同濟大學
TONGJI UNIVERSITY

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

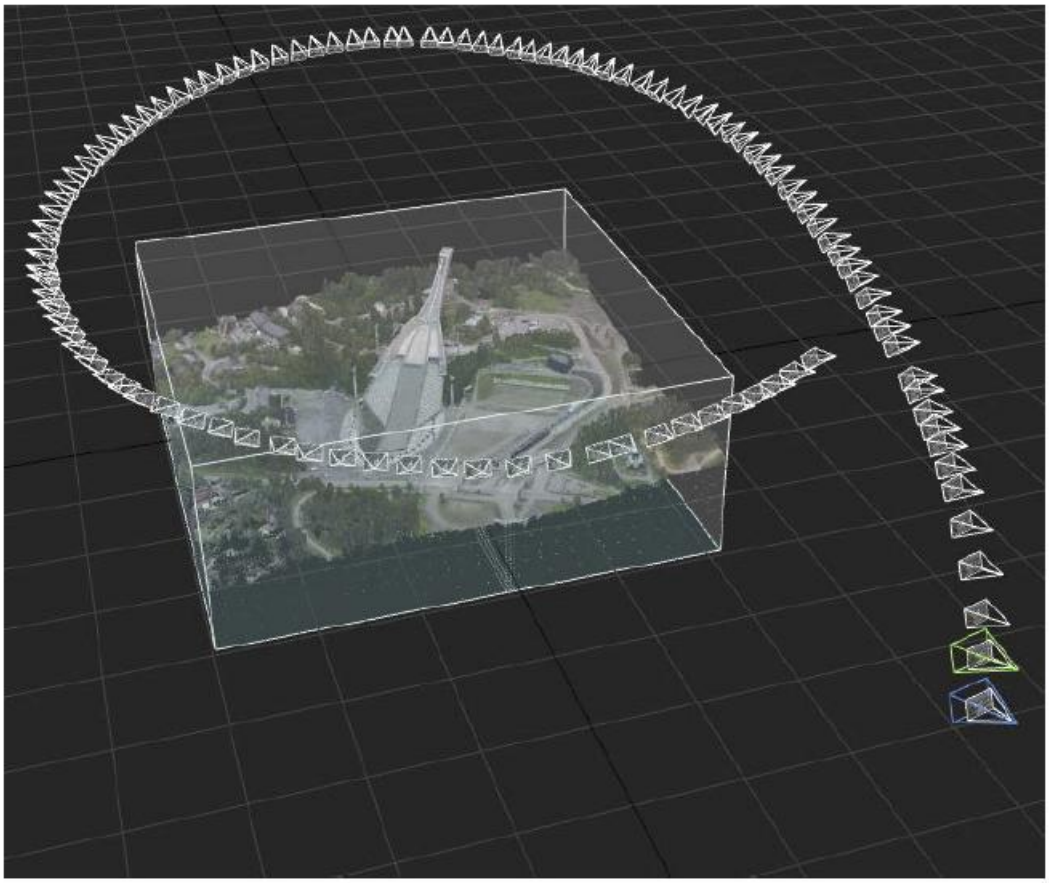
姓　　名：姚天亮

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:

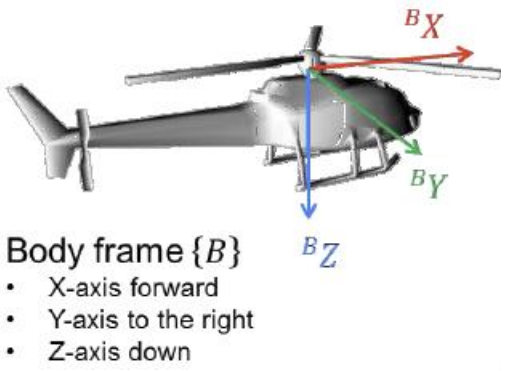
Intrinisc calibration

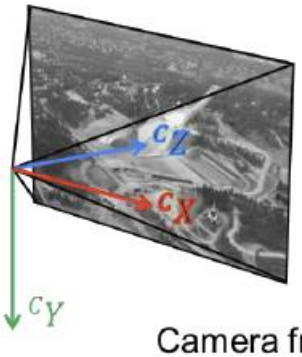
Helicopter pose in geographical coordinates

Camera pose relative to helicopter

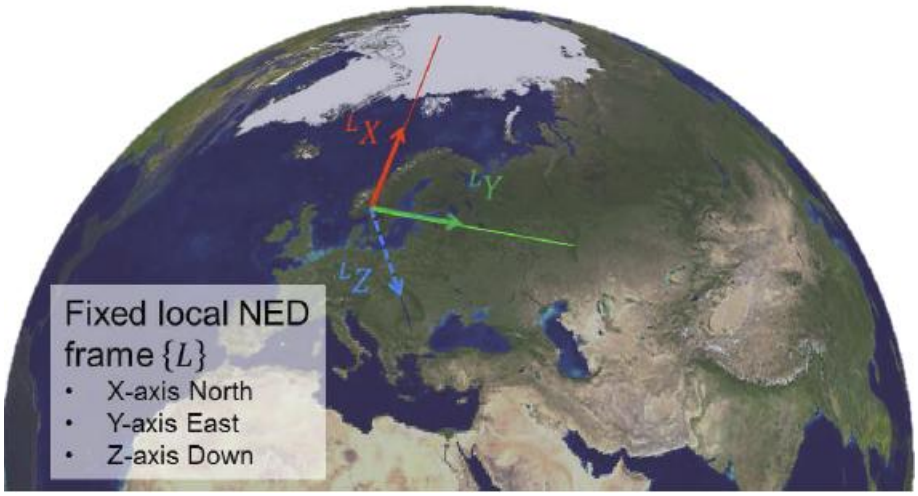
The coordinate systems are shown as below:

helicopter





Camera frame $\{C\}$
- X-axis to the right
- Y-axis down
- Z-axis forward




```
#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;
}
```