than 1000 images, resulting in short prediction times, time complexity can be reduced by storing and searching the training examples in an efficient manner, for example, with tree structures [6].

3.3.4 Filtering

Computer vision-based estimations are often noisy or ambiguous. Texton histograms obtained during flight will not perfectly match the ones in the training dataset: blur, lighting settings, viewing angles, and, other variables change the shape of the histograms.

To filter out outliers and smooth estimates, a popular filter choice is the Kalman filter. However, the Kalman filter is not able to represent multimodal probability distributions [17]. This makes it rather unsuitable for the presented *global* localization approach. The "naive" k-NN regression calculates the mean of the k outputs and forwards this value to the Kalman Filter. However, if the output values are distant, averaging them yields a value in the center between them, which is not likely to be the correct position (Figure 3.7). This approach can lead to biased predictions, especially, if the model outputs belong to distant locations due to similar texton distributions at these positions.

We decided to use a more sophisticated method to capture *multimodal distributions*. Given an adequate measurement model, a general Bayesian filter can simultaneously maintain multiple possible locations and resolve the ambiguity as soon as one location can be favored (Figure 3.8). In this case, the predictions of the k neighbors can be directly fed into the filter without averaging them first. The filter is able to smooth the estimations, handle uncertainty, and simultaneously keep track of several position estimates. However, a general Bayesian filter is computationally complex. Therefore, a variant based on random sampling was used: the particle filter. While its computational complexity is still high compared to a Kalman filter, one can modify the amount of particles to trade off speed and accuracy and adapt the computational payload to the used processor.

The weighted particles are a discrete approximation of the probability density function (*pdf*) of the state vector (x, y-position of the MAV). Estimating the filtered position of the MAV can be described as $p(X_t \mid Z_t)$, where X_t is the state vector at time t and $Z_t = \mathbf{z}_1, ..., \mathbf{z}_t$ are all outputs of the k-NN

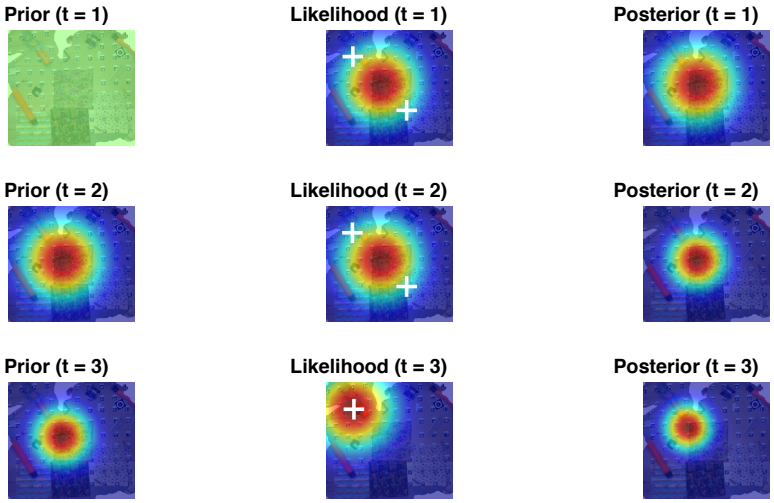

Figure 3.7: The figure illustrates the measurement model of a Kalman filter. The colors represent the probability of an x, y-position (red: high probability; blue: low probability). In timestep $t = 1$, the filter is initialized with an uniformed prior and each position has equal probability. To incorporate measurement error, the likelihood (measurement model) is calculated using a Gaussian distribution that is centered around the mean of the $k = 2$ predictions (white crosses) from the k-NN algorithm. The posterior results from the multiplication of the prior with the likelihood and indicates the position estimates after one timestep. In the next timestep, the previous posterior becomes the new prior. The filter receives distant measurements in time steps $t = 1$ and $t = 2$ that are averaged to receive a position in the middle. In time step $t = 3$, the ambiguity is resolved but the filter only slowly adapts to the new position.

algorithm up to time t, with each $\mathbf{z}_i$ representing the k x, y-outputs of the algorithm at time i.

