Parallel Real-Time Tracking and 3D Reconstruction with TBB for Intelligent Control and Smart Sensing Framework

Mkhoyan, Tigran; de Visser, Coen; De Breuker, Roeland

DOI
10.2514/6.2020-2252

Publication date
2020

Document Version
Final published version

Published in
AIAA Scitech 2020 Forum

AIAA SciTech Forum
6-10 January 2020, Orlando, FL
AIAA Scitech 2020 Forum

10.2514/6.2020-2252

Parallel Real-Time Tracking and 3D Reconstruction With TBB for Intelligent Control and Smart Sensing Framework

T. Mkhoyan*, C. de Visser†, R. De Breuker‡
Delft University of Technology, Delft, The Netherlands

Recent advancements in aircraft controllers paired with increasingly flexible aircraft designs create the need for adaptive and intelligent control systems. To correctly capture the motion of a flexible aircraft wing and provide feedback to the controller, a large number of states (nodes along the span) must be monitored in real-time. Visual sensing methods carry the promise of flexibility needed for this type of smart sensing and control. However, visual sensing requires capturing and tracking keypoint features (marker tracking), while detecting thereof from a feature-rich image can be a computationally intensive task. The computational effort significantly increases with image size or when an image stereo pair is used to find matching keypoints. In this study, a parallel approach is presented with Threading Building Blocks (TBB), using sub-matrix computations, for extraction of corresponding keypoints from an image-stereo pair, and triangulation with the Direct Linear Transform (DLT) method to reconstruct the 3D position of the object in space. Additional robustness is investigated by implementing a Kalman filter for tracking prediction during the domain transition between the sub-matrices. Furthermore, a flexible simulation framework is set up for smart sensing with a coupled unsteady aeroservoelastic model of a 3D wing and a visual model to test the method for intelligent control feedback in a simulation environment. The methodology is tested in a laboratory environment with a stereo camera setup, and in a virtual environment, where the virtual camera parameters are reconstructed to meet a stereo setup. The proposed approach aims to advance the state-of-the-art in smart sensing, particularly in the context of real-time state estimation of aeroelastic structures and control feedback. The parallel approach shows a significant improvement of speed and efficiency, allowing real-time computation from a live image stream at 50 fps.

Nomenclature

A	= Area of a detected shape	$\mathbf{Q}_k$	= Process noise covariance matrix
$\mathbf{A}$	= State matrix	R	= Rotation matrix
$B(x', y')$	= Kernel matrix	R	= Rotation matrix
$\mathbf{B}$	= Input matrix	$\mathbf{R}_k$	= Measurement noise covariance matrix
cam1, cam2	= Camera 1, camera 2	S	= Diagonal singular values matrix (SVD)
c_{circ}	= Circularity score of detected blob shape	T	= Linear translation
c_{min}, c_{max}	= User defined circularity conditions	$\mathbf{t}$	= Orientation vector
c_{d_x}, c_{d_y}	= Shift in principal point	U, V	= Left-, right-singular vector matrices (SVD)
c_x, c_y	= Principal point	$\mathbf{u}_k$	= Input control vector
$f_{dilate}(I(x, y))$	= Dilate operation	V_∞	= Free-stream velocity
$f_{erode}(I(x, y))$	= Erode operation	$\mathbf{w}$	= Process noise vector

*Ph.D. student, Faculty of Aerospace Engineering, Aerospace Structures and Materials department, T.Mkhoyan@tudelft.nl, P.O. Box 5058, 2600GB Delft, The Netherlands.

†Assistant Professor, Faculty of Aerospace Engineering, Control and Operations department, C.C.deVisser@tudelft.nl, P.O. Box 5058, 2600GB Delft, The Netherlands

‡Associate Professor, Faculty of Aerospace Engineering, Aerospace Structures & Computational Mechanics, R.DeBreuker@tudelft.nl, P.O. Box 5058, 2600GB Delft, The Netherlands

$f(I(x,y))$	= Filtering (sequence) operation	W	= Scaling parameter
$f_{morph}(I(x,y))$	= Morphological operations (combined)	w	= Image width
$f_x,\ f_y$	= Focal lengths [in pixels]	w_{sensor}	= Width of the camera sensor
$\mathbf{F}_k$	= State transition matrix	$w_g\,(t)$	= Gust profile
$\mathbf{G}_k$	= Control input matrix	$\hat{\boldsymbol{X}}$	= Estimate of the coordinate vector (4D)
K	= Camera matrix (intrinsic)	$\mathbf{x}_k$	= State vector
K_f	= Actuator stiffness	$\mathbf{x}_{ae}$	= State vector
$\mathbf{K}_s,\ \mathbf{K}_s^{aug}$	= Structural stiffness matrix, augmented	$(x',\ y')$	= Local coordinates
M_f^{act}	= Actuator moment	$(x,\ y)$	= Global coordinates
$\mathbf{M}_s,\ \mathbf{M}_s^{aug}$	= Structural mass matrix, augmented	$x_0,\ y_0$	= Optical centres
N	= Number of threads (parallelization)	$z_1,\ z_2$	= Aerodynamic lag states
O_p	= Projection reference frame	$K_1,\ K_2$	= Camera matrices (cameras 1 & 2)
O_w	= World reference frame	$w,\ \phi,\ \theta$	= Structural degrees-of-freedom
$O_1,\ O_2$	= Reference frames of cameras 1 & 2	α	= Angle-of-attack
o	= Perimeter of a detected shape	β	= Flap rotation angle
$P_1,\ P_2$	= Projection matrices (cameras 1 & 2)	ρ_{air}	= Air density
$\mathbf{P}_k$	= Covariance matrix	ω_t	= Perturbation on particle acceleration

Introduction

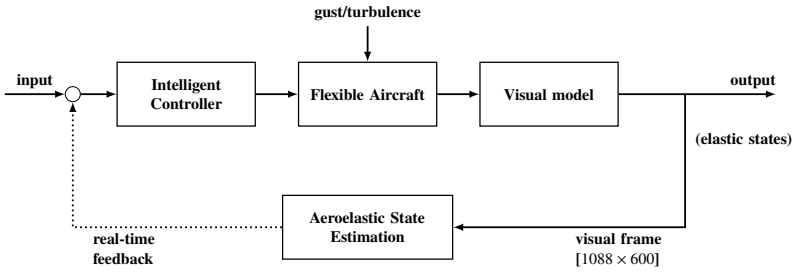


Figure 1 A schematic of the adaptive intelligent control setup consisting of an intelligent controller, flexible aircraft model, visual model, and aeroelastic state estimation using visual tracking. This study aims to contribute to the state-of-the-art in the field of adaptive intelligent control by closing the control feedback loop (dotted line).

With continued advancements in aircraft controllers and increasingly flexible aircraft designs the need arises for the development of novel (smart) adaptive sensing methods. To correctly capture the motion of a flexible aircraft wing and to provide feedback to the controller, a large number of states (nodes along the span) must be monitored in real-time. Visual methods can provide the flexibility needed for this type of smart sensing and control. A simplified schematic of smart sensing with visual feedback is shown in Fig. 1. The use of visual information for observing deformations has been successfully applied to wind tunnel models in early studies [1]. This approach has also seen a wide application in robot manipulation [2]. However, in recent years, the capability in terms of on-board computation and camera quality has immensely increased. At the same time, the hardware has become more compact [3, 4], which creates the possibility for numerous embedded applications using a camera as a sensor.

However, visual-based methods do not come without challenges. Visual sensing requires capturing and tracking of keypoint features. These are points of interest that, in the case of a flexible smart wing, can be visual markers placed at specific wing node locations. However, extracting these keypoints from a feature-rich image can be a computationally intensive task. The computational effort is critical when keypoint detection is being applied in real-time, as resources must be shared between various processes. The factors that negatively impact the cost are (i) high frame rates and (ii) large image sizes. A third factor that can cause a bottleneck in the computation is (iii) when a stereo image pair is used. The latter is the case when the corresponding keypoints need to be extracted from a pair of images intended to reconstruct a 3D point, be it a tracked node on the wing, a point cloud of a scene or an object. In this configuration, the overall

computation time is effectively doubled. Parallelizing the detection and processing pipeline can relax the computational effort by allowing for the computation to occur on multiple threads. In this study, a parallel approach to keypoint detection is proposed using multi-threading with a blob detector. The goal is to (i) speed up the process of keypoint extraction and detection in corresponding image pairs, (ii) reconstruct the 3D points from a stereo correspondence and (iii) reconstruct the state of the aircraft wing in terms of its displacements such that this information can be used as a feedback for the proposed intelligent control methodology. The purpose is to advance the state-fo-the-art in smart-sensing by including visual information into the control feedback loop.

The proposed methodology is tested in a laboratory environment using a stereo setup. Subsequently, a visual simulation framework is developed for an aeroservoelastic wing undergoing gust excitation in which the virtual camera parameters are reconstructed to meet those of the stereo setup. The parallel keypoint extraction is capable of significantly speeding up the extraction process by parallelizing the relatively heavy computational tasks across multiple threads over smaller sub-matrices. Performance multipliers of ×5-6 were achieved using this approach on image sequences of 1088×600. The speed-up allowed saving the remaining computational budget for the other computational tasks and enabled the application of the 3D reconstruction pipeline in a real-time smart sensing control scheme.

To reduce the frequency of computationally heavy tasks, such as feature extraction and matching, an implementation of a Kalman filter is proposed. The task of the Kalman filter is to provide prediction during the missing intervals of measurement updates, which would normally be provided by the feature extraction process. It is shown how simple linear dynamics can increase the robustness of the tracking for reduced intervals of 2, 4, 8, and 10 skips in consecutive frames. Furthermore, it is shown how the corresponding keypoints from a stereo image pair could be used to reconstruct the deflections of an unsteady aeroelastic wing model and serve as sensor feedback to the controller. For this, a smart-sensing virtual framework was developed with an aeroservoelastic model animated in Blender. Recommendations are made on how to use an adaptive sub-matrix division based on keypoint characteristics and how the framework can be used in future studies for the development of smart visual sensing and control.

