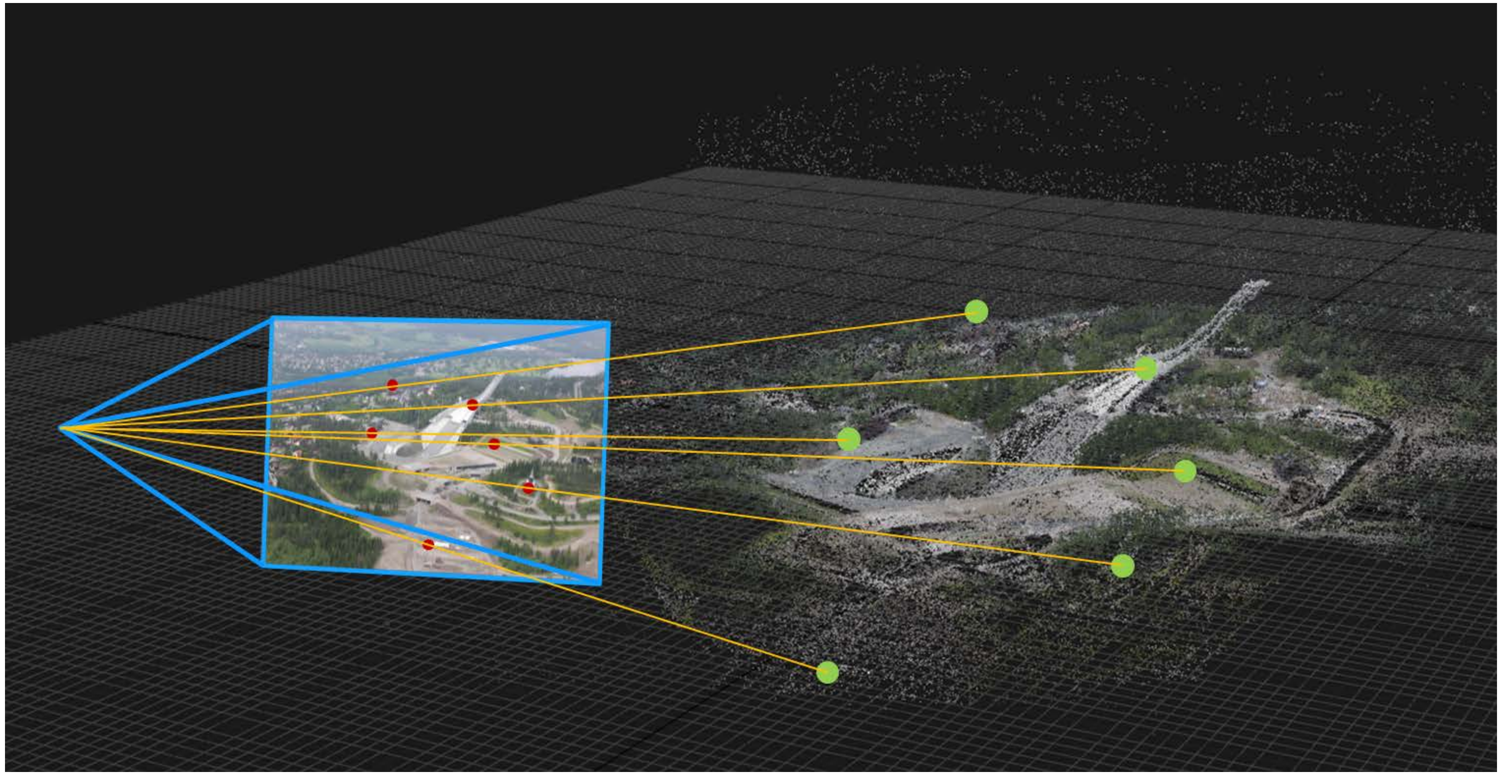
Lab 5 –
Pose estimation and Augmented Reality