The used particle filter is initialized with $M = 100$ particles at random x, y-positions. To incorporate the measurement noise for each of the k estimates from the k-NN algorithm, we developed a two-dimensional Gaussian Mixture Model (GMM) as measurement model. The parameters of the GMM are obtained by comparing the ground truth positions (random variable T) from the motion tracking system to the estimates from the k-NN algorithm (random variable P_j) in a recorded dataset. The GMM is parameterized by the variances $\Sigma^{[j]}, j \in \{1, \ldots, k\}$ that are dependent on the rank j of the

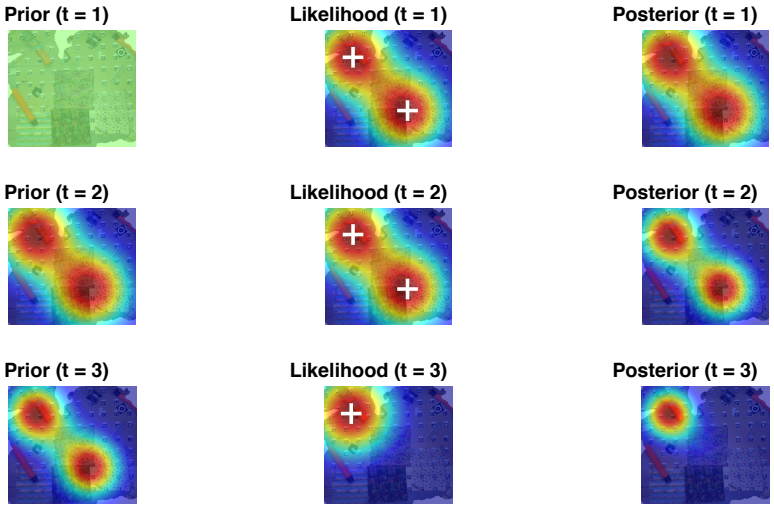

Figure 3.8: Three time steps of a Bayesian filter. The colors represent the probability of an x, y-position (red: high probability; blue: low probability). In contrast to the Kalman filter, the likelihood (measurement model) is calculated using a *mixture* of Gaussian distributions centered around the outputs the k-NN algorithm (white crosses). The filter can immediately resolve the ambiguity in time step 3 and the posterior gets updated accordingly.

prediction of the k-NN algorithm (for example, $j = 2$ is the second nearest neighbor). The variance matrix $\Sigma^{[j]}$ specifies the variances of the deviations in x-direction and y-direction and the correlation ρ between the deviations. The values for $\Sigma^{[j]}$ were determined by calculating the variance-covariance matrix for the difference between the ground truth T and the predictions P_j of the k-NN algorithm: $\Sigma^{[j]} := \mathrm{Var}(T - P_j)$.

In contrast to the measurement model, the used *motion model* is simple. It is solely based on Gaussian process noise and does not consider velocity estimates, headings, or control inputs. Its mean and variance are dependent on the expected velocity of the MAV. We used the forward difference $T_t - T_{t-1}$ to estimate the average movement and its variance-covariance matrix Σ_{process} between timesteps t and $t - 1$. While the employed motion model is simple, the developed software provides functionality for including an odometry-based motion model based on optical flow.

The algorithm of the developed particle filter is presented in the pseudo

code in Algorithm 2. In the pseudo code, $\mathcal{X}$ is the list of particles, f the two-dimensional Gaussian probability density function, $z_t^{[i]}$ the ith neighbor from the kNN prediction, $x_t^{[m]}$ the mth particle at time t, and $w_t^{[m]}$ its corresponding weight.

Algorithm 2 Particle filter update

procedure PARTICLE_FILTER($\mathcal{X}_{t-1}, z_t$)
    ▷ Initialize particle list
    $\mathcal{X}_{\text{temp}} := \varnothing$
    for $m = 1$ to M **do**
        ▷ Add random process noise (motion model)
        $x_t^{[m]} \leftarrow x_t^{[m]} + \mathcal{N}(0, \Sigma_{\text{process}})$
        ▷ Iterate over predictions from k-NN (measurement model)
        $w \leftarrow 0$
        for $i = 1$ to k **do**
            ▷ Gaussian Mixture Model
            $w \leftarrow w + f(z_t^{[i]}; x_t^{[m]}, \Sigma_{\text{measurement}}^{[i]})$
        $\mathcal{X}_{\text{temp}} := \mathcal{X}_{\text{temp}} \cup (x_t^{[m]}, w)$
    ▷ Importance resampling
    $\mathcal{X}_t \leftarrow$ RESAMPLING_WHEEL($\mathcal{X}_{\text{temp}}$)
    return $\mathcal{X}_t$

The "resampling wheel" [42] (Algorithm 3) performs the importance resampling step. Its underlying idea is that the particles are arranged in a "wheel," with each particle occupying a slice that corresponds to its weight. The particles are then resampled with a probability proportional to the area of the slices. This step ensures that particles with a low weight are removed and replaced with well-performing ones. Otherwise, the algorithm might "collapse" when all but one particle have a low weight.