I. Methodology

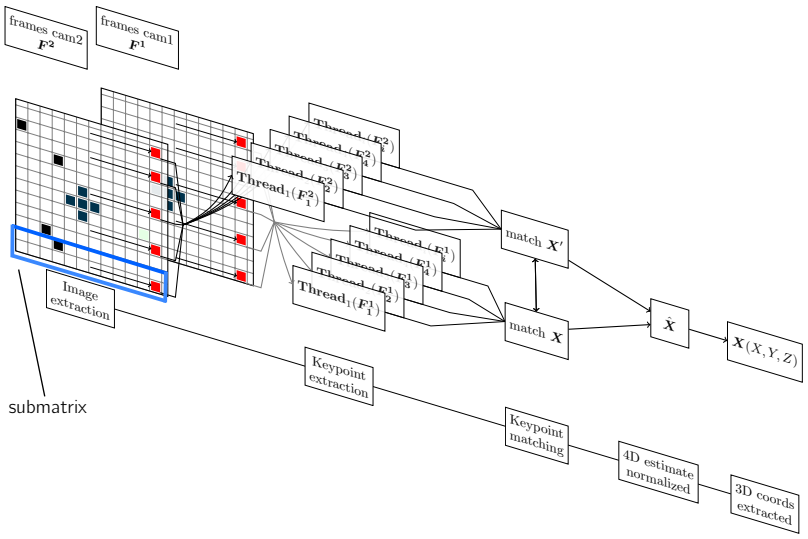


Figure 2 The parallel keypoint extraction and triangulation pipeline. The process consists of frame extraction, parallel keypoint extraction, matching and 3D reconstruction. The result is the 3D location of a point or points in 3D space relative to the reference frame of camera 1 (cam1).

The method used in this study consists of a parallel keypoint detection and 3D point reconstruction pipeline from a pair of continuously streamed stereo images. The parallelization achieved by distributing the keypoint extraction task across smaller N sub-matrices of the original full image matrix in N separate threads. The keypoints are extracted in the local sub-matrix coordinates $(x',\ y')$, and converted to the global coordinates $(x,\ y)$ after each parallel task. The

pipeline is shown in Fig. 2.

A. Image processing and tracking

In order to obtain potential keypoints from two image streams, first, *potential* keypoints of interest need to be extracted from the sub-matrix. In this context, the points of interest are markers of a circular shape, hence the potential keypoints are herewith referred to as blobs. The blobs need to be extracted from a single frame ánd tracked through an image input sequence. Extracting blobs can be done relatively efficiently on grey or binary images. To avoid direct computation on a multichannel ($1088 \times 600 \times 3$) RGB images, a cascade image filtering pipeline can be adopted on each sub-matrix.

1. Image transformations

The proposed cascaded filter entails application of image thresholding followed by morphological operations. The image is thresholded adaptively using the Otsu's method [5] based on bimodal pixel histogram distribution. This allows obtaining a 1-channel pixel mask from a 3-channel input image. To remove the noise from the image and improve blob quality, a sequence of morphological operations can be applied, i.e. erode followed by dilate [6]. Both of these image transformations are essentially convolution operations of the image, $I(x, y)$, with the kernel, $B(x', y')$. The first performs a local *min* operation with a kernel of the desired size (e.g. 3×3), anchored at the centre. As the kernel slides over the image, the pixel value under the anchor point is replaced by the *min* value of the region covered by the kernel $B(x', y')$. Dilate operator works according to the same principle but performs a local *max* operation instead. The filters can be summarized as follows:

$$f_{erode}(I(x, y)) = \min_{(x', y') \in B_{ker}} (I(x + x', y + y')) \tag{1}$$

$$f_{dilate}(I(x, y)) = \max_{(x', y') \in B_{ker}} (I(x + x', y + y')) \tag{2}$$

and combined operation:

$$f_{morph}(I(x, y)) = f_{dilate}(f_{erode}(I(x, y))) \tag{3}$$

For an appropriate kernel size, this operation removes noisy speckles (Gaussian noise) around nearby blobs missed by thresholding, and enlarge the bright pixel areas left after the operation, increasing blob quality. These operations are, in general, useful for the removal of global noise, but also for isolating and joining separate individual elements.

2. Blob detection

To extract the circular blobs from the image, a contour filter, often referred to as a blob detector, is applied. The filter used in the study is based on the Topological Structural Analysis algorithm of binary images and shapes [7]. The general principle involves a border following technique with the aid of topological analysis to define the contours of a border shape. Once the contours are obtained, the blob detector can be configured to filter the shapes that match the desired geometrical properties, such as area, convexity, circularity, inertia ratios, and colour properties. The circularity of a detected shape can be quantified as a constraint on the blob filter defined as c_{circ}:

$$c_{circ} = \frac{4\pi A}{o \cdot o} \tag{4}$$

where A and o are the area and perimeter of the shape, respectively. The detected shape must meet the user-defined c_{min} and c_{max} conditions:

$$c_{min} \leq c_{circ} < c_{max} \tag{5}$$

In this study, the contour filter was tuned in terms of area and circularity such that it allowed geometrical deviation from the perfect circular marker shape and sufficiently visible markers. Parameters of the blob filter are shown in Table 1 in Appendix A.

In the final step, to perform the triangulation of a distinct point represented by marker shape, the centroid of the selected contours is calculated. A schematic of the cascaded tracking filter is shown in Fig. 3. The methods and the processing pipeline were developed in the C++ programming language using the OpenCV open-source computer vision library [8]. The main body of the algorithm is given in Appendix B as Algorithm 3.

Figure 3 Cascaded tracking filter pipeline.

3. Feature matching

For feature mathching, various approaches are possible. In the current study, the feature matching can be performed via keypoint descriptors provided by the blob detector. Blob detector actively filters on the specific geometrical features that are computed by a typical descriptor (e.g. the circularity, convexity etc.). Following the descriptor computations matching can be performed based on these characteristics. In the current implementation circularity, convexity and distance of corresponding points is considered.

Another widely used method is with ORB descriptors (Oriented FAST and Rotated BRIEF) [9]. It was originally developed by the OpenCV [8] team as an alternative to other widely used methods, SIFT (Scale-Invariant Feature Transform) [10] and its faster counterpart SURF (Speeded-Up Robust Features) [11]. ORB has good performance and is widely adopted in many studies [12, 13]. However, it still requires significantly heavy computations. In particular, when the feature detection and descriptor computation are separated, various possibilities exist. The descriptor matching can be either performed in parallel on smaller sub-matrices after the features are efficiently extracted in parallel. The parallel methodology proposed in this study lends itself very useful for this purpose. And the ORB can perform the tasks of the blob detector in the parallel threads. Initial test was performed with ORB in the current methodology.

B. Parallelization

The parallelization approach and the processing pipeline, as schematized in Fig. 2, follow the image acquisition from the two cameras (cam1 and cam2). Here, the keypoint extraction process is split into N threads of N sub-matrices. The global location in the frame $(x,\ y)$ is recovered, after which a matching algorithm finds the matching keypoints. Then an estimate is obtained in 4D $(X,\ Y,\ Z,\ W)$ (homogeneous) of the coordinates, denoted $\hat{\boldsymbol{X}}$, from the correspondence of points. Finally, the 3D location of a point or points is reconstructed using the homogeneous Direct Linear Transform (DLT). The location is obtained relative to reference frame (O_1) of camera 1 (cam1). More details on the 3D reconstruction are given in Section I.E.

Multi-threading is achieved using the Threading Building Blocks (TBB) C++ template library [14]. In the current setup, the image is split into equally sized horizontal sub-matrices. The extraction pipeline is tested on various image sizes to find an optimum for the number of threads. For certain conditions, spawning a thread may be more costly than processing bigger sub-matrices. For this task, it was found that the optimum number of threads was between 4-8, relating to a square pixel area of approximately 80000 pixel^2 per thread of 3-channel image (RGB). The main body of the parallel process with N-threads and the steps performed on the N sub-matrices is given below as Algorithm 1.

Algorithm 1: Definition of *ParDetector*() class and parallel operator body *void operator*() function

```
/* definition of ParDetector with class inputs */
Class ParDetector(Img, keypoints, subWidth, range):
    /* Initialize blobDetector() and reserve containers:  */
    Initiallize(keypoints, subImg, detector(blobDetectorParams));
    /* main body of parallel process */
    Function void operator(range):
        /* Allocate chunks of matrices to each parallel process */
        for k = range.start; while k < range.end; k++ do
            Initialize(subkeypoints) // container local frame keypoints
            Rect subRect(0, (Img.rows/subWidth)·i, Img.cols,Img.rows/subWidth);
            Mat subImg(Img,subRect);
            detector→detect(subImg, keypoints);
            /* draw in local, offset to global position!  */
            drawKeypoints(subImg, keypoints, subImg, Scalar(0, 0, 255));
            offsetKeyPoints(subkeypoints,subRect.topleft) // anchor top left
            /* insert to global keypoints with make_move_iterator */
            keypoints.insert(keypoints.end,iterator(subkeypoints.begin,subkeypoints.end))
        end
        return void; // join parallel processes
    end;
    Function offsetKeyPointsY(keypoint,offsetY):
        for k = keypoint.start; while k < keypoint.end; k++ do
            keypoint.y = keypoint.y + offsetY;
        end
    end;
end class
```

C. Keypoint extraction and matching

Parallelization allows performing an expensive filter operation on smaller sub-frames in parallel fashion while distributing the computational load across multiple cores and thereby reducing the processing time. Parallelization becomes challenging when filter operations cannot be done independently. In the current processing pipeline, the global bottleneck appears when the processing step requires full images from two image streams, as it is the case in the keypoint matching step.

To remedy this problem and also to increase the robustness of the tracking, it is proposed to adopt a so-called dynamic keypoint matching and to apply a Kalman Filter (KF) to reduce the frequency of keypoint matching. Keypoint matching can be considered a static process; however, in the given problem, tracked markers are moving objects. Hence the keypoint matching can be reduced in frequency, while the Kalman filter takes care of the predictions in-between the skipped steps. Additionally, to robustify the tracking, a Kalman filter can also be applied to the final triangulated result.

D. Discrete KF formulation

In the simplest form, the tracking problem can be approached using a linear Kalman filter model, containing the dynamics of a moving particle. This model allows one to obtain a *prediction* in-between the skipped keypoint matching steps, and to *update* when measurements (matched keypoint locations) are available.

