****

IPMV-Experiment-5

Lab Camera geometry with eigen

课程名称： 图像处理与机器视觉

实验地点： 嘉定校区智信馆131

指导教师： Lei Jiang， Rui FAN

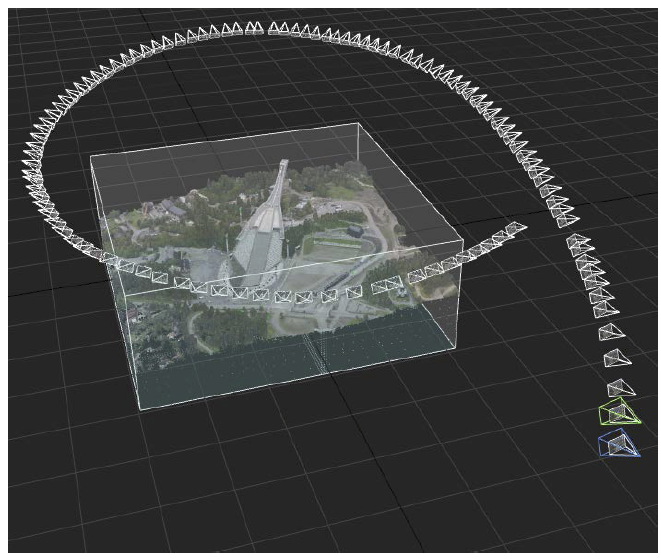
姓 名： 姚天亮

学 号： 2150248

**Task**

Background

In this experiment, we need to project world points into the images.

**

We are going to use Holmenkollen dataset.

110 images taken from helicopter.

For each image, consists:

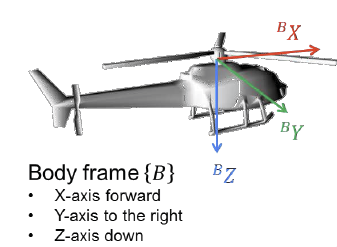
Intrinisc calibration

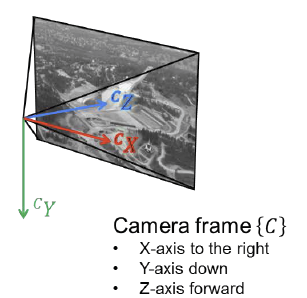
Helicopter pose in geographical coordinates

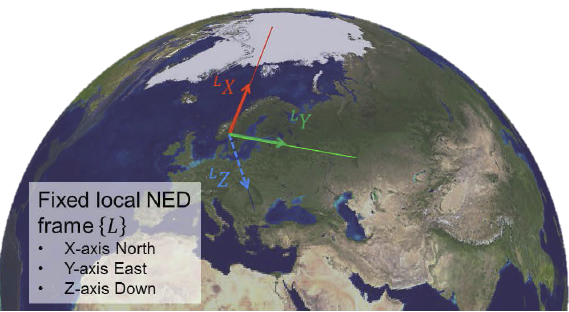
Camera pose relative to helicopter

The coordinate systems are shown as below:

helicopter







#include "camera\_pose\_solution.h"

#include "dataset.h"

#include "local\_coordinate\_system.h"

#include "viewer\_3d.h"

#include "opencv2/highgui.hpp"

#include "opencv2/imgproc.hpp"

CameraPoseSolution::CameraPoseSolution(const std::string& data\_path)

: data\_path\_{data\_path}

, window\_name\_{"World point in camera"}

{}

void CameraPoseSolution::run()

{

// Set up dataset.

Dataset dataset{data\_path\_};

// Define local coordinate system based on the position of a light pole.

const GeodeticPosition light\_pole\_position{59.963516, 10.667307, 321.0};

const LocalCoordinateSystem local\_system(light\_pole\_position);

// Construct viewers.

cv::namedWindow(window\_name\_);

Viewer3D viewer;

// Process each image in the dataset.

for (DataElement element{}; dataset >> element;)

{

// Compute the pose of the body in the local coordinate system.

const Sophus::SE3d pose\_local\_body = local\_system.toLocalPose(element.body\_position\_in\_geo,

element.body\_attitude\_in\_geo.toSO3());