Topics

- Camera calibration
- 3D-2D pose estimation
- AR and 3D visualization



Lab 5 in short

- Calibrate the camera
- Create a planar 3D world model with point descriptors
- Find 3D-2D correspondences
  between the world model and the current frame
- Estimate the camera pose from the 3D-2D correspondences
- Visualize the 3D world in the camera frame (AR)
- Visualize the camera and world model in 3D

Step 1: Camera calibration

- Use the OpenCV application [opencv_interactive-calibration] to calibrate the camera
  - Use tape to fasten the chessboard to the desk
  - Open the terminal and go to the project directory
  - Run opencv_interactive-calibration:

```
opencv_interactive-calibration -ci=0 -t=chessboard -sz=30 -w=8 -h=5 –pf=calibSettings.xml
```

  - Make sure to measure the chessboard from different orientations and positions
  - When you are happy with the calibration, store the results by pressing `s`
  - Quit by pressing `Esc`
- Set calibration parameters in `setupCameraModel()` in lab_5.cpp

Step 2: Implement HomographyPoseEstimator

```
// TODO 2-1: Compute M and M_bar.
// Compute the matrix M
// and extract M_bar (the two first columns of M).
Eigen::Matrix3d M;
Eigen::MatrixXd M_bar;
```

$$\boldsymbol{M} = \boldsymbol{K}^{-1}\boldsymbol{H}_W^C$$

$$\bar{\boldsymbol{M}} = \left[\boldsymbol{m}_1, \boldsymbol{m}_2\right]$$

Step 2: Implement HomographyPoseEstimator

```
// TODO 2-2: Compute Singular Value Decomposition.
// Perform SVD on M_bar.
auto svd = M_bar.jacobiSvd(Eigen::ComputeThinU | Eigen::ComputeThinV);

// TODO 2-3: Compute R_bar.
// Compute R_bar (the two first columns of R)
// from the result of the SVD.
```

$$\bar{\boldsymbol{R}} = \boldsymbol{U}\boldsymbol{V}^T$$

$$\bar{\boldsymbol{R}} = \left[r_1, r_2\right]$$

Step 2: Implement HomographyPoseEstimator

```
// TODO 2-4: Construct R.
// Construct R by inserting R_bar and
// computing the third column of R from the two first.
// Remember to check det(R)!
Eigen::Matrix3d R = Eigen::Matrix3d::Identity();
```

Step 2: Implement HomographyPoseEstimator

```
// TODO 2-5: Compute the scale.
// Compute the scale factor lambda.
double lambda = 0;
```

$$\lambda = \frac{\operatorname{trace}\left(\bar{\boldsymbol{R}}^T \bar{\boldsymbol{M}}\right)}{\operatorname{trace}\left(\bar{\boldsymbol{M}}^T \bar{\boldsymbol{M}}\right)} = \frac{\sum_{i=1}^{3} \sum_{j=1}^{2} R_{ij} M_{ij}}{\sum_{i=1}^{3} \sum_{j=1}^{2} M_{ij}^2}$$

Step 2: Implement HomographyPoseEstimator

```
// TODO 2-6: Compute t.
// Extract the translation t.
Eigen::Vector3d t = M.col(2) * lambda;
```

$$\left[\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{t}\right] = \pm\lambda \boldsymbol{M}$$

Step 2: Implement HomographyPoseEstimator

```
// TODO 2-7: Find correct solution.
// Check that this is the correct solution
// by testing the last element of t.
```

$$\left[\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{t}\right] = \pm\lambda \boldsymbol{M}$$

Step 3: Try the other pose estimators

- `PnPPoseEstimator pose_estimator(camera_model.K);`

- ```
  auto init_estimator = std::make_shared<HomographyPoseEstimator>(camera_model.K);
  GtsamPoseEstimator pose_estimator(init_estimator, camera_model.K);
  ```
  ```

Step 4: Play!

- Try other feature detectors and descriptors
  - What about **cv::xfeatures2d::AffineFeature2D**?

- Add cool new 3D elements to the AR viewer

- Visualize the 3D track over the last *n* frame