With the GMM, the information of all k neighbors can be used, yielding a possibly multimodal distribution. While a multimodal distribution allows for keeping track of several possible positions, certain subsystems—for example a control loop—often need *one* point estimate. Using a weighted average of the particles would again introduce the problem that it could fall into a low density region (an unlikely position). Instead, we used a maximum a posteriori (MAP) estimate, as described by Driessen et al. [19]. This approach is a discrete approximation of the true MAP [19]. It uses

Algorithm 3 Resampling wheel

```
1: procedure RESAMPLING_WHEEL(X_temp)
2:     ▷ Initialize particle list
3:     X_t ← ∅
4:     ▷ Sample random index from the number of particles
5:     sample i ~ M · U(0, 1)
6:     β ← 0
7:     for m = 1 to M do
8:         β ← β + U(0, 1) · 2 · max(w_t)
9:         while β > w_t^[i] do
10:            β ← β − w_t^[i]
11:            i ← (i + 1) mod M
12:        X_t ← X_t ∪ X_temp^[i]
13:    return X_t
```

the following formula to obtain the MAP estimate X_t^{MAP}—the "final" x, y-position:

$$X_t^{\text{MAP}} = \underset{x_t^{[i]} \in \{i=1,\ldots,M\}}{\arg\max} \sum_{j=1}^{M} f(x_t^{[i]}; x_{t-1}^{[j]}, \Sigma_{\text{process}}) w_{t-1}^{[j]} \tag{3.2}$$

Therefore, the final position estimate is equal to the position of one of the particles.

The estimation of *uncertainty* is a core part of the developed approach, due to its importance for safety and accuracy. Therefore, uncertainty was modeled using the spread of the particles—as expressed by their variance in x-direction and y-direction. Initially, we planned to include the distance between the current histogram obtained from the camera image and each of the k neighbors from the training set as confidence value. One could thus reduce the measurement noise if a high similarity between the current histogram and a training histogram is achieved. While we found no correlation between these variables, we still provide the functionality for incorporating the distance in the developed software. We also tried to use the amount of detected keypoints (K) as a confidence value for the quality of the sample in the training set if the homography-based approach is used for labeling. Again, no linear relationship between K and the error in x-direction (X) and the error in y-direction (Y) could be found.

3.4 Pillar III: Map evaluation

3.4.1 Evaluation Scheme

The performance of the developed method depends on the environment: a texture-rich environment without repeating patterns will be better suited than a texture-poor environment. Ideally, one would like to know if the algorithm will work in a given environment. Therefore, we propose an evaluation scheme that can compare different environments and areas within an environment. This scheme assigns a global fitness value to a "map"—expressed as dataset $\mathcal{D}$ consisting of N texton histograms h_i and corresponding x, y-coordinates $\text{pos}_i = (x_i, y_i)$. The fitness value is proportional to the accuracy that can be expected when using this dataset as training set for the developed localization algorithm. The scheme allows for inspecting the dataset and detecting regions within the map that are responsible for the overall fitness value.

The idea behind the global loss function L is that histograms h_i and h_j in closeby areas should be similar and the similarity should decrease with increasing distance of the corresponding x, y-coordinates pos_i and pos_j. Therefore, the approach is based on the difference between *actual* and *ideal* texton histogram similarities in a dataset. The ideal texton similarity distribution is modeled as a two-dimensional Gaussian distribution around each x, y-position in the dataset (Figure 3.9). Using this idea, a histogram is compared to all others by comparing expected similarities to actual similarities. This results in a loss value per sample of the dataset. Applying the algorithm to each sample in the dataset yields the global loss of a dataset. A visualization of the global loss is illustrated in Figure 3.10.