In the given tracking problem a general model for particle motion can be adopted:

$$\ddot{x}(t) = \omega(t) \tag{6}$$

In this model we assume a constant velocity particle subject to perturbation $\omega(t)$ which is assumed to have a random distribution. Following this, a continuous state-space model can be derived, with the state vector representing the position and velocity (i.e. $\mathbf{x} = \begin{bmatrix} x & \dot{x} \end{bmatrix}^T$):

$$\frac{\partial \dot{x}(t)}{\partial t} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\omega(t) \tag{7}$$

And the state matrix $\mathbf{A}$ and the input matrix $\mathbf{B}$ given by:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{8}$$

The state-space formulation allows to conveniently write a vector of states in terms of its first order derivatives and forms a basis for the Kalman filter.

To compute the Kalman filter updates and predictions at a specific capture intervals, $h = \Delta t$, of the camera, a discrete Kalman filter form is required. This can be achieved by deriving a typical Euler integration, of state variable x in the form:

$$\frac{dx(t)}{dt} = \dot{x}(t) = f'(x(t)) = \frac{x(t+h) - x(t)}{h} \tag{9}$$

Rewriting gives the definition of next time step:

$$x(t+h) = x(t) + f'(x(t))h \tag{10}$$

Now in discrete form using $k = t + h$ the state x for current step is defined in terms of its derivative as:

$$x_k = x_{k-1} + \dot{x}_{k-1} \cdot h \tag{11}$$

This Euler integration shown above puts the state vector in a convenient state space from, where we can rewrite a vector of states in terms of its first order derivatives. Kalman filter model has a similar form to a typical state-space model, and for a state vector $\mathbf{x} \in \mathbb{R}^{n\times 1}$, containing the n-desired states, and m-inputs, the general model can be represented as:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{G}_k \mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{12}$$

Here, $\mathbf{F} \in \mathbb{R}^{n\times n}$ is the state transition matrix, $\mathbf{G}_k \in \mathbb{R}^{n\times m}$ the control input transition matrix, $\mathbf{u}_k \in \mathbb{R}^{m\times 1}$ the input control vector and $\mathbf{w}$ the system or process noise vector. The state vector can be observed by measurement at time k, taking the form of the linear equation:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{13}$$

To track a point in 2D image, the motion of a particle can be described by position and velocities in $x,\ y$. Following the kinematic model of a particle under perturbation as described earlier (7 and 8), a representative discrete Kalman filter model of a particle motion in $x,\ y$ for time steps $h = \Delta t$ can be constructed as follows:

$$\mathbf{x}_k = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, \ \mathbf{F}_k \begin{bmatrix} 1 & 0 & h & 0 \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

It must be noted that in this form we have an observation model without known control inputs hence the model is assumed to be driven by perturbation of assumed statistical properties. The state transition matrix takes this form by adopting the Euler integration over interval h as shown previously. It can be shown that the form can be directly obtained from 7 by taking the Laplace transform followed by inverse Laplace transform shifted forward in time as: $\mathbf{F}_k = e^{\mathbf{A}h}$. Given that the keypoint matching provides a feedback of position measurement, the observations take the from:

$$\mathbf{z}_k = \begin{bmatrix} z_x \\ z_y \end{bmatrix}, \quad \mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{15}$$

The model under consideration is subject to uncertainty due to randomly distributed perturbation of the acceleration. Measurements can be subject to uncertainty as well due to sensor errors. These uncertainties can be collected in the process noise covariance matrix, $\mathbf{Q} \in \mathbb{R}^{n\times n}$, and measurement noise covariance matrix, $\mathbf{R} \in \mathbb{R}^{m\times m}$. Various strategies exist to assemble these matrices, for the given problem it can be assumed that the standard deviation of process noise,

σ_q, and measurement noise, σ_r are constant, and that the process noise in $x,\ y$ is considered not to have cross-coupled terms. The process noise covariance matrix for n states can be assembled as:

$$\mathbf{Q}_k = E\left[w_k w_k^T\right] = \begin{bmatrix} \frac{1}{3}h^3 & 0 & \frac{1}{2}h^2 & 0 \\ 0 & \frac{1}{3}h^3 & 0 & \frac{1}{2}h^2 \\ \frac{1}{2}h^2 & 0 & h & 0 \\ 0 & \frac{1}{2}h^2 & 0 & h \end{bmatrix} \sigma_r^2 \tag{16}$$

Here σ_r^2 is a scaling factor related to the standard deviation of the acceleration. Similarly, we assume the standard deviation of measurement noise in x, y, σ_r, to be constant, and measurement noise covariance matrix for the observations of m outputs take form:

$$\mathbf{R}_k = \begin{bmatrix} \sigma_{q_x} & 0 \\ 0 & \sigma_{q_x} \end{bmatrix} \sigma_r^2; \tag{17}$$

Uncertainty in the initial state can be assembled in the covariance matrix at the initial step, as a diagonal matrix with state variances:

$$\mathbf{P}_{0|0} = \begin{bmatrix} \sigma_x^2 & 0 & 0 & 0 \\ 0 & \dot{\sigma}_x^2 & 0 & 0 \\ 0 & 0 & \sigma_y^2 & 0 \\ 0 & 0 & 0 & \dot{\sigma}_y^2 \end{bmatrix} l_p \tag{18}$$

An a priori estimate of the state vector $\mathbf{x}_k$, can be obtained:

$$\mathbf{x}_k^- = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{G}_k \mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{19}$$

An a priori estimate of covariance matrix, $\mathbf{P}_k$, can be obtained:

$$\mathbf{P}_k^- = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k + \mathbf{Q}_{k-1} \tag{20}$$

Where the two equations are used two predict based on past expectation. Then the Kalman gain can be computed to weigh new observations and perform optimal updates on $\mathbf{x}_k$ and $\mathbf{P}_k$:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_{k-1} \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-2} \tag{21}$$

The update is then performed using:

$$\mathbf{x}_k = \mathbf{x}_k^- \mathbf{K}_k (\mathbf{z}_k^- - \mathbf{H}_k \mathbf{x}_k^- + \mathbf{R}_k)^{-2} \tag{22}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}\mathbf{H}_k)\mathbf{P}_k^- \tag{23}$$

The tracker can then be composed of N-instances of trackers.

E. 3D reconstruction approach

The 3D points are reconstructed using the DLT method described by [15]; details about the method are presented in Section I.F. Triangulation requires the intrinsic and extrinsic parameters of the cameras. The intrinsic parameters are needed to compensate radial and tangential lens distortion for the keypoints found in each individual frame, while the extrinsic parameters are used to obtain the projection matrices describing the relative position of the cameras with respect to each other.

The intrinsic parameters are obtained by calibrating each camera individually on a sequence of chequerboard images. The chequerboard used for calibration is an $A3$ sized 9×6 board, each square having a unit size of 35.25 mm. The square unit size is the scaling unit in the rotation and translation matrices. The calibration method used in this study is described in [16]. It consists of an optimisation problem where the goal is to minimize the re-projection error, which is the error between the extracted and re-projected pixel coordinates of a calibration pattern.

A pinhole camera is used for the calibration sequence of the chequerboard patterns, where the orientation of the object in (X, Y, Z) world coordinates (reference frame O_w) relative to its projection on the image plane in (x, y) coordinates (reference frame O_p), is expressed by the following transform:

$$\underbrace{s\begin{bmatrix}x\\y\\1\end{bmatrix}}_{O_p\ c.s.} = \underbrace{\begin{bmatrix}f_x & 0 & x_0\\0 & f_y & y_0\\0 & 0 & 1\end{bmatrix}}_{K}\underbrace{\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{bmatrix}}_{R\,|\,\mathbf{t}}\underbrace{\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}}_{O_w\ c.s.} \tag{24}$$

where K is the camera matrix; x_0 and y_0 are the optical centres (optical points) in x and y pixel coordinates of the frame; f_x and f_y are the focal lengths in pixel units. Then, for each snapshot, matrix R and vector $\mathbf{t}$ represent the rotation and orientation of the chequerboard plane, with the reference frame O_w located at the top left corner. In the image sequence, the chequerboard corners ($9 \times 6 = 54$) are the world coordinate input points, and all the points are assumed to lie on the Z plane. It is therefore essential to use a good quality calibration board and ensure that the surface is not deformed. Furthermore a distortion matrix $D_1 \in \mathbb{R}^{1\times5}$ is obtained after the calibration sequence that contains five distortion parameters.

The extrinsic parameters are obtained by using stereo-rectification on the corresponding chequerboard corners extracted from the corresponding image pairs. The output of the rectification (amongst others) is the augmented 3×4 matrix $\left[R\,|\,\mathbf{t}\right]$ describing the relative position and orientation in 3D space of cam2 with respect to cam1. The rotation and translation matrices contain respectively in units corresponding to the scaling unit size.

$$T = \begin{bmatrix}T_x & T_y & T_z\end{bmatrix}^T \tag{25}$$

where T is given as a linear translation along the respective axes, and the rotation matrix:

$$R = \begin{bmatrix}r_{11} & r_{12} & r_{13}\\r_{21} & r_{22} & r_{23}\\r_{31} & r_{32} & r_{33}\end{bmatrix} \tag{26}$$

where each column corresponds to the orientation of the cam2 with respect to cam1 in X, Y and Z axes, expressed in radians. For computation, generally a more compact form of the rotation matrix is used consisting of the following parameters in vector form $\mathbf{r}_{vec} = \begin{bmatrix}r_x & r_y & r_z\end{bmatrix}$. In this form the desired rotation of the coordinate system is given by a rotation around a single vector by an angle θ corresponding to the magnitude of $\mathbf{r}$. The conversion to the compact form is achieved by the Rodrigues' formula [15]:

$$R = \mathbf{I} + \sin(\theta)\mathbf{r}\mathbf{r}^t + (1 - \cos(\theta))\,\mathbf{r}\mathbf{r}^t\begin{bmatrix}0 & -r_2 & r_y\\r_z & 0 & r_x\\r_y & r_x & 0\end{bmatrix} \tag{27}$$

where the 3×3 matrix on the right is composed of X, Y and Z components of the vector $\mathbf{r}$, and its magnitude $\|r_{vec}\|$. The rotation matrix R (or the rotation vector), as well as the translation vector T must establish a reference frame of cam1, O_1, for the rectification and the triangulation. This is illustrated in Fig. 4. In order to estimate a 3D point given by estimate $\hat{\mathbf{X}}$, the projection matrices P_1 and P_2 for each camera must be found. These matrices enable construction of a homogeneous representation of a points $\mathbf{X}$, $\mathbf{x}$ and $\mathbf{x'}$ in their respective frames. The two projection matrices are related through:

$$\hat{\mathbf{x}} = P\hat{\mathbf{X}}, \quad \hat{\mathbf{x}}' = P'\hat{\mathbf{X}} \tag{28}$$

