AWL Lidar Sensor Libraries

## Software Library Layers

The C++ library is structured in layers for platform-independence.

It relies on the standard C++ *STL* library as much as possible for support of the basic logic and addditional items :

* Sized data types when they are required (ex: uint8\_t)
* String types std::string

For the support of some of the trickier OS-dependent features, some elements of the BOOST library are used:

* Shared Pointer types (ex: boost::shared\_ptr)
  + Guarantee deletion of the pointer in the lifecycle of the object and avoids mix-ups in multi-threaded sharing of information.
* Containers (ex: boost::container::vector)
  + Again we favor boost::container objects over the STL equivalents because they use stateful allocators and are more compatible with shared memory programming.
* Threads
  + Just so we have a good layer of platform-independence and avoid making direct OS calls.
* Serial communications (boost:asio)

At the user interface level,the software should rely on the rely on the QT Library.

* An open-source version of QT is available and provides an easy migration path to many platforms (Windows, Linux, Mac, Android).
* We take great care of decoupling user interface libraries from the low level acquisition and processing libraries. Users who wish to develop interfaces based on a different GUI platform may choose to do so.
* For programmers, this means that we have to be extra careful not to use QT data types in low-end layers of the programes.
* As a side effect, the UI level of the demo code, you may get a mix of QT types and BOOST types performing similar functions (ex: QVector vs bost::container::vector, thread management).

Some parts of the demo software display possibilities of interfacing with OpenCV (for video capture / display) and mathematical acceleration of image processing.

## Application concepts

At the core, user-interface independent elements of the software relies on some important functional elements:

* Acquiring data from the physical AWL units \*(*ReceiverCapture* and derived classes).
* 3D coordinate calculations (CoordinateSystem).
* Integrating above concepts in the physical configuration of the application (*ReceiverPostProcessor* and *AWLCoord*).
* Managing threaded and application controlled loops (*TheadedWorker* and *LoopedWorker*).
* Establishing a Publisher/Subscriber mechanism for user interface objects wishing to access data asynchronously (*Publisher* / *Subscription*).