Motion Estimation Algorithms

General Figures

- Image Dimensions (max.): 112V x 112H
- Bit Depth: 14 bits (stored in uint16) → change to 8 !!
- Time for acquiring one whole image: 46ms (≈ max. 21 images/s)
  (@ CPU clock 24MHz with clock dividers 4 for ADC and Timers)
  (can not arbitrarily be reduced without loss in quality because of camera)
- Time for storing one image on FRAM: 287ms (measured)
  (can be reduced by factor ½ if changing bit depth to 8!)
- **Camera Specifications:**
  - focal length f: 4mm
  - difference between two pixels: 25µm
  - size of focal plane s: 112 * 25µm = 2.8mm
  - field of view (FOV): 2*asin(s/2/f) ≈ 41°
  - size of image plane in distance d: s/f * d = 1.26 m for d = 1.8m
  - pixel size in distance d: 11.2mm for d = 1.8m
- **Example:** @ 20 fps ⇒ 1m/s → 50 mm = 4.5 pixels displacement per frame
  @ 20 fps ⇒ 1.4m/s → 70 mm = 6.25 pixels displ. per frame
  ⇒ We need to recover displacements of ±10 pixels for normal walking speed!

Differential Methods

- **Lucas-Kanade Algorithm**
  - Only for small motions, such that differential equation holds (often less than the pixel spacing)
    - pyramidal approach
      - small obsticles get lost
      - poor estimates propagate through all levels
  - can only recover displacements on edges / if enough brightness structure is available
  - MATLAB: performs well for very small displacements (~ 1 pixel)
  - high computational effort

- **Kanade–Lucas–Tomasi feature tracker**
  - same motion estimation as Lucas-Kanade-Method
  - only tracking features that are suitable for the tracking algorithm. The proposed features would be selected if both the eigenvalues of the gradient matrix were larger than some threshold.
  - same Problems as Lucas-Kanade algorithm

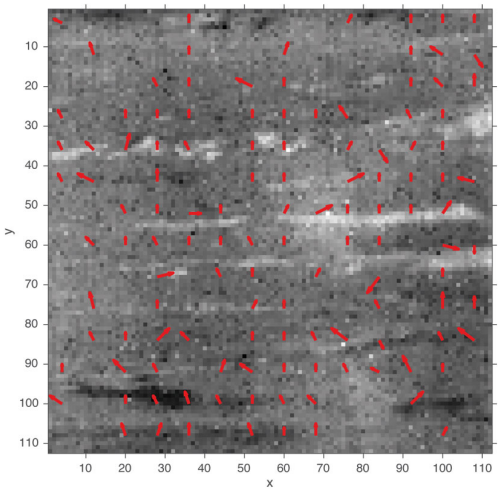
Frequency-based Methods

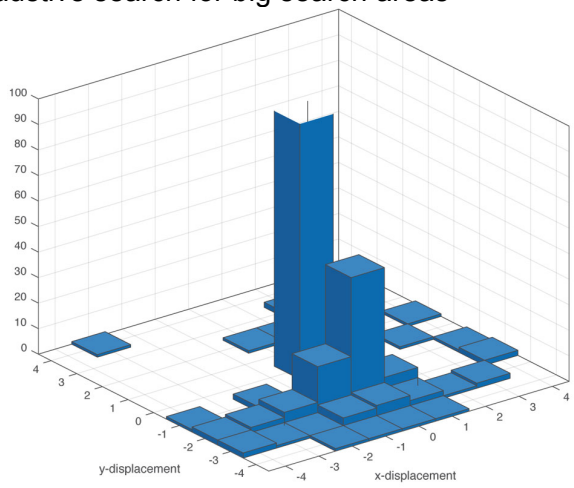
Idea: Transform image sequence from time-domain into frequency domain

- Only useful if „large“ sequence of images available
- high computational effort