The method uses the cosine similarity (CS) to compare histograms:

$$CS(h_i, h_j) = \frac{h_i^T h_j}{||h_i||\,||h_j||} \tag{3.3}$$

The cosine similarity has the convenient property that its values are bounded between -1 and 1. In the present case, since the elements of the histograms are non-negative, it is even bounded between 0 and 1. Let the function f describe the non-normalized one-dimensional Gaussian probability density function:

$$f(x; \mu, \sigma) = e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.4}$$

Figure 3.9: *Left:* Actual similarity between histogram h_i (pos_i: white cross) and all other histograms; the heatmap shows low similarity in blue and high similarity in red. For the visualization, the actual similarities were smoothed with a Gaussian filter. *Middle:* Ideal histogram similarity distribution for the given position pos_i. Histograms h_j taken at closeby positions should have a high similarity to h_i. The farther away the position pos_j, the lower the similarity between h_i and h_j should be. *Right:* The difference between the actual and the ideal similarity shows regions that do not follow the ideal similarity distribution for histogram h_i (high loss: red; low loss: blue)

Since we assume that the ideal similarity in x-position is independent of the y-position, the ideal two-dimensional similarity function $d_e(\text{pos}_i, \text{pos}_j; \Sigma)$ can be modeled as the product of the respective one-dimensional function f:

$$d_e(\text{pos}_i, \text{pos}_j; \Sigma) = f(x_i; x_j, \sigma_x) \cdot f(y_i; y_j, \sigma_y) \tag{3.5}$$

This function is also bounded between 0 and 1, which makes the functions d_e and CS—ideal similarity and actual similarity—easily comparable. In summary, we propose the following global loss function (L) for evaluating a given dataset ($\mathcal{D}$):

$$L(\mathcal{D}) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} CS(h_i, h_j) - f(x_i; x_j, \sigma_x) \cdot f(y_i; y_j, \sigma_y) \tag{3.6}$$

The simple difference—in contrast to least absolute deviations or least square errors—ensures that similarities that are *less* similar than the ideal similarity *reduce* the loss. Therefore, a high variation in texture is always seen as "positive". The variances σ_x and σ_y specify the dimension of the region, where similar histograms are desired. The lower their value, the more focused the ideal similarity will be, requiring a high texture variety for getting a low loss value. A high value might overestimate the suitability of a dataset. While the approach is relatively robust to the choice of the parameter values, we still need to find a heuristic for suitable values.

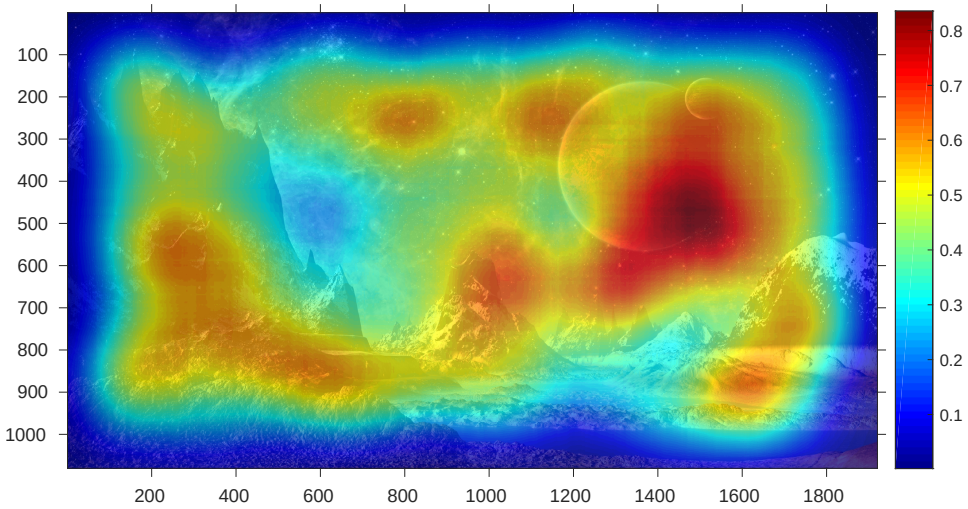

Figure 3.10: The figure shows the loss of a map: the regions that did not follow the ideal similarity pattern are displayed in red. For the visualization, the loss values per sample in the dataset were smoothed with a Gaussian filter. This assigns a loss value to each x, y-position of the map. The synthetic data generation tool was used for generating the underlying dataset (Section 3.4.2).

