Pose from homography estimation

Introduction

The homography can be decomposed to retrieve the pose. We consider here that all the points lie in the plane $^wZ = 0$.

Source code

The following source code that uses OpenCV is also available in **pose-from-homography-dlt-opencv.cpp** file. It allows to compute the pose of the camera from at least 4 coplanar points.

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/calib3d/calib3d.hpp>

cv::Mat homography_dlt(const std::vector< cv::Point2d >
       &x1, const std::vector< cv::Point2d > &x2)
{
  int npoints = (int)x1.size();
  cv::Mat A(2*npoints, 9, CV_64F, cv::Scalar(0));

  // We need here to compute the SVD on a (n*2)*9 matrix
       (where n is
  // the number of points). if n == 4, the matrix has
       more columns
  // than rows. The solution is to add an extra line with
       zeros
  if (npoints == 4)
    A.resize(2*npoints+1, cv::Scalar(0));

  // Since the third line of matrix A is a linear
       combination of the first and second lines
  // (A is rank 2) we don't need to implement this third
       line
  for(int i = 0; i < npoints; i++) {              //
       Update matrix A using eq. 23
```

```
    A.at<double>(2*i,3) = -x1[i].x;                    // -xi_1
    A.at<double>(2*i,4) = -x1[i].y;                    // -yi_1
    A.at<double>(2*i,5) = -1;                          // -1
    A.at<double>(2*i,6) =  x2[i].y * x1[i].x;          // yi_2 * xi_1
    A.at<double>(2*i,7) =  x2[i].y * x1[i].y;          // yi_2 * yi_1
    A.at<double>(2*i,8) =  x2[i].y;                    // yi_2

    A.at<double>(2*i+1,0) =  x1[i].x;                  // xi_1
    A.at<double>(2*i+1,1) =  x1[i].y;                  // yi_1
    A.at<double>(2*i+1,2) =  1;                        //  1
    A.at<double>(2*i+1,6) = -x2[i].x * x1[i].x;        // -xi_2 * xi_1
    A.at<double>(2*i+1,7) = -x2[i].x * x1[i].y;        // -xi_2 * yi_1
    A.at<double>(2*i+1,8) = -x2[i].x;                  // -xi_2
  }

  // Add an extra line with zero.
  if (npoints == 4) {
    for (int i=0; i < 9; i ++) {
      A.at<double>(2*npoints,i) = 0;
    }
  }

  cv::Mat w, u, vt;
  cv::SVD::compute(A, w, u, vt);

  double smallestSv = w.at<double>(0, 0);
  unsigned int indexSmallestSv = 0 ;
  for (int i = 1; i < w.rows; i++) {
    if ((w.at<double>(i, 0) < smallestSv) ) {
      smallestSv = w.at<double>(i, 0);
      indexSmallestSv = i;
    }
  }

  cv::Mat h = vt.row(indexSmallestSv);

  if (h.at<double>(0, 8) < 0) // tz < 0
    h *=-1;

  cv::Mat _2H1(3, 3, CV_64F);
  for (int i = 0 ; i < 3 ; i++)
    for (int j = 0 ; j < 3 ; j++)
      _2H1.at<double>(i,j) = h.at<double>(0, 3*i+j);

  return _2H1;
```

```
}

void pose_from_homography_dlt(const std::vector<
       cv::Point2d > &xw,
                                    const std::vector<
       cv::Point2d > &xo,
                                    cv::Mat &otw, cv::Mat &oRw)
{
  cv::Mat oHw = homography_dlt(xw, xo);

  // Normalization to ensure that ||c1|| = 1
  double norm = sqrt(oHw.at<double>(0,0)*oHw.at<double>
       (0,0)
                      + oHw.at<double>(1,0)*oHw.at<double>
       (1,0)
                      + oHw.at<double>(2,0)*oHw.at<double>
       (2,0));
  oHw /= norm;

  cv::Mat c1  = oHw.col(0);
  cv::Mat c2  = oHw.col(1);
  cv::Mat c3 = c1.cross(c2);

  otw = oHw.col(2);

  for(int i=0; i < 3; i++) {
    oRw.at<double>(i,0) = c1.at<double>(i,0);
    oRw.at<double>(i,1) = c2.at<double>(i,0);
    oRw.at<double>(i,2) = c3.at<double>(i,0);
  }
}

int main()
{
  int npoints = 4;

  std::vector< cv::Point3d > wX;  // 3D points in the
       world plane

  std::vector< cv::Point2d > xw;  // Normalized
       coordinates in the object frame
  std::vector< cv::Point2d > xo;  // Normalized
       coordinates in the image plane