Correlation-based Methods

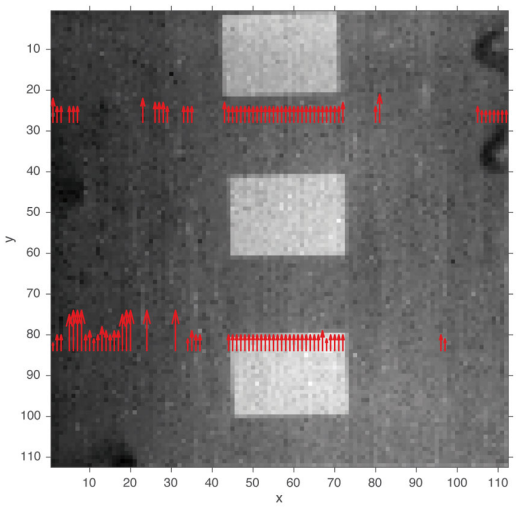
- **Block Correlation Algorithm (2D)**
  - MATLAB: blocks of 8x8 pixels, +- 4 pixels displacement search area
    - Number of Cost-Function evaluations:
      - numblocks_x * numblocks_y * (2*searcharea_x+1) * (2*searcharea_y+1) * blocksize_x * blocksize_y
      - = numpixels_x * numpixels_y * (2*searcharea_x+1) * (2*searcharea_y+1)
      - in this case: 1'016'064
    - performs well if enough brightness structure is available
    - estimates get arbitrary if too less structure
    - bias on (0, 0) displacement because of noise pattern from camera
    - relatively high computational effort (esp. for big search areas)
  - TODO:
    - introduce quality figure for each estimate to detect poor estimates
    - how to deal with mask/noise pattern?
    - implement other than exhaustive search for big search areas

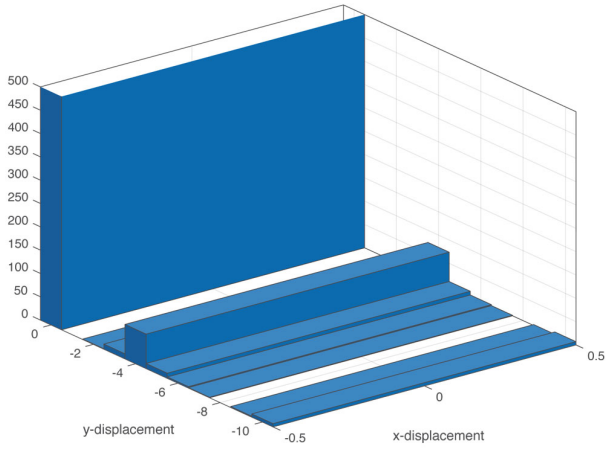




- **Simplified Block Correlation Algorithm, evaluating only certain points**
  - Idea: same as block correlation algorithm, but the cost function is only evaluated at some predefined points within the block
  - MATLAB:
    - Number of cost-function evaluations:
      - numblocks_x * numblocks_y * (2*searcharea_x+1) * (2*searcharea_y+1) * (blocksize_x + blocksize_y)
      - in this case: 254'016
    - Performance is significantly worse than normal Block Correlation Algorithm
    - less computational effort

- **1D Correlation Algorithm**
  - MATLAB: block size: 1x56 pixels, +- 10 pixel search area (in y-direction)
    - Number of cost-function evaluations:
      - numblocks_x * numblocks_y * (2*searcharea_x+1) * (2*searcharea_y+1) * blocksize_x * blocksize_y
      - = numpixels_x * numpixels_y * (2*searcharea_x+1) * (2*searcharea_y+1)
      - in this case: 263'424
    - computational effort is not too big, and can even be reduced if one considers only single columns
    - performs well, also for big displacements up to 10 pixels
    - estimates get arbitrary if too less structure
    - bias on (0, 0) displacement because of noise pattern from camera
    - If displacement is not exactly along the y-axis, the estimates may get arbitrarily bad (depending on the brightness structure)
  - TODO:
    - introduce quality figure for each estimate to detect poor estimates
    - how to deal with mask/noise pattern?
    - implement other than exhaustive search for big search areas





- **2x1D Correlation Algorithms using only certain rows/columns**
  - Idea: Only evaluate certain rows and columns and estimate the optical flow at the intersection points by evaluating horizontal and vertical 1D-displacement separately.
  - MATLAB: block size 8x8, search area +- 4 pixels
    - Number of cost-function evaluations:
      - numblocks_x * numblocks_y * { (2*searcharea_x+1) * blocksize_x + (2*searcharea_y+1) * blocksize_y }
      - in this case: 28'224
    - use less memory (not whole image has to be acquired/stored)
    - if displacements are not purely horizontal or vertical, small objects can easily be lost and estimate can get very bad!