6 Aug 2014

Odometry system

The odometry process proposed is based on calculations from stereo images.

The ***Odometry*** class processes new frames to track features and their location in a reference coordinate system. Then tracks the camera pose in this reference coordinate system from the known points using the ***solvePNP*** OpenCV function.

Tracking algorithm.

On each loop the algorithm updates the camera position in a reference coordinate system, based on known points which are called verified points, and adds some new points to the map. For a point to become verified it has to be seen a number of times, this is set with variable *sightingsToVerify*. If a verified point is no longer seen it is stored in a point cloud for later usage (i.e. making a map of the area).

For the algorithm to work there should be always some verified points being observed, so the camera speed and frame rate are important factors to take in account. In addition, to start the odometry algorithm the verified point feed and descriptor feed should be filled with initial points.

The following variables are the ones used by the algorithm:

```
vector<pointAndFeat> pointCloud;//Storage for verified points once they are no longer seen
vector<Point3d> vPointFeed;
Mat uDescriptorFeed;
Mat vDescriptorFeed;
vector<int> sightings;        //Sighting counter for unverified points.
vector<int> unSightings;      //Unsighting counter for unverified points.
vector<int> vunSightings;     //Unsighting counter for verified points.

       //Parameters and thresholds
int sightingsToVerify;  //how many a new key point has to bee detected to become validated.
int unSightingsToStore; //How many times a point has to not be detected to be stored.
int unSightingsToDelete;//How many times an unverified descriptor has to be unseen to be deleted.
```

Odometry loop:

1. Invoke ***camera.updateStereoPair()***, get the main image (the one which pixels will coincide with the depth map's pixels), find key points and compute descriptors.
2. Look for matches with the ***vDescriptorFeed***, update ***vunSightings***, add the descriptors and points that verify ***vunsightings == unSightingsToStore*** to ***pointCloud*** and delete them from ***vDescriptorFeed*** and ***vPointFeed***.
3. Compute ***solvePNP()*** with the matches with ***vDescriptorFeed***.
4. Look for matches with the ***uDescriptorFeed*** and update ***sightings*** and ***unSightings.***
5. Add the descriptors that verify ***sightings == sightingsToVerify*** to the ***vDescriptorFeed***, compute the corresponding 3D point in the local coordinate system, translate it to the reference coordinate system and add it to the ***vPointFeed***.
6. Delete the descriptors from ***uDescriptorFeed*** that verify ***unSightings == unSightingsToDelete***.
7. Add the unmatched descriptors to ***uDescriptorFeed.***