The representation is referred to as homogeneous, meaning that only the ratio of the projection matrix elements is significant and the projective transformation is maintained when a scaling factor is applied to this matrix. The scaling factor is related to the representation of a point in 4D (X, Y, Z, W) space using the 4th (scaling) parameter W, which resolves the scale ambiguity. After applying the transformation K_1 and K_2 and for the purpose of consistency we use

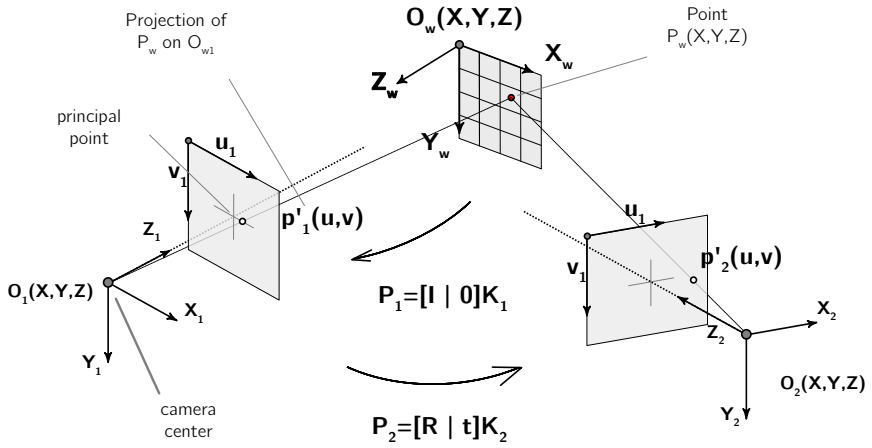


Figure 4 A schematic of the stereo camera setup used in the study, and the various coordinate systems. Cam1 coordinate system (O_1) is used as the reference coordinate system for the 3D reconstruction.

subscript 1 and 2 for the notation of the projection matrices of cameras 1 and 2, respectively. The projection matrices, both $\in \mathbb{R}^{3\times3}$, P_1 for cam1 (positioned in O_1) and P_2 for cam2 (positioned in O_2) are constructed as follows:

$$P_2 = K_2\left[R|T\right], \quad P_1 = K_1\left[I|\mathbf{0}\right] \tag{29}$$

where $\mathbf{0} = [0 \quad 0 \quad 0]^T$ is a null vector and I is the 3×3 identity matrix. Since the reference coordinate system is the origin of O_1, matrix R is taken as all-ones and translation $T = [0 \quad 0 \quad 0]$. This effectively means that the camera matrix is aligned with the world axis (thus, O_1 is the reference for 3D reconstruction). Now, with the matrices K_1, K_2, P_1, P_2, everything is in place to apply triangulation to a correspondence of points obtained by parallel keypoint extraction and matching.

F. DLT homogeneous method

The Direct Linear Transform (DLT) method by [15] is discussed in this section. In each image, a correspondence of the projection of a point $P(X, Y, Z)$ in 3D space (O_w), is found in pixel coordinates of image frames 1 and 2, corresponding to the reference frames 0_{p1} and 0_{p2}, respectively. This is a set of measurements $\mathbf{x} = P\mathbf{X}$, $\mathbf{x}' = P'\mathbf{X}$ in two frames. In this equations P and P' are the projection matrices P_1 and P_2, obtained in the calibration step. This method is referred to as homogeneous since it assumes that the 3×1 vectors on the left and the right-hand side of both equations have the same direction, but not the same scale. In other words when represented in 4D 1×4 vector (X, Y, Z, W), the two vectors ($\mathbf{x}$ andPX) will have an equal direction but different last scaling parameter W. The two observations can be combined into an linear equation of the form $AX = 0$. The first step is to eliminate the scaling factor by applying a crossproduct $\mathbf{x} \times (P\mathbf{X})$ yielding the following three equations (the last row is eliminated):

$$x\left(\mathbf{p}^{3T}X\right) - \left(\mathbf{p}^{1T}X\right) = 0 \tag{30}$$

$$y\left(\mathbf{p}^{3T}X\right) - \left(\mathbf{p}^{2T}X\right) = 0 \tag{31}$$

$$x\left(\mathbf{p}^{2T}X\right) - y\left(\mathbf{p}^{1T}X\right) = 0 \tag{32}$$

Here, $\mathbf{p}^i$ entries are the rows of the P-matrices. Herewith, the linear components of $\mathbf{X}$ can be compiled into:

$$\begin{bmatrix} x(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{1T}\mathbf{X}) = 0 \\ y(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{2T}\mathbf{X}) = 0 \\ x'(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{1T}\mathbf{X}) = 0 \\ y'(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{2T}\mathbf{X}) = 0 \end{bmatrix} \tag{33}$$

In the homogeneous DLT, the solution to the four equations is found by performing a unit Singular Value Decomposition (SVD) of the smallest value of A:

$$A = USV \tag{34}$$

where the matrix V contains the homogeneous 4D coordinate points:

$$\begin{bmatrix} V_{14} & V_{24} & V_{34} & V_{44} \end{bmatrix} \tag{35}$$

To convert these from homogeneous 4D to 3D coordinates given in the reference frame of cam1 (0_1), the last entry is used as scaling:

$$\hat{\mathbf{X}} = \frac{1}{V_{44}} \begin{bmatrix} V_{14} & V_{24} & V_{34} & V_{44} \end{bmatrix} \tag{36}$$

The entire pipeline is shown in Fig. 2.

II. Smart Control and Sensing Framework

This section describes the methodology behind the proposed smart sensing framework and how it can be used to develop and asses intelligent control routines for visual-based control methods using a realistic virtual environment.

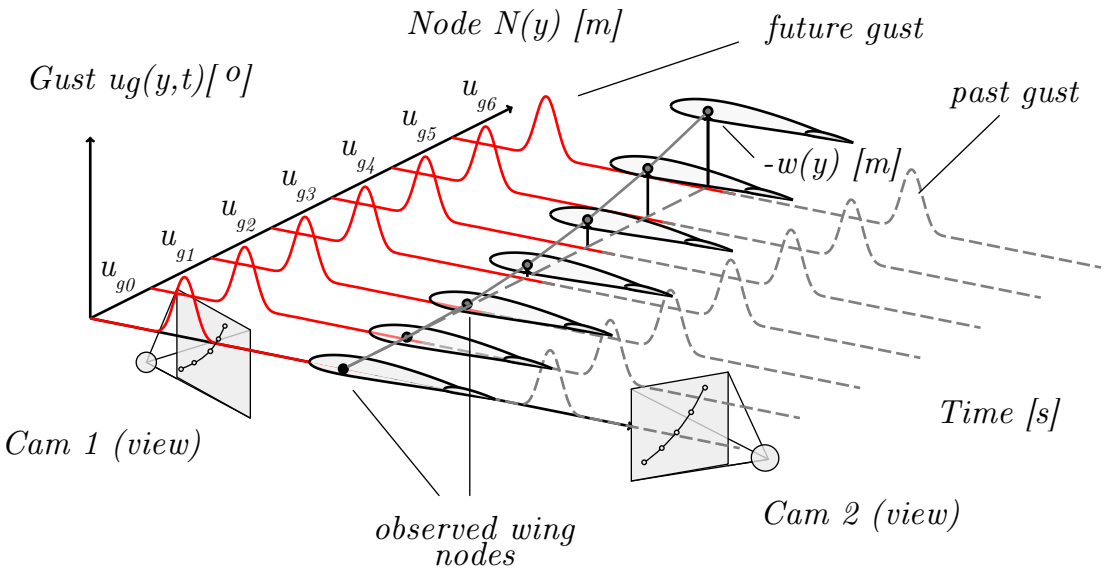


Figure 5 Schematic overview of the observations from two camera viewpoints of the wing undergoing gust excitation. The camera matrices are extracted from the visual model such that DLT triangulation method can be applied.

A. Aeroservoelastic model

To verify the parallel 3D reconstruction approach in the context of smart sensing of an aircraft model, an aeroservoelastic wing model was built in SIMULINK [17]. The model is adapted from [18] and represents a coupled unsteady aeroservoelastic model, trimmed at an air density $\rho_{\text{air}} = 1.22$ kg/m^3 and free stream velocity $V_\infty = 10$ m/s. The model features 8 aerodynamics strips placed at equal distances along the span and 8 flaps. The structure is modelled as a linear Euler-Bernoulli beam, the actuator model is constructed of a lumped spring with stiffness K_f and an applied actuator moment M_f^{act}.

The distribution of the aerodynamics on a 3D wing is modelled using the two-dimensional strip theory where the unsteady aerodynamic forces on each strip are represented in a time-domain formulation, equivalent to the Theodorsen's

frequency-domain model [19]. The time-domain formulation used in this study is the indicial function approximation by Leishman [20]. The coupling of the structural and the aerodynamic models is described in [18].

The full aeroservoelastic model contains the following 6 states:

$$\mathbf{x}_{ae} = \begin{bmatrix} w & \phi & \theta & \beta & z_1 & z_2 \end{bmatrix} \tag{37}$$

where w, ϕ, θ represent the beam nodal degrees-of-freedom for each of the 8 nodes, z_1, z_2 are the aerodynamic lag states, and β the flap rotation angle. The state vector $\mathbf{x}_{ae}$ can be extended to include the first time-derivatives of the structural and aerodynamic states ($\dot{w}$, $\dot{\phi}$, $\dot{\theta}$, $\dot{\beta}$, $\dot{z_1}$, $\dot{z_2}$) for control purposes. In the aerodynamic model, the circulatory lift coefficient due to pitch and the lift coefficient due to flap angle, described by Leishman [20], have been rewritten into a single equation to eliminate two of the four lag states (details can be found in [18]).

