# VEDLEGG B: BRUKERMANUAL

## Visual Odometry

VisualOdometry mappen inneholder tre mapper. Bildetaker, Stereo_Kamera_Kalibrering og Visual_Odometry_Algoritme.

**Bildetaker**
Koden i mappen er brukt for å lage bildeseriene til de ulike testmiljøene i oppgaven. Mens koden kjøres vises bildene som blir tatt på skjermen og lagrer de i stereoLeft og stereoRight mappene. Koden tar ca 10 bilder per sekund og kan endres med denne funksjonen:

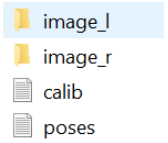
```
39      #Denne bestemmer hvor mange FPS bildene blir tatt i ved 0.0025 tilsvarer ca 10 FPS
40      time.sleep(0.025)
```

**Stereo_Kamera_Kalibrering**
I mappen vil man finne tre koder, to mapper med bilder og en .xml fil.
**calibration_images.py** brukes til å ta enkeltbilder med to kameraer og lagrer bildene i bildemappene. **stereovision_calibration.py** prosesserer bildene som lager en xml fil med verdier fra kalibreringen. I tillegg blir en print() funksjon brukt til å skrive ut projection matrix til kameraene. Dette er kalibreringen som benyttes i oppgaven:

```
projL [[ 1.61261182e+03  0.00000000e+00 -7.52986923e+02  0.00000000e+00]
 [ 0.00000000e+00  1.61261182e+03  4.00863571e+02  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
projR [[ 1.61261182e+03  0.00000000e+00 -7.52986923e+02 -3.41461230e+05]
 [ 0.00000000e+00  1.61261182e+03  4.00863571e+02  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
```

stereovision.py leser av xml filen og for å implementere kalibreringen til kameraene for å vise effekten.

**Visual_Odometry_Algoritme**
Stereo_Visual_Odometry.py er posisjoneringsalgoritmen og programmet som har gitt resultatene i oppgaven. For å kjøre koden trengs en mappe med denne strukturen:




I mappene ligger bilder fra høyre og venstre kamera. calib.txt filen inneholder projection matrix. poses.txt er en fil som inneholder sann posisjon, men denne filen inneholder kun et og samme punkt ettersom denne funksjonen ikke blir benyttet i oppgaven. I Aula_TEST mappen inneholder også kalibreringsfilene til både kalibreringskoden (calib_nermeste.txt) og kalibrering til en bil som benytter visuell posisjonering(calib_orginal).

Funksjonene brukt i Stereo_Visual_Odometry.py

**Valg av datasett**

```
385    def main():
386        data_dir = 'Aula_TEST'  # try
387        vo = VisualOdometry(data_dir)
```

Dette er linjen hvor mappen med bilder og kalibrering blir valgt. For å bytte mappe så byttes ut Aula_TEST med navnet på mappen som ønskes å bli kjørt.

**Initial**

```
13      def __init__(self, data_dir):
14          self.K_l, self.P_l, self.K_r, self.P_r = self._load_calib(data_dir + '/calib.txt')
15          self.gt_poses = self._load_poses(data_dir + '/poses.txt')
16          self.images_l = self._load_images(data_dir + '/image_l')
17          self.images_r = self._load_images(data_dir + '/image_r')
```

I __init__ blir de valgte filene lest av og implementert i koden. Disse filene må være i mappen som er valgt i main().

**FAST**

Istedenfor at FAST gjennomføres på hele bildet, så blir bildet delt opp i bokser på 10x20 piksler. Deretter blir alle features overført på bildet og 100 beste features blir brukt videre.

```
365        # Get teh tiled keypoints
366        kp1_l = self.get_tiled_keypoints(img1_l, 10, 20)
```

For å bestemme hvor store bokser bildet skal fordeles i, endre på verdiene i funksjonen(img1_l, y-retning, x-retning).

```
177                if len(keypoints) > 100: ←
178                    keypoints = sorted(keypoints, key=lambda x: -x.response)
179                    return keypoints[:100] ←
```

For å bestemme antall beste features som skal gå videre til optical flow, endre 100 til en ønsket verdi.

**Plotting resultater**

Aula_TEST
Bibliotek_TEST
Kai_TEST

Etter programmet har prosessert all data så vil en HTML fil dannes som vil vise en grafisk fremstilling av resultatet. Disse resultatene kan åpnes med en nettleser.

# Visual SLAM

## Kalibrering

For kalibrering, se filstruktur vedlagt i GitHub. README fil i kalibreringsmappen forklarer dette på en god måte. Se Vedlegg C**.**

## Posisjoneringsalgoritmen

### Valg av datasett

```
%baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_
%dataFolder      = fullfile(tempdir, 'tum_rgbd_dataset', filesep);
dataSet          = fullfile(tempdir, 'Vanntank_24_fps', filesep);
dataSet2         = fullfile(tempdir, 'Bibliotek_24_fps', filesep);
dataSet3         = fullfile(tempdir, 'Aula_24_fps', filesep);
options          = weboptions(Timeout=Inf);
```

Posisjoneringsalgoritmen starter med å hente datasettene fra *Temp* som finnes i %appdata%. Disse datasettene er mapper med bilder laget fra video med 24 fps.

