Hints 1

Warning: the depth images are stored with 16 bit depth, so in this case calling the cv::imread() function you shoud specify the flag cv: IMREAD_ANYDEPTH:

```
cv::Mat input_depth = cv::imread("test_depth.png", cv:: IMREAD_ANYDEPTH);
```

Warning: the input depth images should be scaled by a 0.01 factor in order to obtain distances in meters. You could use the openCv function

```
input_depth_img.convertTo(scaled_depth_img, CV_32F,0.001);
```

As camera matrix, use the following default matrix

```
float fx = 512, fy = 512, cx = 320, cy = 240;
Eigen::Matrix3fcamera_matrix;
camera_matrix << fx, 0.0f, cx, 0.0f, fy, cy, 0.0f, 0.0f,1.0f;
```

As re-projection matrix, use the following matrix:

```
Eigen::Matrix4f t_mat;
t_mat.setIdentity();
t_mat.block<3, 3>(0, 0) = camera_matrix.inverse();
```

For each (x,y) with depth, obtain the corresponding 3D points:

```
Eigen::Vector4f point = t_mat * Eigen::Vector4f(x*d, y*d, d, 1.0);
```

(the last coordinate of point can be ignored)

Hint 2

Warning: Since we are working with organized point clouds, also points with depth equal to 0 that are not valid, should be added to the computed cloud as NaN, i.e. in pseudocode::

```
const float bad_point = std::numeric_limits<float>::quiet_NaN();
      if( depth(x, y) == 0) { p.x = p.y = p.z = bad_point;}
```

To get the global transform of the current cloud just perform the following multiplication after you computed the registration:

```
Eigen::Matrix4f globalTransform = previousGlobalTransform *
                    alignmentTransform;
```

PreviousGlobalTransform is the global transformation found for the previous point cloud
AlignmentTransform is the local transform computed using Generalized ICP
Warning: the first global transform has to be initialized to the identity matrix

Kinect topics

```
"/camera/depth_registered/image_rect_raw"
"/camera/rgb/image_rect"
```

Topic subsctiption, syncronization and callback registration

```
#include <message_filters/subscriber.h>
#include<message_filters/synchronizer.h>#include
<message_filters/sync_policies/approximate_time.h>
ros::NodeHandlenh;message_filters::Subscriber<Image> depth_sub(nh,
"topic1", 1);  message_filters::Subscriber<Image> rgb_sub(nh, "topic2",
1);  typedef sync_policies::ApproximateTime<Image, Image> syncPolicy;
Synchronizer<syncPolicy> sync(syncPolicy(10), depth_sub, rgb_sub);
      sync.registerCallback(boost::bind(&callback, _1,_2));
```