The structural mass and stiffness matrices $\mathbf{M}_s$ and $\mathbf{K}_s$ are augmented to include the effect of the flap, yielding $\mathbf{M}_s^{aug}$ and $\mathbf{K}_s^{aug}$, as follows:

$$\mathbf{M}_s^{aug} = \begin{bmatrix} \begin{bmatrix} & \mathbf{M}_s & \end{bmatrix} & \begin{matrix} S_\beta \\ 0 \\ I_\beta + b(c-a)S_\beta \end{matrix} \\ \begin{matrix} S_\beta & 0 & I_\beta + b(c-a)S_\beta \end{matrix} & I_\beta \end{bmatrix}, \qquad \mathbf{K}_s^{aug} = \begin{bmatrix} \begin{bmatrix} & \mathbf{K}_s & \end{bmatrix} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & K_\beta \end{bmatrix} \tag{38}$$

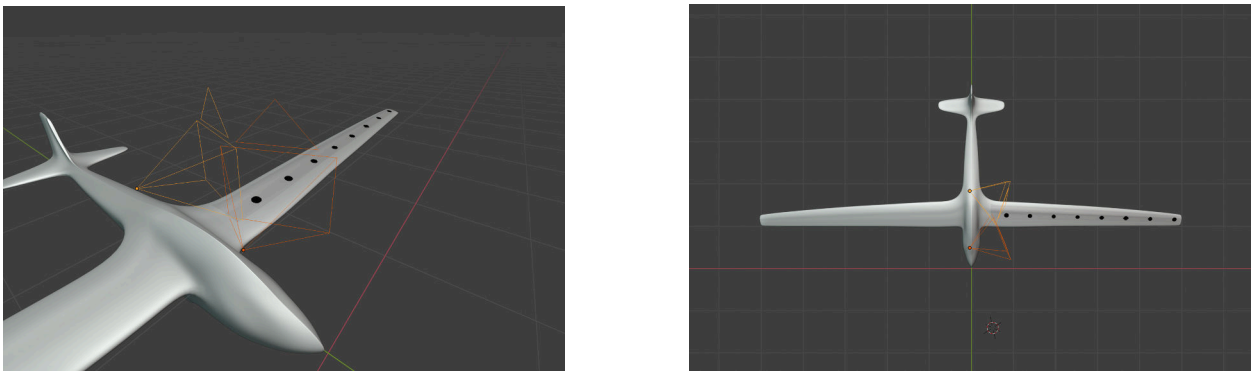
Here, the state β is coupled through inertia coupling and a rotational spring, serving as actuator stiffness. The states that are observed and attempted to be reconstructed from the visual model are the node displacements (plunge) w. In total 7 nodes are considered; the 8$^{\text{th}}$ node is the reference node at the root (clamped). The model is controlled by an Linear-Quadratic Regulator (LQR) controller designed to minimize the gust response for minimum bending moment and control effort. The gust loads are calculated at an angle-of-attack α of 0°. The gust model is a 1-cosine gust profile, assumed uniform across the span and represented as an increment in α:

$$w_g(t) = W_g \left(1 - \cos\left(\omega_g t\right)\right) \tag{39}$$

where W_g is the gust intensity set at 2, and ω_g, the gust frequency set at $10\,(2\pi)$ rad/s. The goal is to use the aeroservoelastic model to animate a visual model by feeding the displacements of output of the wing. This visual is then observed using the parallel 3D reconstruction approach, as illustrated in Fig. 6. The process is schematized in Fig. 5. Here, a wing is shown undergoing gust excitation, causing deflections of the pre-defined wing nodes. The observed displacements from two viewpoints are reconstructed to obtain an $\hat{X}_i$ estimate for each i-th node, such that it can be used for control feedback and the assessment of the controller. More details regarding the virtual setup are provided in the following section.

B. Virtual environment

The virtual graphical environment was created using an open-source 3D animation software Blender [21]. In the virtual environment, the model is represented as a high aspect ratio flexible wing glider in line with the mathematical model by [18] described in Section II.A. A rendering from the virtual environment is given in Fig. 6, showing an overview of the glider model and the global orientation of the cameras (the orange triangular objects) positioned in a typical stereo camera setup. The virtual environment enables experimenting with the positioning of the cameras without the need for an elaborate calibration routine.

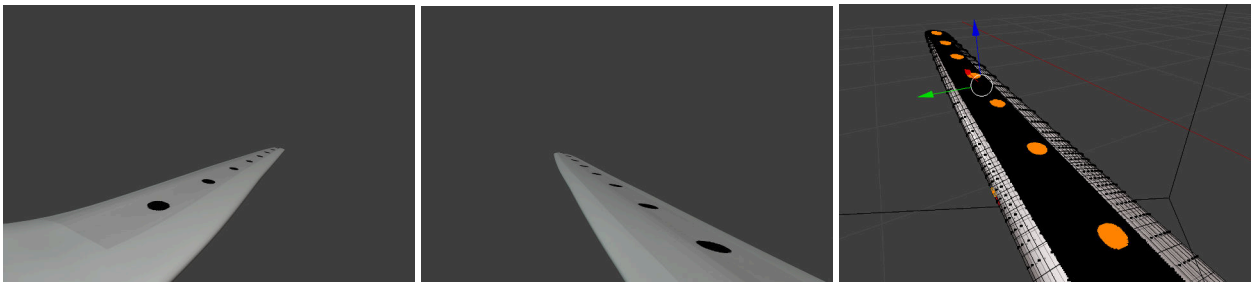


(a) Angled side view **(b) Angled top view**

Figure 6 A multi-view rendering of the virtual (dynamic) environment created in Blender: a flexible glider with the stereo camera setup mounted on the fuselage. The camera matrices are calculated from Blender's internal model and used in the DLT triangulation.

1. Modelling markers

To be able to track the markers in the virtual environment, a blob pattern was embedded onto the wing of the model. When the intrinsic and extrinsic parameters of the virtual cameras are known, the distance between the markers is not relevant for the reconstruction of the 3D location. However, the blobs need to be sized well, such that they are visible from the chosen camera viewpoint. To embed the marker on the wing model, the 2D marker pattern in $(x,\ y)$ space must be mapped onto the 3D mesh surface in $(x,\ y,\ z)$ as shown in Fig. 7c. This is done using the UV-mapping functionality in Blender [22]. The result of the UV-mapping seen from both camera views is shown in Figures 7a and 7b, for cam1 and cam2 respectively.



(a) View camera 1 (cam1) (leading edge) **(b) View camera 2 (trailing edge)** **(c) View mesh and UV mapped pattern**

Figure 7 Two camera views and UV-mapping of the virtual wing.

2. Camera parameter extraction

The intrinsic camera parameters can be extracted from the rendered model in two ways: (i) by calibrating an animated calibration pattern and applying calibration routine by [16] described in Section I.E, or (ii) by extracting the intrinsic parameters directly from Blender's internal animation model. Since the virtual cameras in Blender are internally modelled by user specified camera properties, the latter approach is chosen, as the camera parameters can be acquired accurately and easily from the Blender's internal interface. The camera matrix will contain:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{40}$$

where, c_x, c_y are the principal point in x, y (shift) from image centre. In a pinhole camera model, the focal lengths f_x, f_y represent the distance between the pinhole and the image frame, that for a perfect pinhole camera (no distortion)

have equal value in $x,\ y$. The latter denotes the intersection point of the perpendicular axis from the pinhole to image plane. The required parameters for K are defined as follows:

$$f_x = f_{cam}/w_{sensor} \cdot w \qquad c_x = w \cdot (0.5 - c_{d_x}) \tag{41}$$

$$f_y = f_x \cdot Ar_{pix} \qquad c_y = h \cdot 0.5 + w \cdot c_{d_y} \tag{42}$$

where, $Ar_{pix} = f_x/f_y$, f_{cam} is the focal length in mm, w is the image width, w_{sensor} the width of the camera sensor, and $c_{d_x},\ c_{d_y}$ the shift of principal point in pixels in $x,\ y$. All these parameters can be extracted from Blender internally [22].

The extrinsic camera parameters can be assembled from the intrinsic parameters by extracting the relative orientation of the two cameras with respect to each other and assembling the projection matrices as described in Section I.E.

C. Simulation setup and scheme

The simulation setup consists of an aeroelastic model built in SIMULINK [17] and a visual model built in Blender [21]. Fig. 6 shows the setup of the initial visual model. The aeroelastic model is adopted from [18] and implemented into a real-time model in SIMULINK. The SIMULINK and the Blender models are coupled via a UDP (User Datagram Protocol) data stream. The reconstructed 3D displacements (from the parallel 3D reconstruction pipeline explained in Chapter I), are fed back to SIMULINK using another UDP data stream.

The model is simulated by inducing the wing with a discrete 1-cosine gust at discrete time intervals. In figure 11, the response of the model is shown to input gust excitations. The Blender animation is updated at 60 HZ with the newest data obtained from the SIMULINK model running at 50 HZ.

1. Distributed frame and data streaming

To enable synchronized, fast and robust data streaming a Redis [23] network server is setup. Redis is a novel communication protocol designed to deliver low latency data stream on localhost (locally on one computer) across multiple processes, but also across multiple computers and processes connected to the same server cluster [23, 24]. Redis protocol uses an in-memory (RAM) data transfer to maximize efficiency, meaning that a chunk of memory is allocated such that two processes can read and write efficiently without using the (usually slower) interface hard disc.

Efficiently processing stream data is a key part in the framework as the tracking process need to be detached from the visual model via a robust communication interface without additional delays. Redis databases are also starting to play an important role in distributed learning an deployment of model free Reinforcement Learning (RL) [25] in several studies [26, 27]. Due to the computational load and iterations required for deep reinforcement learning based control approaches, Redis lends itself very well for sharing distributed data through asynchronous controller updates [28].