### Lagring av datasett

```
%imageFolder   = [dataFolder,'rgbd_dataset_freiburg3_long_office_household/rgb/'];
imageFolder1  = dataSet;
imageFolder2  = dataSet2;
imageFolder3  = dataSet3;
imds          = imageDatastore(imageFolder1);

```

Disse datasettene blir så tildelt variabel for å benyttes i koden. imageDatastore leser mappene med bilder og lagrer disse i variabelen imds.

### Inspiser første bilde

```

% Inspect the first image
currFrameIdx = 1;
currI = readimage(imds, currFrameIdx);
himage = imshow(currI);

```

Deretter fortsetter koden å lese filen. currFrameIdx bestemmer hvilket bilde koden skal starte å lese fra. currFrameIdx = 1; betyr at koden skal starte på bilde nr 1.

### Parametere for kalibrering

```
35
36  % Create a cameraIntrinsics object to store the camera intrinsic parameters.
37  % The intrinsics for the dataset can be found at the following page:
38  % https://vision.in.tum.de/data/datasets/rgbd-dataset/file_formats
39  % Note that the images in the dataset are already undistorted, hence there
40  % is no need to specify the distortion coefficients.
41  focalLength    = [535.4, 539.2];    % in units of pixels
42  principalPoint = [320.1, 247.6];    % in units of pixels
43  imageSize      = size(currI,[1 2]);  % in units of pixels
44  intrinsics     = cameraIntrinsics(focalLength, principalPoint, imageSize);
45  load('Chess_kalibrering.mat')
46  intrinsics     = calibrationSession.CameraParameters.Intrinsics;
47
```

I focalLength og principalPoint lagres verdiene hentet fra kalibreringssettet. [535.4, 539.2] er pikselverdier i [x,y] format. Det samme gjelder for principalPoint. Disse verdiene blir lagret i intrinsics, som tar høyde for forvrengning i hvert bilde i datasettet.

### Antall kjennetegn i bildet

```
48  % Detect and extract ORB features
49  scaleFactor = 1.2;
50  numLevels   = 8; %8
51  numPoints   = 3000; %1000
```

numPoints kan justeres for å øke antall feature points algoritmen leter etter i hvert bilde. Desto flere numPoints, jo tregere kompilerer algoritmen, men dette til kostnad for nøyaktigheten. Ved mye støy i bilde kan denne justeres ned for å senke antall features

### Øke hastighet

```
266     % localKeyFrameIds:   ViewId of the connected key frames of the current frame
267     numSkipFrames     = 20;
268     numPointsKeyFrame = 80;
```

numSkipFrames kan økes for gode datasett. Dette øker hastigheten på kompileringen men reduserer igjen nøyaktigheten.
numPointsKeyFrame definerer hvor mange features et bilde må inneholde for å bli et key frame. Ved å øke denne parameteren, øker nøyaktigheten til systemet, dersom datasettet er godt.

### Ground truth

```
369 % Load ground truth
370         %gTruthData = load("orbslamGroundTruth.mat");
371         %gTruth     = gTruthData.gTruth;
372
373         % Plot the actual camera trajectory
374         %plotActualTrajectory(mapPlot, gTruth(addedFramesIdx), optimizedPoses);
375
376         % Show legend
377         %showLegend(mapPlot);
378
```

Ground truth er den faktiske posisjonen systemet har hatt i gjennomføringen. Dette datasettet er tatt ved bruk av et akselerometer, noe oppgaven ikke hadde tilgjengelig. Dette er et godt sammenligningsgrunnlag i stedet for GPS data for å fortelle om faktisk posisjon.

**Kalibrering**

```
385  % In this example, the images are already undistorted. In a general
386  % workflow, uncomment the following code to undistort the images.
387
388   if nargin > 4
389       intrinsics = varargin{1};
390   end
391  Irgb  = undistortImage(Irgb, intrinsics);
392
393  % Detect ORB features
394  Igray  = im2gray(Irgb);
395
```

Denne delen av koden kalibrerer alle bildene gjennom intrinsics som definert tidligere.

**Øke maks avstand eller konfidensgrad**

```
405  function [H, score, inliersIndex] = helperComputeHomography(matchedPoints1, matchedPoints2)
406
407  [H, inliersLogicalIndex] = estgeotform2d( ...
408      matchedPoints1, matchedPoints2, "projective", ...
409      MaxNumTrials=1e3, MaxDistance=4, Confidence=90);
```

MaxDistance kan økes eller reduseres for å endre distansen algoritmen leter etter features. Høyere distanse fjerner støy i nærheten av kamera.
Confidence er konfidensgraden til hver feature. Her er graden satt til 90%. Alle features med lavere konfidens, filtreres vekk.