3.4.2 Synthetic Data Generation

To compare environments before actually flying in them, a software tool was developed that creates synthetic images to simulate those taken during an actual flight. The tool generates the patches based on perspective transformations of an image. Examples of generated images are displayed in Figure 3.11.

The application allows for comparing and predicting the performance of different "maps" as specified by an image. The software is written in C++ and OpenCV 3.0.0. The algorithm simulates a simple camera model that moves above the image (Figure 3.12). It generates a specified amount of image patches using random values—sampled from uniform and normal probability distributions—for various parameters:

- rotational angles: roll α, pitch β, yaw γ
- translational shifts: dx, dy, dz

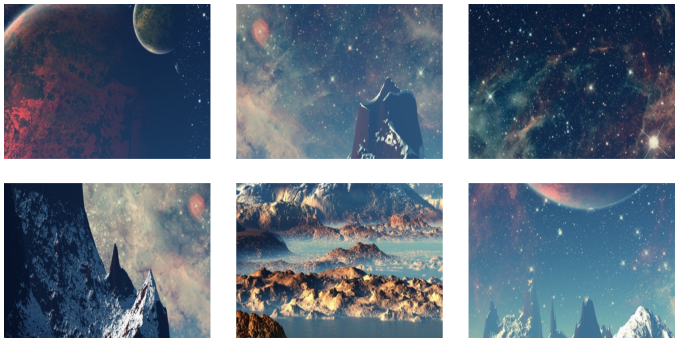

Figure 3.11: Six image patches generated by means of the synthetic data generation tool.

- brightness: addition of constant value b to all pixels
- contrast: multiplication of pixel values with constant value c
- blur: application of a box filter with kernel size $kw \times kh$

By finding a homography M—a perspective transformation specified by rotational and translational parameters—one can obtain image patches and consequently texton histograms to create a training dataset. The tool labels the generated patches with the corresponding simulated x, y-position of the camera model, which represents the position of the UAV.

Figure 3.12: Illustration of the camera model for the synthetic flight. The developed tool extracts image patches from an given image to simulate those taken with the bottom camera of the MAV during an actual flight.

The steps for specifying the homography are outlined in the following. The implementation is partly based on work by Jepson [29]. Hartley et al. [27] describe multiple view geometry and image transformations in computer vision in detail. To simulate camera movements in the 3D world, a 2D to 3D projection of the image is performed first, using the matrix P_3, with the

width w and height h of the image:

$$P_3 = \begin{bmatrix} 1 & 0 & -\frac{w}{2} \\ 0 & 1 & -\frac{h}{2} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The result is a 3D space with the center of the image as point of origin. The camera rotations are specified by the rotation matrix $R = R_x \cdot R_y \cdot R_z$. By building rotation matrices R_x, R_y, and R_z around the axes x, y, and z, the rotations with the corresponding angles α, β, and γ can be defined separately:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The 3D translational matrix T specifies the location of the camera in world coordinates:

$$\tilde{T} = \begin{bmatrix} 1 & 0 & 0 & \mathrm{d}x \\ 0 & 1 & 0 & \mathrm{d}y \\ 0 & 0 & 1 & \mathrm{d}z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, a rotation followed by translation can be specified by matrix H:

$$H = T \cdot R \tag{3.7}$$

However, this matrix H describes how the world is transformed relative to the camera coordinates, while the position of the camera is fixed. Instead, we would like to specify the camera movement relative to a fixed world. To this end, the inverse of H is needed:

$$H' = (T \cdot R)^{-1} = R^{-1} \cdot T^{-1} \tag{3.8}$$

The transposed rotation matrix is equal to its inverse: $R' = R^{-1}$. The inverse of T negates the translations:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -\mathrm{d}x \\ 0 & 1 & 0 & -\mathrm{d}y \\ 0 & 0 & 1 & -\mathrm{d}z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To obtain a 2D image again, a projection from 3D space to 2D is applied using the matrix P_2. The matrix needs the focal distance f (the distance between camera and image).

$$P_2 = \begin{bmatrix} f & 0 & \frac{w}{2} & 0 \\ 0 & f & \frac{h}{2} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The ratio between $\mathrm{d}z$ and f specifies the size of the patch. The final 3×3 perspective transformation matrix M becomes:

$$M = P_2 \cdot R^{-1} \cdot T^{-1} \cdot P_3$$

The pixel values of the image patch at position x, y are calculated by applying the perspective transformation M to the original image:

$$\begin{aligned} \mathrm{patch}(x, y) &= \mathrm{original}(x', y') \\ &= \mathrm{original}\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right) \end{aligned}$$

For modifying brightness and contrast, each pixel value is transformed with

$$\mathrm{patch}(x, y) := c \cdot \mathrm{patch}(x, y) + b$$

The blurring is performed by convolving the image patch with a box filter:

$$\frac{1}{kw \cdot kh} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

The script provides a command line interface for selecting the original image and the amount of desired image patches. It creates a dataset of image patches and a comma-separated values (CSV) file that specifies the sampled values from the random distributions per patch.