While Redis just provides the mechanism and the protocol for sharing data, a key step in piping data through a Redis server is to pack and unpack the data in the correct manner on the sending and receiving side. A flexible way to do this is through a buffer of bytes. The data can be of any type (e.g double, float, int), or a container holding the data type (e.g. vector, matrix, array) can be converted to a stream of bytes. In the framework described in this study, the data is a 3-channel matrix with integer ($uint8$) RBG values in the range $(0 - 255)$, and can be cast into a continuous stream of long buffer. The procedure used to cast and unpack the data is described in Algorithm 2. In the current pipeline, the Redis server communication was implemented using the C implementation of the protocol by $hredis$ library and the C++ wrapper $redis - plus - plus$ [29].

Algorithm 2: Redis send and receive implementations for image data. On the receiver side, the endianness (byte encoding order) must be taken into account while unpacking

```
Function main(args, opts):
    Initialize containers;
    redisObj.Redis(host='localhost', port=6379, db=0) // connect to Redis server
    while loopRunning do
        Mat Img = receiveRedisImg(redisObj,redisId);
        /* do work on image */
        sendRedisImg(redisObj,redisId,Img);
    end
end;
Function sendRedisImg(redisObj, redisId, imgdata):
    Initialize buffer, get imgdata;
    /* mempcy(dst,src) from image or imgdata.tobytes() */
    memcpy(buffer.data,imgdata.data,imgdata.size);
    /* send to redis server using a unique redisId identifier */
    redisObj.set(redisId,buffer.data);
end;
Function receiveRedisImg(redisObj, redisId):
    Initialize buffer and empty imgdata container;
    /* mempcy(dst,src) unpack to image or imagedata.frombuffer() */
    memcpy(imgdata.data,buffer.data,buffer.size);
    redisObj.get(redisId,buffer.data);
end;
```


III. Results and Discussion

The methodologies and framework discussed in the previous section were subject to tests on live image sequences from recorded and generated data. The results of the parallel feature extraction and tracking pipeline are presented in this section.

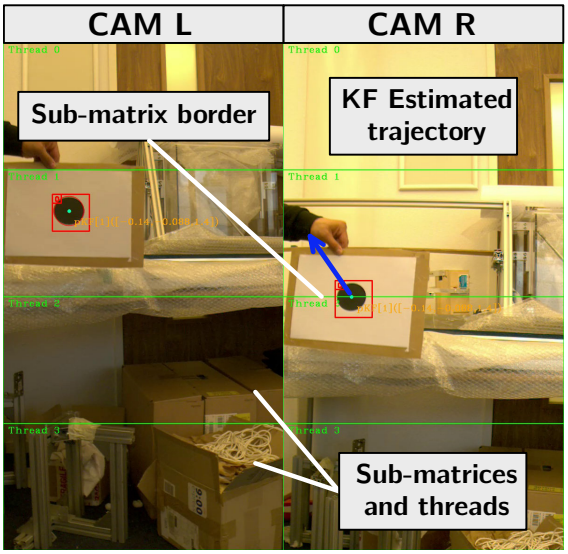
Code development, testing, and assessment of the method was done using standard Dell Optiplex 7400 and 2.3 GHz Intel Core i5 16G MacBook. For motion capture and the Basler C++, Pylon API [30] was used. For the development of the methodology and analysis, OpenCV open-source computer vision library [8] and C++11 STL (Standard Library Extensions were used) [31]. The image and tracking data were extracted and plotted in using the OpenCV-Matlab parsing interface tmkhoyan/cvyamlParser [32]. The code, dataset and tolls developed are available under tmkhoyan/adaptiveClusteringTracker [33]. The results were focussed on the effects and benefit of adding a Kalman filter in the parallel tracking routine and show analysis on 136 images sequence captured in a laboratory environment. Furthermore, an initial assessment of the smart sensing control framework is discussed.

A. Experimental Set-up and Data Collection

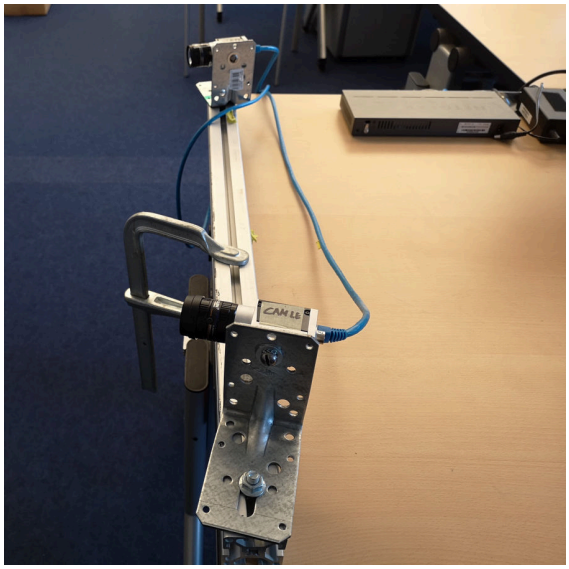
The parallel 3D reconstruction pipeline was tested on live images obtained from two Basler GigE Ethernet cameras and lenses [30, 34]. The cameras were calibrated beforehand using the stereo calibration process, as described in the study in section I.E [16]. The motion capture setup is shown in Fig. 8 and the results from the parallel pipeline are shown in Fig. 8a, for the image pairs received from camera 1 and 2 respectively. Green regions $(1-4)$ correspond to the 4 sub-matrices and parallel threads. The number of threads and sub-matrices is a direct input parameter in the parallel tracking pipeline.

B. Border transition prediction with Kalman filtering

In the description of the building blocks of the parallel tracking pipeline, the relevance of the Kalman filter was pointed out. It was proposed to use the Kalman filter to reduce the frequency of keypoint matching and avoid computations of full image matrices. Aside from this, the Kalman filter can help overcome a limitation related to parallel computation on sub-matrices. Namely, when a large keypoint (object of interest) travels from one sub-matrix domain to the other. For small particles traveling in and out of sub-matrix borders, this is not necessarily a problem.



(a) Experimental testing of the parallel 3D reconstruction pipeline. The left and right images correspond to camera 1 and 2 respectively. The images shown the Kalman filtering process when the keypoints switches sub-matrix domain (border transition)



(b) Close up of the stereo camera setup

Figure 8 Setup and the motion capture system of the parallel 3D reconstruction pipeline in the laboratory test-setup environment.

However, a large particle may spend a relatively long time during the transition. The problem that arises is that during this interval, no tracking and thus no correspondence is available from two image pairs. In this case, a simple linear Kalman filter provides a prediction of the trajectory of the particle and ensures that the points correspondence and thus the reconstruction is still valid during the transition. The result of this process is shown in figure 8a. Here, it is observed that the right keypoint pair situated precisely on the border of the sub-matrix. As the Kalman Filter contains information about the particle's previous velocity and position, it can draw an estimated trajectory quite well. This prediction can cover the gap until the keypoints enter the neighboring sub-matrix, and the measurement from feature extraction is online again.

C. Reduction of keypoint extraction and matching frequency with Kalman filtering

To investigate the effect of the Kalman filtering on tracking performance with a reduced sampling frequency of the keypoint extraction and matching processes tracking of an input image sequence pair of 136 images was performed. The results are presented in Fig. 9 for image instances (11,30,60,120). These instances were chosen as the most interesting points in the Kalman filter prediction trajectory. This particular motion sequence belonged to the image stream of the left camera and was selected as this sequence featured acceleration and deceleration of a blob in motion. In this configuration, it was of particular interest to see, how the how well a Kalman filter with linear dynamics was able to cope with non-linear motion. When a measurement is continuously available, and the intervals of measurement loss are small, a simple linear dynamics should suffice in most motion conditions. In this experiment, the frequency of measurement feedback and Kalman model update was reduced in steps of 2,4,8 and 10, and the Kalman filter was required to produce longer predictions gradually, thereby relying on fewer measurement updates (i.e., only 13 updates in the last condition with 136 frames). These numbers, are referred to as experiment conditions F2-F10, correspond to the number of consecutive frames skipped in the keypoint matching, and thus measurement updates. The rows of the image sequence shown in 9, correspond to the experimental conditions. The processing pipeline is shown in the bottom row. In the last column, the reconstructed X, Y, Z coordinates, and the trajectory is shown. For these two image pairs are used, from two Kalman filter predictions, and the DLT method is applied [15]. The red line is the Kalman Filter prediction for all the steps, and the black dots are intervals of available measurements.

