VXDTF2 activities

Eugenio, Thomas Lück

Last weeks activities

- Knowledge transfer from Eugenio to Thomas (in progress)
- Code review while transferring the knowledge
- Review of the Todo scattered over the VXDTF code (order of 60):
  - Prioritize the list (still in progress)
  - Work on the most dangerous one (in my view)

First To Do in detail

- Persist the SectorMap so that one can train it once and debug it forever.
- The SectorMap2 is more flexible w.r.t. the SectorMap1 most notably:
- In Sector Map1 all the sensors are made equal:
  i.e. all the sensors are divided at u {.15, .5, .85, 1.} and v {.1, .3, .5, .7, .9, 1.} (aka 4x6)
- In Sector Map 2 each sensor is divided at user wills
  - Sector map granularity can depend on layer number, ladder position
  - Possible to tune the sectors for broken chip on a given sensor

First To Do in detail

- Class: `trackFindingVXD/sectorMapTools/SecMapTrainer.hh`

~ ok

! ok

BAD

```
** for given normalized local variables and VxdID a FullSecID is determined.
 * returns unsigned int max if correct ID could not be found. */
FullSecID createSecID(VxdID iD, double uVal, double vVal)
{
  // TODO replace by new secMap-design-approach
  std::vector<double> uTemp = {0.};
  uTemp.insert(uTemp.end(), m_config.uSectorDivider.begin(), m_config.uSectorDivider.e\
nd());
  std::vector<double> vTemp = {0.};
  vTemp.insert(vTemp.end(), m_config.vSectorDivider.begin(), m_config.vSectorDivider.e\
);
  auto secID = SectorTools::calcSecID(uTemp, vTemp, {uVal, vVal});
  if (secID == std::numeric_limits<unsigned short>::max())
    return FullSecID(std::numeric_limits<unsigned int>::max());
  return FullSecID(iD, false, secID);
}
```

- SectorTools is a relic of VXDTF1

The implemented method as requested by Jakob.

```
/// returns fullSecID for given sensorID and local coordinates.
// JKL: what happens here if no FullSecID could be found?
// EP: you will get an exception, as you wrote an exception throw over there...
FullSecID VXDTFFilter::getFullID(VxdID aSensorID, double normalizedU, double normalizedV) const
{
   // TODO WARNING how to catch bad cases?
   return m_compactSecIDsMap.getFullSecID(aSensorID, normalizedU, normalizedV);
}
```

```
FullSecID CompactSecIDs::SecID(VxdID aSensorID,
                               double normalizedU, double normalizedV) const
{

      auto layer = aSensorID.getLayerNumber() ;
      auto ladder = aSensorID.getLadderNumber();
      auto sensor = aSensorID.getSensorNumber();

      auto sectorsOnSensor =
        m_compactSectorsIDMap.at(layer).at(ladder).at(sensor);

      if (normalizedU < 0. || 1. < normalizedU)
        throw(unboundedNormalizedU()
              << layer << ladder << sensor << normalizedU);
     ...

      return sectorsOnSensor(normalizedU, normalizedV);

}
```

Next week plan

- Get rid of the relic SectorTools
  - Put it back in the VXDTF1 directory tree (since VXDTF1 is still using it)
- Get rid of duplicated information in the SectorMapConfig namely:
  - the uSectorDivider and vSectorDivider
  - the filter names