Bibliography

[1] Markus Achtelik et al. "Onboard IMU and monocular vision based control for MAVs in unknown in- and outdoor environments". *Robotics and automation (ICRA), 2011 IEEE international conference on*. IEEE. 2011, pp. 3056–3063.

[2] Spencer Ahrens et al. "Vision-based guidance and control of a hovering vehicle in unknown, GPS-denied environments". *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE. 2009, pp. 2643–2648.

[3] Eman R AlBasiouny, Amany Sarhan, and T Medhat. "Mean-shift-FAST algorithm to handle motion-blur with tracking fiducial markers". *Computer Engineering & Systems (ICCES), 2015 Tenth International Conference on*. IEEE. 2015, pp. 286–292.

[4] Adrien Angeli et al. "2D simultaneous localization and mapping for micro air vehicles". *European Micro Aerial Vehicles (EMAV)*. 2006.

[5] Abraham Galton Bachrach. "Autonomous flight in unstructured and unknown indoor environments". PhD thesis. Massachusetts Institute of Technology, 2009.

[6] Nitin Bhatia et al. "Survey of nearest neighbor techniques". *arXiv preprint arXiv:1007.0085* (2010).

[7] Michael Blösch et al. "Vision based MAV navigation in unknown and unstructured environments". *2010 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2010, pp. 21–28.

[8] Pascal Brisset et al. "The Paparazzi solution". *2nd US-European Competition and Workshop on Micro Air Vehicles 2006*. 2006.

[9] Haiyang Chao, Yu Gu, and Marcello Napolitano. "A survey of optical flow techniques for UAV navigation applications". *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*. IEEE. 2013, pp. 710–716.

[10] Hung-Kuo Chu et al. “Halftone QR codes”. *ACM Transactions on Graphics (TOG)* 32.6 (2013), p. 217.

[11] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. “Torch7: A matlab-like environment for machine learning”. *BigLearn, NIPS Workshop*. EPFL-CONF-192376. 2011.

[12] *CPSC 340: Machine Learning and Data Mining*. URL: `https://www.cs.ubc.ca/~schmidtm/Courses/340-F15/L7.pdf` (visited on 08/10/2016).

[13] Pablo d’Angelo et al. *Hugin - Panorama photo stitcher*. URL: `http://hugin.sourceforge.net/` (visited on 08/10/2016).

[14] G.C.H.E. De Croon et al. “Design, aerodynamics, and vision-based control of the DelFly”. *International Journal of Micro Air Vehicles* 1.2 (2009), pp. 71–97.

[15] GCHE De Croon et al. “Sub-sampling: Real-time vision for Micro Air Vehicles”. *Robotics and Autonomous Systems* 60.2 (2012), pp. 167–181.

[16] G.C.H.E De Croon et al. “The appearance variation cue for obstacle avoidance”. *IEEE Transactions on Robotics* 28.2 (2012), pp. 529–534.

[17] Frank Dellaert et al. “Monte carlo localization for mobile robots”. *IEEE International Conference on Robotics and Automation, 1999.* Vol. 2. IEEE. 1999, pp. 1322–1328.

[18] Alexey Dosovitskiy et al. “Discriminative unsupervised feature learning with convolutional neural networks”. *Advances in Neural Information Processing Systems*. 2014, pp. 766–774.

[19] Hans Driessen and Yvo Boers. “MAP estimation in particle filter tracking”. *2008 IET Seminar on Target Tracking and Data Fusion: Algorithms and Applications*. IET. 2008, pp. 41–45.

[20] Daniel Eberli et al. “Vision based position control for MAVs using one single circular landmark”. *Journal of Intelligent & Robotic Systems* 61.1-4 (2011), pp. 495–512.

