Fast High-Resolution Disparity Estimation for Laparoscopic Surgery

Jan Müller*, Reuben Docea†, Matthias Hardner‡, Katja Krug§, Paul Riedel¶, Ronald Tetzlaff*

*Institute of Fundamentals of Electrical Engineering, Technische Universität Dresden, Germany
†Translational Surgical Oncology, National Center for Tumor Diseases (NCT), Partner Site Dresden, Germany
‡Institute of Photogrammetry and Remote Sensing, Technische Universität Dresden, Germany
§Institute of Multimedia Technology, Technische Universität Dresden, Germany
¶Institute of Software and Multimedia Technology, Technische Universität Dresden, Germany
(jan.mueller, matthias.hardner, katja.krug, paul.riedel, ronald.tetzlaff)@tu-dresden.de, reuben.docea@nct-dresden.de

***Abstract*—An intraoperative Image Guidance System (IGS), facilitating the localisation of pathological tissue or vasculature, could greatly support medical decisions during minimally invasive interventions. In our IGS for laparoscopic surgery, the 3D reconstruction of abdominal organs requires fast and accurate depth information from stereo images. To this end, we employ a state-of-the-art algorithm for dense disparity estimation. To cope with low processing performance, previous solutions used only downscaled images, and hence produced disparities of low quality. In this work, we present methods and implementations which improve and accelerate disparity estimation such that it runs with FullHD resolution images at full camera framerate.**

***Index Terms*—Image guided surgery, machine vision, stereo disparity, neural network, parallelisation, multithreading**

I. Introduction

In recent years, abdominal surgery has moved increasingly from open surgery to Minimally Invasive Surgery (MIS) methods, using instruments and an endoscope inserted through a few small cuts in the abdomen (*laparoscopy*). This is motivated by the lesser degree of morbidity and faster recovery of the patient [1]. However, these methods introduce new challenges, such as a limited field of view, the inability to palpate organs for tumours, and the need for good hand-eye coordination [2].

To overcome these difficulties, considerable effort has been invested in the development of Image Guidance Systems (IGS), which highlight to the surgeon the positions of structures of interest such as tumours or vasculature [3]. This is often complemented with Augmented Reality (AR) by superimposing target structures over the organs (see Fig. 1).

Our IGS/AR system for liver laparoscopy [4] relies on the creation of an intraoperative 3D reconstruction of the liver during the operation, to which a preoperatively acquired computed tomography (CT) scan is registered. This 3D reconstruction continuously integrates multiple point clouds, incorporating sets of the following data: (i) a pose derived from a simultaneous localisation and mapping (SLAM) method [5], [6], and (ii) a dense point cloud created through disparity estimation, which includes RGB color information from the laparoscope (see Fig. 2). The rate of formation of these fused point clouds is therefore constrained by the frequency of availability of these data. Furthermore, the quality of the data substantially impacts the resulting 3D reconstruction.

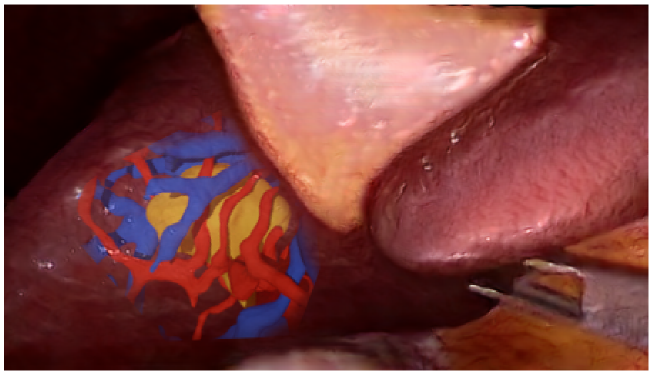
Fig. 1. Augmented Reality visualisation of tumour (yellow) and vasculature (blue & red) over liver

The disparity estimator we chose is the Hierarchical Stereo Matching (HSM) network [7], as it delivers high-quality *dense* disparity similar to state-of-the-art methods, while being faster than comparable approaches. For the initial versions of our system [8], we used inputs and outputs of reduced size (Quarter HD). However, to achieve an acceptable throughput, the resolution was downscaled even further, such that the network processed only 1/16th (480×270 px) of the endoscope's FullHD resolution. This resulted in a significantly deteriorated quality of the depth information.