  // Ground truth pose used to generate the data
  cv::Mat otw_truth = (cv::Mat_<double>(3,1) << -0.1,
       0.1, 1.2); // Translation vector
  cv::Mat orw_truth = (cv::Mat_<double>(3,1) <<
       CV_PI/180*(5), CV_PI/180*(0), CV_PI/180*(45)); //
       Rotation vector
  cv::Mat oRw_truth(3,3,cv::DataType<double>::type); //
       Rotation matrix
  cv::Rodrigues(orw_truth, oRw_truth);
```

```
  // Input data: 3D coordinates of at least 4 coplanar
      points
  double L = 0.2;
  wX.push_back( cv::Point3d(  -L, -L, 0) ); // wX_0 (-L,
      -L, 0)^T
  wX.push_back( cv::Point3d( 2*L, -L, 0) ); // wX_1 ( L,
      -L, 0)^T
  wX.push_back( cv::Point3d(   L,  L, 0) ); // wX_2 ( L,
      L, 0)^T
  wX.push_back( cv::Point3d(  -L,  L, 0) ); // wX_3 (-L,
      L, 0)^T

  // Input data: 2D coordinates of the points on the
      image plane
  for(int i = 0; i < wX.size(); i++) {
    cv::Mat oX = oRw_truth*cv::Mat(wX[i]) + otw_truth; //
      Update oX, oY, oZ
    xo.push_back( cv::Point2d( oX.at<double>(0,
      0)/oX.at<double>(2, 0),
                               oX.at<double>(1,
      0)/oX.at<double>(2, 0) ) ); // xo = (oX/oZ, oY/oZ)

    xw.push_back( cv::Point2d( wX[i].x, wX[i].y ) ); //
      xw = (wX, wY)
  }

  cv::Mat otw(3, 1, CV_64F); // Translation vector
  cv::Mat oRw(3, 3, CV_64F); // Rotation matrix

  pose_from_homography_dlt(xw, xo, otw, oRw);

  std::cout << "otw (ground truth):\n" << otw_truth <<
      std::endl;
  std::cout << "otw (computed with homography DLT):\n" <<
      otw << std::endl;
  std::cout << "oRw (ground truth):\n" << oRw_truth <<
      std::endl;
  std::cout << "oRw (computed with homography DLT):\n" <<
      oRw << std::endl;

  return 0;
}
```

Source code explained

First of all we include OpenCV headers that are requested to manipulate vectors and matrices.

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/calib3d/calib3d.hpp>
```

Then we introduce the function that does the homography estimation from coplanar points. This function is detailed in **Homography estimation**.

```
cv::Mat homography_dlt(const std::vector< cv::Point2d >
      &x1, const std::vector< cv::Point2d > &x2)
```

Then we introduce the function that does the pose from homography estimation.

```
void pose_from_homography_dlt(const std::vector<
      cv::Point2d > &xw,
                            const std::vector<
      cv::Point2d > &xo,
                            cv::Mat &otw, cv::Mat &oRw)
```

Based on equation (27) $\mathbf{x}_0 = {}^0\mathbf{H}_w \mathbf{x}_w$ we first estimate the homography ${}^0\mathbf{H}_w$.

```
  cv::Mat oHw = homography_dlt(xw, xo);
```

Then using the constraint that $||\mathbf{c}^0_1|| = 1$ we normalize the homography.

```
  // Normalization to ensure that ||c1|| = 1
  double norm = sqrt(oHw.at<double>(0,0)*oHw.at<double>
      (0,0)
                    + oHw.at<double>(1,0)*oHw.at<double>
      (1,0)
                    + oHw.at<double>(2,0)*oHw.at<double>
      (2,0));
  oHw /= norm;
```

Let us denote $\mathbf{M} = \Pi^{-1}\, {}^0\mathbf{H}_w$

Noting that matrix M is also equal to $\mathbf{M} = (\mathbf{c}^0_1, \mathbf{c}^0_2, {}^0\mathbf{T}_w)$ we are able to extract the corresponding vectors.

```
  cv::Mat c1  = oHw.col(0);
  cv::Mat c2  = oHw.col(1);
```

The third column of the rotation matrix is computed such as $\mathbf{c}^0_3 = \mathbf{c}^0_1 \times \mathbf{c}^0_2$

```
  cv::Mat c3 = c1.cross(c2);
```

