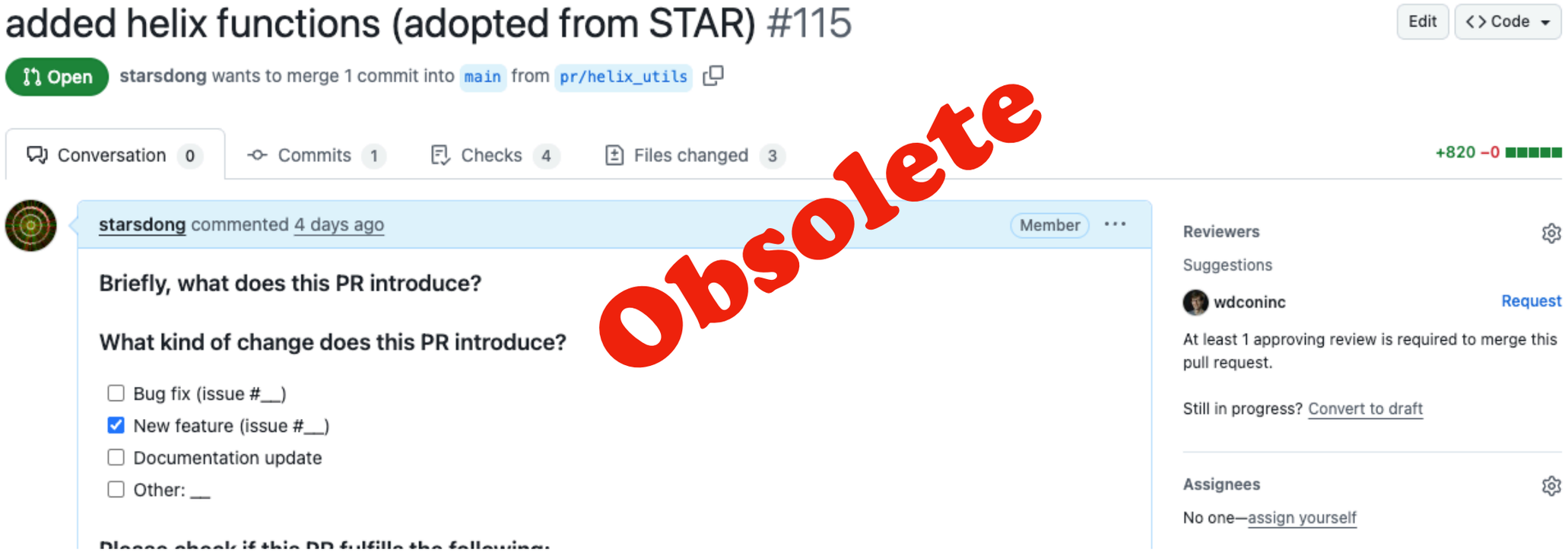
```
edm4eic::SecondaryVertex:
  Description: "EIC secondary vertex"
  Author: "SV authors"
  Members:
    - int32_t             type          // Type flag, to identify what type of vertex it is (e.
    - float               chi2          // Chi-squared of the vertex fit
    - int                 ndf           // NDF of the vertex fit
    - edm4hep::Vector4f   position      // position [mm] + time t0 [ns] of the vertex. Time is
    ## this is named "covMatrix" in EDM4hep, renamed for consistency with the rest of edm4eic
    - edm4eic::Cov4f      positionError // Covariance matrix of the position+time. Time is 4th
    - emd4hep::Vector4f   parentMomentum  // parent momentum, energy
    - edm4hep::Vector3f   parentL               // parent decay length L
    - float               parentLChi2           // parent L/dL
    - float               parentDca2PV          // parent dca to primary vertex
    - float               parentDca2PVChi2      // parent dca/sigma to primary vertex
  OneToOneRelation:
    - edm4eic::Vertex     primaryVertex      // associated primary vertex
  VectorMembers:                             // all daughter information
    - emd4hep::Vector3f   daughterMomentum   // daughter track momentum
    - float               daughterMass       // daughter mass
    - float               daughterDca2PV     // daughter dca to primary vertex
    - float               daughterDca2PVChi2 // daughter dca/sigma to primary vertex
  VectorMembers:                             // daughter pair information
    - int                 daughterPairIndices    // pair track indices
    - float               daughterPairDca        // pair dca to primary vertex
    - float               daughterPairDcaChi2    // pair dca/sigma to primary vertex
  OneToManyRelations:
    - edm4eic::ReconstructedParticle daughterParticles // particles associated to this vertex.
```

Summary

1. If this approach is agreed, we plan to submit the PR based on pr/secondaryvertex-helix first
2. We will propose a new edm4eic container: SecondaryVertex
3. Continue the development of secondary VFs (Helix with more particles included, KFParticle etc.)