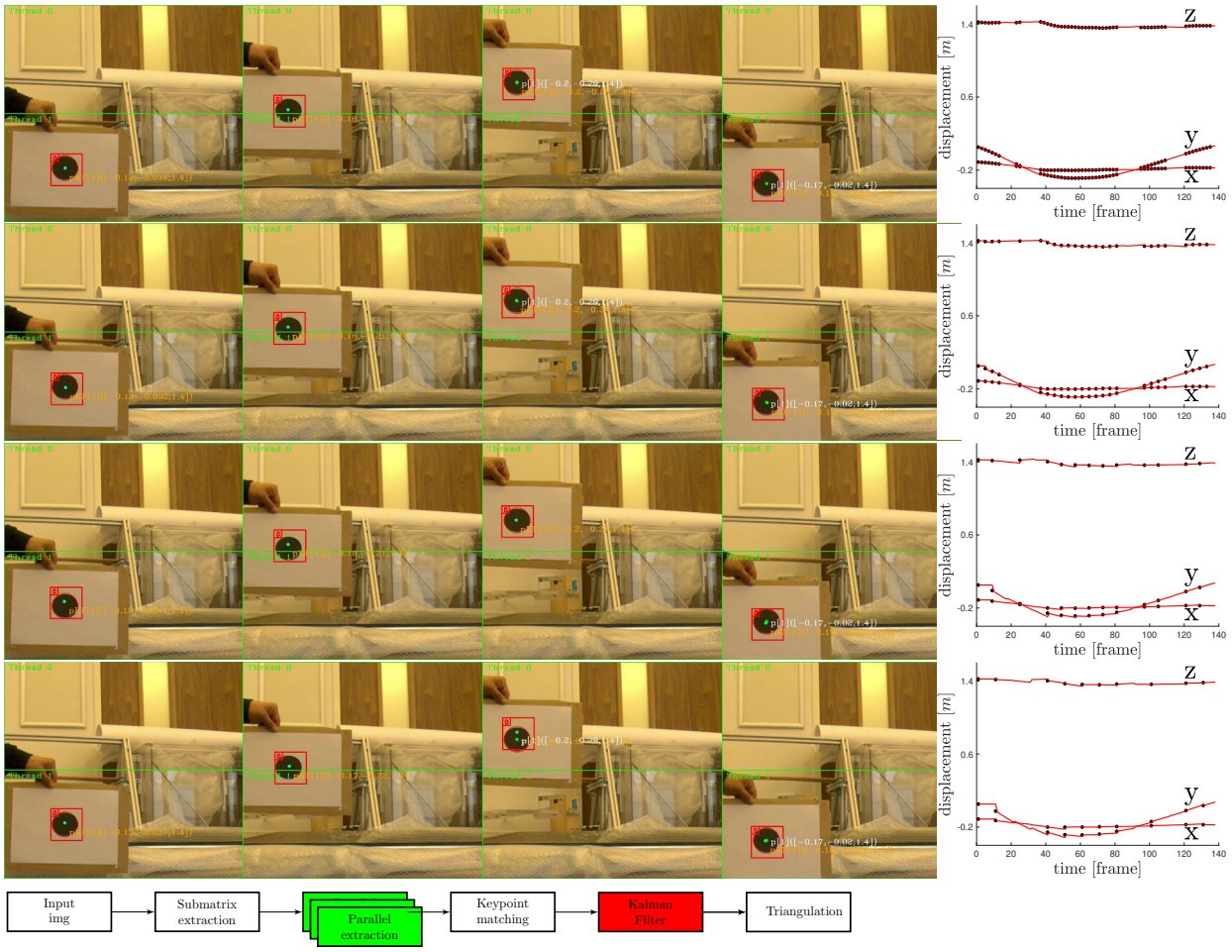
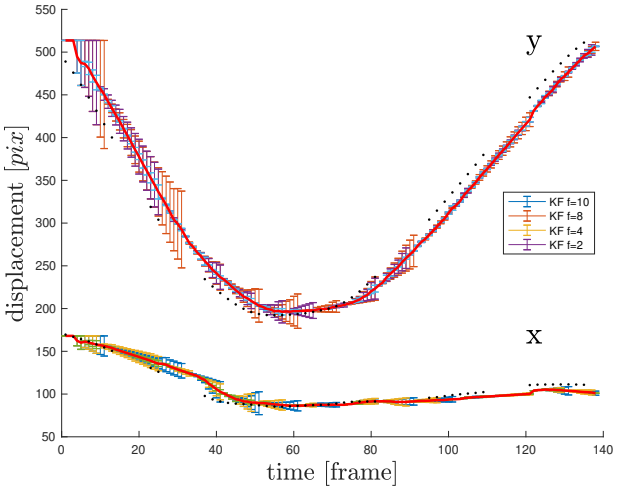
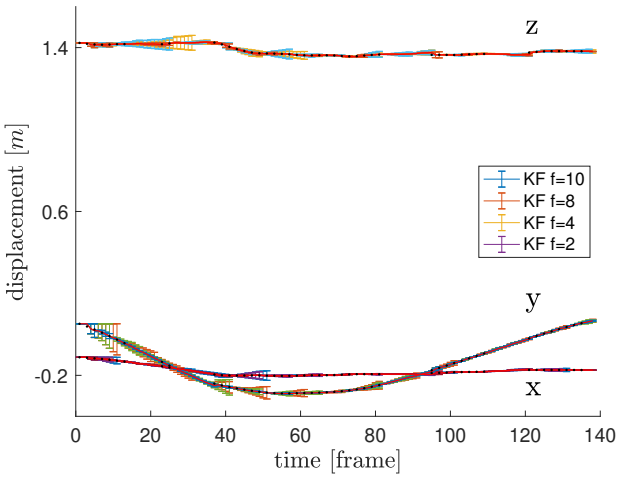


Figure 9 Tracking sequence on input images from left camera (11,30,60,120) for experimental conditions F2-F10 with longer Kalman prediction intervals and reduced frequency of keypoint matching and extraction. Rows 1-4 correspond to the conditions F2,F4,F8 and F10, the last column shows the reconstructed motion in X, Y, Z from Kalman predictions of the two image pairs and the available measurements (black dots).

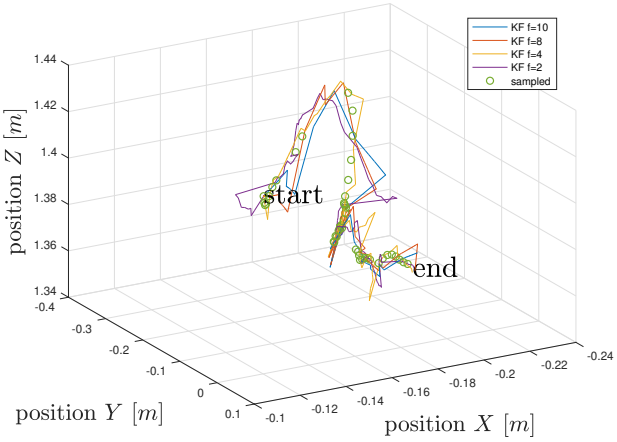
As shown from the figures, the Kalman filter has the best prediction for the lowest interval. The performance gradually worsens as the intervals decrease; however, the Kalman filter can predict the trajectory of motion reasonably well. As a significant benefit, if a particular error is acceptable in the tracking, and the follow-up motion reconstruction, the computational load in the Kalman prediction interval, can be significantly reduced. The computational load is directly related to the frequency multiplier of the experiment condition; for F2, the number of frames to process is reduced by two, F4 by 4, and so forth for the remaining conditions. In these skipped intervals, no expensive computations are required for extraction, descriptor training, and keypoint matching, and thus, the computational load is reduced. Furthermore, from the figures, it can be observed that the benefit of the Kalman filter is two-fold, namely that it provides a prediction of the trajectory during the border transition of the keypoint. The Kalman filter model is its current set-up, with state vectors $\mathbf{x}, \mathbf{x}, bvecny, \mathbf{y}$, (position and velocity in x, y) the model is nearly model free. To further increase the robustness of the Kalman filter, more elaborate models can be included; however, this introduces more model dependency and model knowledge requirement. To relax the model dependency, and provide predictions for more complex dynamics, an augmented Kalman filter form can be formulated, where the model has time-varying parameters included as explicit states or modeled drivers. This has been applied in several studies [35, 36].



(a) Errorbar plot of the Kalman filter prediction trajectories in x, y image pixel coordinates for F4-10 relative to the best case, F2, observed for left camera.



(b) Errorbar plot of the reconstructed trajectories in X, Y, Z from the Kalman filter prediction for F4-10 relative to the best case, F2.



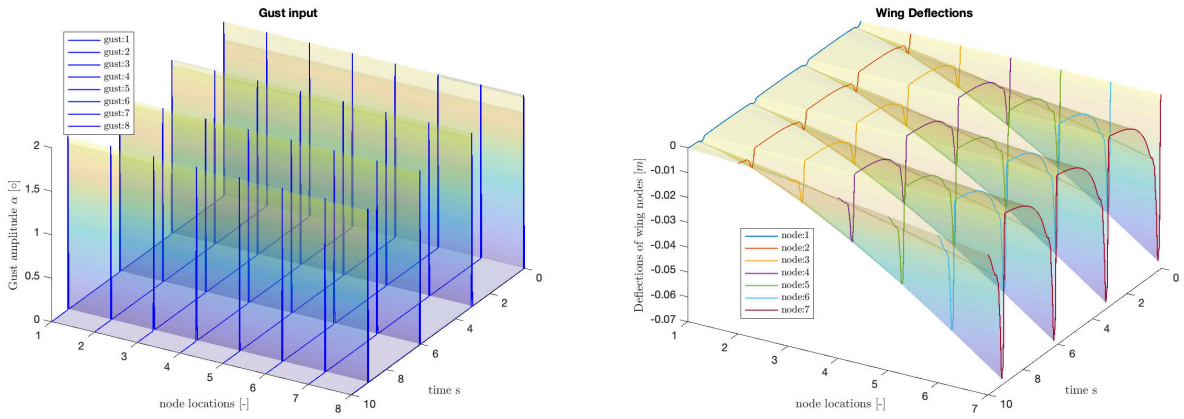
(c) Reconstructed 3D trajectory from Kalman filter prediction for F2-F10. Scattered points are the measurement updates in F2.

Figure 10 Trajectory reconstruction of the Kalman filter predictions for F2-F10

D. Reconstruction of motion trajectory

The motion trajectory was reconstructed using the DLT method explained in section I.E by Hartley et al. [15]. For DLT, the Kalman prediction from two image pairs was combined, and the correspondence of points was established. For the current case, this is a relatively straight forward approach, as a single blob was observed in the input; however, in the parallel tracking pipeline, a descriptor computation is performed based on geometrical properties (circularity, area) and the keypoint distances. The result of the reconstruction is shown in the 3D view in Figure 10. The motion trajectory for all conditions is shown; the scattered points represent the measurement intervals for F2 (best case). The F10 has the worst prediction again but still agrees with the trajectory despite only 13 measurement updates.

In Fig. An error bar plot is shown 10b of the trajectories in X, Y, Z for Kalman filter predictions F4-10 relative to the best case, F2. F10 shows the highest error with a delta of 0.024 m again. The latter is the extreme corner case and corresponds to the point when no measurement is available, and the tracked object shows-high non-linearity. The error can also be viewed in the image plane in x, y pixel coordinates streamed from the left camera. A similar errorbar, for the trajectory of F4-F10 concerning F10 in pixel x, y coordinates is shown in Figure 10a, the error bar plots are shown. Here we see a maximum delta of 56 pixels. This high error is mainly constrained to the areas of the trajectory where the non-linearity is the highest, and the tracked feature abruptly changes direction. Naturally, with only minimal



(a) Uniform gust input generated at discrete time intervals (3 s).