// Add body coordinate axes to the 3D viewer.

viewer.addBodyAxes(pose\_local\_body, element.img\_num);

// Compute the pose of the camera relative to the body.

const Sophus::SE3d pose\_body\_camera(element.camera\_attitude\_in\_body.toSO3(),

element.camera\_position\_in\_body.toVector());

// Compute the pose of the camera relative to the local coordinate system.

const Sophus::SE3d pose\_local\_camera(pose\_local\_body \* pose\_body\_camera);

// Add camera coordinate axes to the 3D viewer.

viewer.addCameraAxes(pose\_local\_camera, element.img\_num);

// Construct a camera model based on the intrinsic calibration and camera pose.

const PerspectiveCameraModel camera\_model{element.intrinsics.toCalibrationMatrix(),

pose\_local\_camera,

element.intrinsics.toDistortionCoefficientVector()};

// Undistort image.

cv::Mat undistorted\_img = camera\_model.undistortImage(element.image);

// Project world point (the origin) into the image.

const Eigen::Vector2d pix\_pos = camera\_model.projectWorldPoint(Eigen::Vector3d::Zero());

// Draw a marker in the image at the projected position.

const Eigen::Vector2i pix\_pos\_int = (pix\_pos.array().round()).cast<int>();

cv::drawMarker(undistorted\_img, {pix\_pos\_int.x(), pix\_pos\_int.y()}, {0.,0.,255.}, cv::MARKER\_CROSS, 40, 3);

// Add the camera frustum with the image to the 3D viewer.

viewer.addCameraFrustum(camera\_model, undistorted\_img, element.img\_num);

// Show the image.

cv::imshow(window\_name\_, undistorted\_img);

// Update the windows.

viewer.spinOnce();

cv::waitKey(100);

}

// Remove image viewer.

cv::destroyWindow(window\_name\_);

// Run 3D viewer until stopped.

viewer.spin();

}

#include "perspective\_camera\_model.h"

#include "opencv2/core/eigen.hpp"

#include "opencv2/imgproc.hpp"

#include "opencv2/calib3d.hpp"

PerspectiveCameraModel::PerspectiveCameraModel(const Eigen::Matrix3d& K,

const Sophus::SE3d& pose\_world\_camera,

const Vector5d& distortion\_coeffs)

: K\_{K}

, pose\_world\_camera\_{pose\_world\_camera}

, distortion\_coeffs\_{distortion\_coeffs}

{

camera\_projection\_matrix\_ = computeCameraProjectionMatrix();

}

Sophus::SE3d PerspectiveCameraModel::getPose() const

{

return pose\_world\_camera\_;

}

Eigen::Matrix3d PerspectiveCameraModel::getCalibrationMatrix() const

{

return K\_;

}

PerspectiveCameraModel::Matrix34d PerspectiveCameraModel::getCameraProjectionMatrix() const

{

return camera\_projection\_matrix\_;

}

PerspectiveCameraModel::Matrix34d PerspectiveCameraModel::computeCameraProjectionMatrix()

{

return K\_ \* pose\_world\_camera\_.inverse().matrix3x4();

}

Eigen::Vector2d PerspectiveCameraModel::projectWorldPoint(Eigen::Vector3d world\_point) const

{

return (camera\_projection\_matrix\_ \* world\_point.homogeneous()).hnormalized();

}

Eigen::Matrix2Xd PerspectiveCameraModel::projectWorldPoints(Eigen::Matrix3Xd world\_points) const

{

return (camera\_projection\_matrix\_ \* world\_points.colwise().homogeneous()).colwise().hnormalized();

}

cv::Mat PerspectiveCameraModel::undistortImage(cv::Mat distorted\_image) const

{

// Convert to cv::Mats

cv::Mat K\_cv;

cv::eigen2cv(K\_, K\_cv);

cv::Mat dist\_coeffs\_cv;

cv::eigen2cv(distortion\_coeffs\_, dist\_coeffs\_cv);

// Undistort image.

cv::Mat undistorted\_image;

cv::undistort(distorted\_image, undistorted\_image, K\_cv, dist\_coeffs\_cv);

return undistorted\_image;

}