[21] Sean P Engelson and Drew V McDermott. “Error correction in mobile robot map learning”. *IEEE International Conference on Robotics and Automation, 1992 Proceedings.* IEEE. 1992, pp. 2555–2560.

[22] Dieter Fox et al. “Monte carlo localization: Efficient position estimation for mobile robots”. *AAAI/IAAI* 1999 (1999), pp. 343–349.

[23] S. Garrido-Jurado et al. “Automatic generation and detection of highly reliable fiducial markers under occlusion”. *Pattern Recognition* 47.6 (2014), pp. 2280–2292.

[24] S. Garrido-Jurado et al. “Automatic generation and detection of highly reliable fiducial markers under occlusion”. *Pattern Recognition* 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. DOI: http://dx.doi.org/10.1016/j.patcog.2014.01.005. URL: http://www.sciencedirect.com/science/article/pii/S0031320314000235.

[25] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. “Towards a navigation system for autonomous indoor flying”. *IEEE International Conference on Robotics and Automation, 2009 (ICRA'09)*. IEEE. 2009, pp. 2878–2883.

[26] Isabelle Guyon and André Elisseeff. “An introduction to feature extraction”. *Feature extraction*. Springer, 2006, pp. 1–25.

[27] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[28] Eva Hornecker and Thomas Psik. “Using ARToolKit markers to build tangible prototypes and simulate other technologies”. *IFIP Conference on Human-Computer Interaction*. Springer. 2005, pp. 30–42.

[29] Michael Jepson. *Rotation in 3D using OpenCV's warpPerspective*. URL: http://jepsonsblog.blogspot.nl/2012/11/rotation-in-3d-using-opencvs.html (visited on 12/10/2015).

[30] Hirokazu Kato and Mark Billinghurst. “Marker tracking and HMD calibration for a video-based augmented reality conferencing system”. *2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99), 1999 Proceedings*. IEEE. 1999, pp. 85–94.

[31] Alex Kendall, Matthew Grimes, and Roberto Cipolla. “PoseNet: A convolutional network for real-time 6-DOF camera relocalization”. *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2938–2946.

[32] Teuvo Kohonen. “The self-organizing map”. *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480.

[33] Mirosław Kordos, Marcin Blachnik, and Dawid Strzempa. “Do we need whatever more than k-NN?” *International Conference on Artificial Intelligence and Soft Computing*. Springer. 2010, pp. 414–421.

[34] Yann LeCun et al. “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[35] David G. Lowe. “Object recognition from local scale-invariant features”. *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[36] Kimberly McGuire et al. “Local Histogram Matching for Efficient Optical Flow Computation Applied to Velocity Estimation on Pocket Drones”. *arXiv preprint arXiv:1603.07644* (2016).

[37] *Microsoft Image Composite Editor*. URL: http://research.microsoft.com/en-us/um/redmond/projects/ice/ (visited on 08/02/2016).

[38] Parrot. *CES 2015 : Parrot Bebop Dance Choreography*. 2015. URL: https://www.youtube.com/watch?v=A_3UifFb45Y (visited on 10/15/2015).

[39] *Parrot Bebop Drone*. URL: http://www.parrot.com/products/bebop-drone/ (visited on 07/14/2016).

[40] Franck Ruffier et al. “Bio-inspired optical flow circuits for the visual guidance of micro air vehicles”. *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*. Vol. 3. IEEE. 2003, pp. III–846.

[41] Stephen Se, David Lowe, and Jim Little. “Global localization using distinctive visual features”. *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*. Vol. 1. IEEE. 2002, pp. 226–231.

[42] S. Thrun. *Artificial intelligence for robotics*. URL: https://www.udacity.com/course/artificial-intelligence-for-robotics--cs373 (visited on 03/10/2016).

[43] Manik Varma and Andrew Zisserman. “A statistical approach to texture classification from single images”. *International Journal of Computer Vision* 62.1-2 (2005), pp. 61–81.

[44] Manik Varma and Andrew Zisserman. “Texture classification: Are filter banks necessary?” *2003 IEEE Computer Society Sonference on Computer Vision and Pattern Recognition, 2003 Proceedings*. Vol. 2. IEEE. 2003, pp. II–691.

[45] *Wiki PaparazziUAV*. URL: https://wiki.paparazziuav.org/wiki/Main_Page (visited on 07/14/2016).