(b) Deflections of the aeroelastic model (wing) under excitation of 1-cos gust. The nodes 1-7 span-wise location of 7 nodes aligned at the actuator span-wise location

Figure 11 The input and output response of the aeroelastic model for the duration of 10 seconds.

measurement updates, the linear predictions cannot realistically stretch the gap of measurement in the interval.

E. Initial assessment of the Control and Smart Sensing Framework

The aeroelastic model as described in section II.A was subjected to discrete gust signals , show in figure 11a. As seen in figure 11b, large wing deflection (up to 70 mm) is observed for this very flexible model. The model, therefore, lends itself very useful for assessing the tracking routine by visual feedback. In the current model, an LQR controller is implemented. In closed-loop, the controller limits the deflections and the root bending moment — further details regarding the design and the analysis id given in [18].

Follow-up studies are planned to investigate different types of controllers and further asses the feasibility and benefits of the virtual framework with visual feedback. The critical aspect is the proper delegation of communication in an efficient and distributed fashion via the Redis server.

IV. Conclusion and Recommendation

In this study, a parallel tracking pipeline was constructed with TBB (Threading Building Blocks TBB) for efficient implementation of feature extraction needed for feature matching and 3D reconstruction. The parallel keypoint extraction was capable of significantly speeding up the extraction process by parallelizing the relatively heavy computational tasks across multiple threads, divided over smaller sub-matrices. The approach was tested in real-time on image sequences of $1088 \times 600 \times 3$.

The complete tracking pipeline was implemented with a blob detector and descriptor computation based on geometrical properties of keypoints, followed by the DLT (Direct Linear Transform) method for the reconstruction of the 3D position from the correspondence of two points. Furthermore, a Kalman filter was implemented in order to reduce the feature matching intervals. The intervals were reduced for numbers of 2, 4, 8, and 10 of frames skipped consecutively. It was shown that the Kalman filter implementation provided good robustness for the prediction of position and subsequent reconstruction of the motion trajectory under the absence of measurement updates from the feature extractor. It was also observed that the Kalman filter was able to provide the tracking robustness by position prediction during the transition from one sub-matrix domain to the other. This transition happens when a keypoint enters a domain computed by another thread than previously. In particular, for large single keypoints, it showed to be a relevant strategy for robust reconstruction of motion trajectory.

A smart sensing flexible framework was proposed for the reconstruction of displacements from a visual blender model driven by a numerical model in SIMULINK. It was discussed how the required intrinsic and extrinsic camera parameters coupled be extracted for the reconstruction of the wing displacements. Finally, the importance of distributed

fast communication was emphasized, and mainly how this could play a key role, not only for the development of the smart sensing framework but also in the assessment of the novel visual-based intelligent feedback control methods. Further studies are planned to further asses the framework and investigate more elaborate schemes with the parallel tracking pipeline.

References

[1] Burner, A. W. and Liu, T., "Videogrammetric model deformation measurement technique," *Journal of Aircraft*, Vol. 38, No. 4, 2001, pp. 745–754.

[2] Corke, P. I., "Visual control of robot manipulators–a review," *Visual Servoing: Real-Time Control of Robot Manipulators Based on Visual Sensory Feedback*, World Scientific, 1993, pp. 1–31.

[3] Wang, X., "Intelligent multi-camera video surveillance: A review," *Pattern recognition letters*, Vol. 34, No. 1, 2013, pp. 3–19.

[4] Belbachir, A. N., *Smart cameras*, Vol. 2, Springer, 2010.

[5] Otsu, N., "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, Vol. 9, No. 1, 1979, pp. 62–66.

[6] Serra, J., *Image analysis and mathematical morphology*, Academic Press, Inc., 1983.

[7] Suzuki, S. et al., "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, Vol. 30, No. 1, 1985, pp. 32–46.

[8] Bradski, G., "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[9] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. R., "ORB: An efficient alternative to SIFT or SURF." *ICCV*, Vol. 11, Citeseer, 2011, p. 2.

[10] Lowe, D. G., "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, Vol. 60, No. 2, 2004, pp. 91–110.

[11] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L., "Speeded-up robust features (SURF)," *Computer vision and image understanding*, Vol. 110, No. 3, 2008, pp. 346–359.

[12] Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D., "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics*, Vol. 31, No. 5, 2015, pp. 1147–1163.

[13] Miksik, O. and Mikolajczyk, K., "Evaluation of local detectors and descriptors for fast feature matching," *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, IEEE, 2012, pp. 2681–2684.

[14] Reinders, J., *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*, " O'Reilly Media, Inc.", 2007.

[15] Hartley, R. and Zisserman, A., *Multiple view geometry in computer vision*, Cambridge university press, 2003.

[16] Zhang, Z., "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, Vol. 22, 2000.

[17] MathWorks, "Simulink - Simulation and Model-Based Design - MATLAB & Simulink," .

[18] De Breuker, R., Binder, S., and Wildschek, A., "Combined Active and Passive Loads Alleviation through Aeroelastic Tailoring and Control Surface/Control System Optimization," 2018.

[19] Theodore Theodorsen, "General theory of aerodynamic instability and the mechanism of flutter," .

[20] Leishman, J. G., "Subsonic unsteady aerodynamics caused by gusts using the indicial method," *Journal of Aircraft*, Vol. 33, No. 5, sep 1996, pp. 869–879.

[21] Blender.org, "blender.org - Home of the Blender project - Free and Open 3D Creation Software," .

[22] Blender, "Blender Documentation Contents — Blender 2.79.0 855d2955c49 - API documentation," .

[23] Redis, "Redis," .

[24] Carlson, J. L., *Redis in action*, Manning Shelter Island, 2013.

[25] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D., "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, No. 7540, feb 2015, pp. 529–533.

[26] Nishihara, R., Moritz, P., Wang, S., Tumanov, A., Paul, W., Schleier-Smith, J., Liaw, R., Niknami, M., Jordan, M. I., and Stoica, I., "Real-Time Machine Learning: The Missing Pieces," *Proceedings of the Workshop on Hot Topics in Operating Systems - HOTOS*, Vol. Part F1293, IEEE Computer Society, may 2017, pp. 106–110.

[27] Li, T., Xu, Z., Tang, J., and Wang, Y., "Model-free control for distributed stream data processing using deep reinforcement learning," *Proceedings of the VLDB Endowment*, Vol. 11, No. 6, 2018, pp. 705–718.

[28] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K., "Asynchronous methods for deep reinforcement learning," 2016, pp. 1928–1937.

[29] Senewnew, "sewenew/redis-plus-plus: Redis client written in C++," .

[30] Basler, "Basler ace acA1300-75gc - Area Scan Camera," .

[31] ISOCCP, "Current Status : Standard C++," .

[32] Mkhoyan, T., "tmkhoyan/cvyamlParser: Initial public release," .

[33] Mkhoyan, T., "tmkhoyan/parallelTrackingTBB: initial release," .

[34] Basler, "Basler Lens C125-0618-5M F1.8 f6mm - Lenses," .

[35] Azam, S. E., Chatzi, E., and Papadimitriou, C., "A dual Kalman filter approach for state estimation via output-only acceleration measurements," *Mechanical Systems and Signal Processing*, Vol. 60, 2015, pp. 866–886.

[36] Lourens, E., Reynders, E., De Roeck, G., Degrande, G., and Lombaert, G., "An augmented Kalman filter for force identification in structural dynamics," *Mechanical Systems and Signal Processing*, Vol. 27, 2012, pp. 446–460.

Appendix A. Blob parameters

Table 1 Parameters of the blob detector. Maximum entries without defined value mean that maximum value of the parameter is not considered as a constraint. Internally, the detector will still produce valid shapes as the input contours are sorted based on [7].

Blob parameter	*Value [pix]*
max threshold	10
min threshold	300
min Area	200
max Area	-
min Circularity	0.1
max Circularity	-
min Convexity	0.87
max Convexity	-

Appendix B. Algorithms summary

The algorithms presented in the methodology section are summarized in this section. The main body of the algorithm can be summarized in 3. The functions definitions used in the main body are summarized in 4. The main body of the parallel process and the steps performed on the submatrices is shown in 1.

Methods are investigated to redistribute the sub-matrices in a more optimal arrangement based on keypoint properties and distribution across the image.

Algorithm 3: Main pipeline body

Input: $Img\_in, Img\_out, keypoints, subMatYlen, range$
Output: $p\_triangulated$
/* Initialize blobDetector() and reserve containers: */
Initialize containers,camera parameters($camParams$),Redis connection(redisObj);
/* main body of parallel pipeline */
Function `main`($args, opts$)**:**
  while *loopRunning* **do**
    /* receive images from redis server */
    grabRedisImage(keyIdL,&redis);
    grabRedisImage(keyIdR,&redis);
    /* spawn Nthr parallel threads to compute submatrices */
    parallel_for(Range(0,$Nthr$), ParDetector(img_L,$keypointsL$,0, $Nthr$));
    parallel_for(Range(0, $Nthr$), ParDetector(img_R,$keypointsR$,0, $Nthr$));
    /* perform matching, populate $pxyL$, $pxyR$ and triangulate */
    Npts = matchDetectedPoints(keypointsL,keypointsR,pxyLvec,pxyRvec);
    for $k = 0$; *while* $k < Npts$; k++ **do**
      triangulatePoints($pxyL[k], pxyL[k], camParams$);
    end
  end
  return 0 ;

Algorithm 4: Main pipeline function definitions.

Function `triangulatePoints`($pxyL, pxyR, camParams$)**:**
  /* remove lens imperfections */
  undistortPoints($pxyL, pxyL\_unidst, camParams.K1, camParams.D1$);
  undistortPoints($pxyR, pxyR\_unidst, camParams.K2, camParams.D2$);
  /* DLT SVD operation */
  applyDLT($camParams.P1, camParams.P2, pxyL\_unidst, pxyR\_unidst, p4D$);
  convertPointsFromHomogeneous($P4D.reshape(4, 1), pxy3D$);
  return $pxy3D$;
end;
Function `grabRedisImage`($redisId, redisObj$)**:**
  /* redisObj and buffer are accessed via pointer (e.g. buffer→data) */
  buffer = redisObj.get(redisId);
  memcpy(img.data,buffer.data,buffer.size);
  return img;;
end;