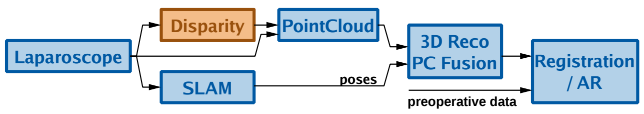

Fig. 2. Workflow (simplified) of IGS/AR system for liver laparoscopy.

The authors gratefully acknowledge funding for this research by the State of Saxony via Schsische Aufbaubank (SAB) in the scope of the ARAILIS project (100400076). This measure is co-financed with tax funds on the basis of the budget passed by the Saxon state parliament.

In this work, we address the *rate and quality* of disparity estimation in stereo laparoscopic imaging, and provide a means of obtaining more accurate disparity estimation at higher framerate. In accordance with the hardware conditions, our *goal* is to process stereo images and produce disparities with FullHD resolution (1920×1080 px) at the full endoscope framerate (30 frames/s).

In the following section, we describe the methods and techniques that were applied to reach this goal. These are the result of an extensive investigation of the considered algorithm, system, and suitable libraries and tools. Solutions and parameters specific to our system setup will then be discussed in Section III. This is followed by a presentation of representative results of quality and speed measurements.

II. Algorithm and Methods

A. Algorithm and General Optimisations

A rough outline of the disparity estimation algorithm is shown in the block diagram in Fig. 3a. The core of the algorithm is the *inference* of the trained HSM network [7]. This neural network searches for stereo correspondences in a coarse-to-fine hierarchy. Coarse resolution is used to estimate large disparities, which are then employed to bias fine-scale disparity estimates. In the *optimisation*, only a few modifications, which do not change the network's structure, are feasible. This allows us to preserve the provided weights, having been tuned on a large number of training samples.

The initial step is the *pre-processing* of the input images. The algorithm receives a pair of rectified [9], [10] RGB stereo images as input, which are converted to a suitable tensor data format. This is complemented by the customary normalisation of neural network input data (also *standardisation* [11]), and a transposition and padding of the images to the expected shape.

After network inference, in the *post-processing* step, the resulting one-dimensional array of disparity values is transformed back to a 2D shape of the original resolution.

Direct optimisation measures include slimming loops (by extracting instructions from them), removal of redundant code, and replacement of unnecessary data copy operations. The substitution of costly operations required a comparison of different libraries to find the most efficient implementations.

B. Inference and Pre-processing Using TensorRT and DALI

As expected, the network *inference* proved to be the most time-consuming part. In order to accelerate the inference while maintaining the network structure and weights, we decided to develop an implementation based on NVIDIA TensorRT (TRT) [12]. This library and tool set for high performance inference on GPUs optimises the inference using the CUDA parallel programming model [13]. The internal data layers can be processed using several (reduced) data types: 32-bit floating-point (FP32), 16-bit floating-point (FP16), 8-bit integer (INT8), and combinations thereof (*mixed-precision*). This data reduction helps to reduce the memory footprint and transport delays while increasing the processing speed, but may come at the cost of reduced accuracy of the result.

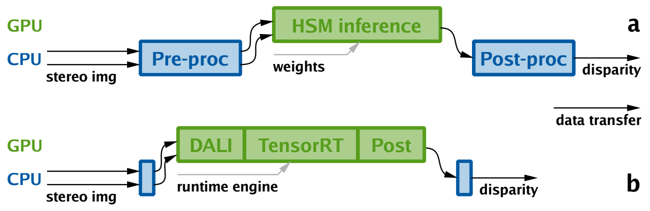

Fig. 3. Block diagram of disparity algorithm, a) original, b) with TensortRT and DALI acceleration; processing lengths are symbolic.