Backup

Adding Helix Functions in EDM4eic (obsolete!)



1) Helix afterburner reconstruction is used in D^0 reconstruction (later part), targeted to be used for updated physics projection plots.
2) Constructor includes using EICrecon TrackParameters as input.
3) Handling constant z-magnetic field (or zero field - straight-line)
   - can be extended to handle varying B-field for track projection
4) Iterative varying-step-scan to find DCA positions between helices numerically.

Usage of Helix Method

Helix method has been used in many heavy flavor hadron analysis (Rongrong/Shyam/Connie etc.)

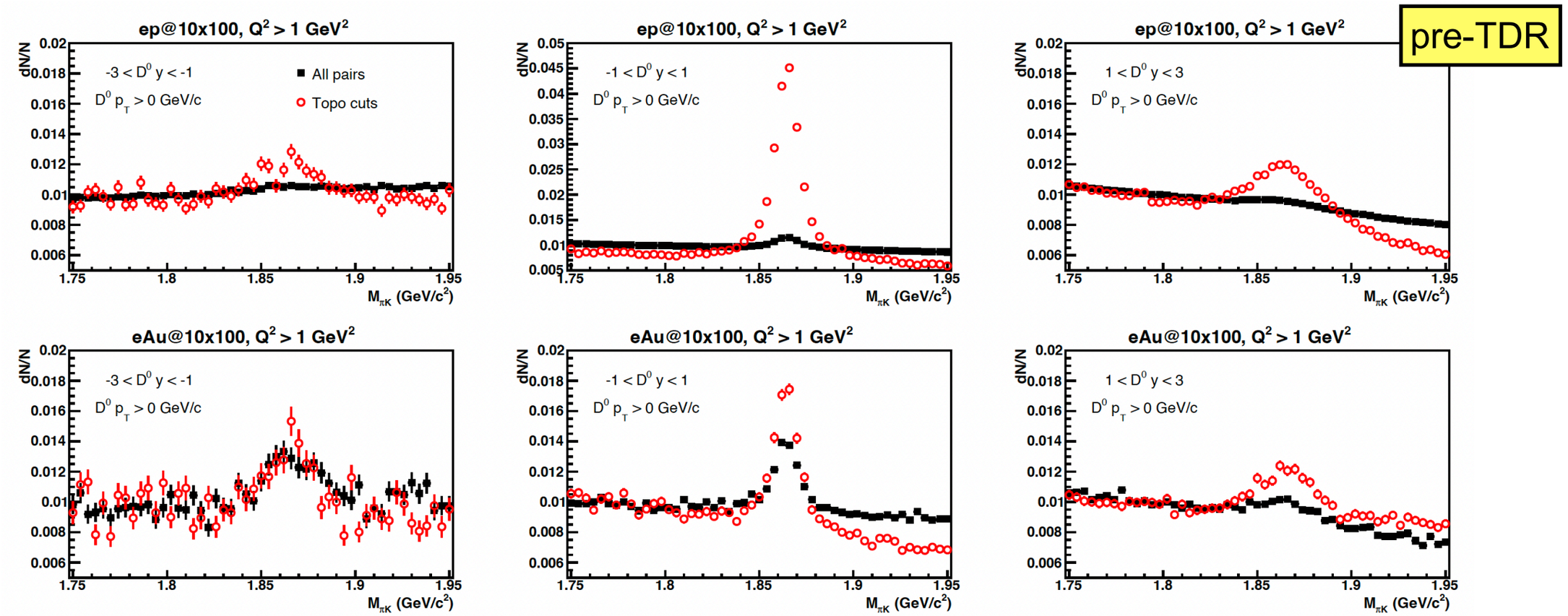


Figure 2.22: Invariant mass distributions of $\pi + K$ pairs with (red circles) and without (black squares) topological selections in 10×100 GeV e+p (top) and e+Au (bottom) collisions with a minimum Q^2 of 1 GeV^2. Different panels from left to right correspond to different D^0 rapidity intervals: $-3 < y < -1$ (left), $-1 < y < 1$ (middle) and $1 < y < 3$ (right).

Example of using Helix method:
https://github.com/marrbnl/ePIC/tree/main/HF_reco/helix