To finish we update the homogeneous transformation that corresponds to the estimated pose ${}^0\mathbf{T}_w$.

```
otw = oHw.col(2);

for(int i=0; i < 3; i++) {
  oRw.at<double>(i,0) = c1.at<double>(i,0);
  oRw.at<double>(i,1) = c2.at<double>(i,0);
  oRw.at<double>(i,2) = c3.at<double>(i,0);
}
```

Finally we define the main function in which we will initialize the input data before calling the previous function and computing the pose from the estimated homography.

```
int main()
```

Then we create the data structures that will contain the 3D points coordinates *wX* in the world frame, their normalized coordinates *xw* in the world frame and their normalized coordinates *xo* in the image plane obtained by perspective projection. Note here that at least 4 coplanar points are requested to estimate the 8 parameters of the homography.

```
int npoints = 4;

std::vector< cv::Point3d > wX;  // 3D points in the
     world plane

std::vector< cv::Point2d > xw;  // Normalized
     coordinates in the object frame
std::vector< cv::Point2d > xo;  // Normalized
     coordinates in the image plane
```

For our simulation we then initialize the input data from a ground truth pose with the translation in *ctw_truth* and the rotation matrix in *cRw_truth*. For each point *wX*[i] we compute the perspective projection *xo*[i] = (xo, yo, 1). According to equation (27) we also set $\mathbf{x}_w = ({}^wX, {}^wY, 1)$ in *xw* vector.

```
// Ground truth pose used to generate the data
cv::Mat otw_truth = (cv::Mat_<double>(3,1) << -0.1,
     0.1, 1.2); // Translation vector
cv::Mat orw_truth = (cv::Mat_<double>(3,1) <<
     CV_PI/180*(5), CV_PI/180*(0), CV_PI/180*(45)); //
     Rotation vector
cv::Mat oRw_truth(3,3,cv::DataType<double>::type); //
     Rotation matrix
cv::Rodrigues(orw_truth, oRw_truth);
```

```
// Input data: 3D coordinates of at least 4 coplanar
    points
double L = 0.2;
wX.push_back( cv::Point3d(  -L, -L, 0) ); // wX_0 (-L,
    -L, 0)^T
wX.push_back( cv::Point3d( 2*L, -L, 0) ); // wX_1 ( L,
    -L, 0)^T
wX.push_back( cv::Point3d(   L,  L, 0) ); // wX_2 ( L,
    L, 0)^T
wX.push_back( cv::Point3d(  -L,  L, 0) ); // wX_3 (-L,
    L, 0)^T

// Input data: 2D coordinates of the points on the
    image plane
for(int i = 0; i < wX.size(); i++) {
  cv::Mat oX = oRw_truth*cv::Mat(wX[i]) + otw_truth; //
    Update oX, oY, oZ
  xo.push_back( cv::Point2d( oX.at<double>(0,
    0)/oX.at<double>(2, 0),
                             oX.at<double>(1,
    0)/oX.at<double>(2, 0) ) ); // xo = (oX/oZ, oY/oZ)

  xw.push_back( cv::Point2d( wX[i].x, wX[i].y ) ); //
    xw = (wX, wY)
}
```

From here we have initialized $\mathbf{x_0} = (x_0, y_0, 1)^T$ and $\mathbf{x}_w = ({}^wX, {}^wY, 1)^T$. We are now ready to call the function that does the pose estimation.

```
cv::Mat otw(3, 1, CV_64F); // Translation vector
cv::Mat oRw(3, 3, CV_64F); // Rotation matrix

pose_from_homography_dlt(xw, xo, otw, oRw);
```

Resulting pose estimation

If you run the previous code, it we produce the following result that shows that the estimated pose is equal to the ground truth one used to generate the input data:

```
otw (ground truth):
[-0.1;
 0.1;
 1.2]
otw (computed with homography DLT):
[-0.1;
 0.09999999999999999;
 1.2]
oRw (ground truth):
```

```
[0.7072945483755065, -0.7061704379962989,
      0.03252282795827704;
 0.7061704379962989, 0.7036809008245869,
      -0.07846338199958876;
 0.03252282795827704, 0.07846338199958876,
      0.9963863524490802]
oRw (computed with homography DLT):
[0.7072945483755065, -0.7061704379962993,
      0.03252282795827707;
 0.7061704379962989, 0.7036809008245873,
      -0.07846338199958841;
 0.03252282795827677, 0.07846338199958854,
      0.996386352449081]
```