The use of TRT first required an adaptation of the network to TRT specifications (see Section III). We then investigated the impact of various parameters on the resulting runtime engine. Especially in the selection of data types and precision, we had to balance disparity quality against speed and efficiency.

With a reduced processing time of the inference, the share of the *pre-processing* step becomes more relevant. We designed a GPU-centered pipelining approach for maximum throughput in this step, using the NVIDIA Data Loading Library (DALI) [14], [15]. To this end, we investigated available DALI operators and their parameters to represent the original functionality in the most accurate and, ideally, fastest way.

The introduction of TRT and DALI requires an adaptation of the remaining algorithm to optimally feed the pipeline and TRT engine. In connecting the building blocks, we had to modify the recommended structure such that most operations are executed on the GPU, and data transfers between CPU and GPU are minimised, as sketched in Fig. 3b.

C. Multithreading

The methods described above exploit fine-grained parallelism and pipelining to process one stereo image pair at a time. They utilise only one processor core (single thread) and are, moreover, designed for use on a single GPU. While the methods render very good timing results (see Section IV), they are still not sufficient to reach the formulated goal.

We therefore designed a multithreaded processing scheme that allows a parallel execution of several CPU threads and the usage of multiple GPUs (and combinations thereof). This scheme is applicable to the disparity algorithm since consecutive images have no data dependence on each other. We developed various approaches using synchronisation and threading mechanisms of the underlying system – due to space restrictions only the most efficient solution is presented here.

At the first stage the *sync module* synchronises associated stereo images and passes them on as pairs, with a special tagging dependent on the envisaged number of parallel threads. We are then able to distribute the disparity processing to different execution threads and to assign these to distinct GPUs (see Fig. 4). An even distribution of the load to threads is intended, assuming an approximately even temporal distribution of incoming image pairs. However, other distribution schemes are possible within our approach, such as adapting to heterogeneous CPU cores or GPUs.

III. Implementation

A. ROS Integration and Interfacing

The IGS system described in the introduction is built using the ROS (Robot Operating System [16]) framework. This framework facilitates modularisation and provides many standard image processing modules. On the other hand, it introduces overhead that makes adaptation of the code necessary.

On launch, the disparity module is initialised (especially TRT engine and DALI pipeline). The processing itself is executed in a callback function which has to be registered at the beginning. The main control loop runs in the ROS kernel and invokes this function when input data are available. In the callback, our disparity module extracts information on the received input images from the ROS data structure. After post-processing of the inference results, an output data structure is created and filled with disparity information.

B. Restrictions for TensorRT

When implementing the core element of our algorithm, the network inference, the original *PyTorch* implementation has to be adapted to the system requirements while preserving the trained network weights. Since TensorRT (TRT) cannot work directly on PyTorch networks, the network has to be exported to the Open Neural Network Exchange (ONNX) format [17], which is then processed to build the TRT engine. However, the set of operations for both these steps is restricted, making an adaptation of the neural network code inevitable.

The TRT engines were built for combinations of the available data types. In addition, several build parameters were varied, leading to a large number of engine versions that had to be compared with regard to quality and speed (in Section IV).

C. Pipeline Technique for DALI

The DALI library has originally been designed for pipelined file processing. To utilise a DALI pipeline in ROS callbacks we had to redefine the interface, now feeding one input image pair into the pipeline upon each callback. In reference implementations, the TRT data are copied to/from the GPU using streaming transfers. We modified the TRT implementation such that the TRT engine operates directly on results of the DALI pipeline *in GPU memory*, as indicated in Fig. 3b. Similarly, the result of the TRT inference is kept in GPU memory as well, enabling faster post-processing.

D. Multithreading Using ROS Threading Support

The *sync* module described in Section II synchronises incoming stereo images by means of their creation time stamps [18]. For the thread tagging, a thread number $t = c \bmod N_{\mathrm{T}}$ (c - callback counter, N_{T} - number of threads) is determined, and is appended to the identifiers of the stereo image pairs. In the ROS run scripts, disparity modules are then launched in separate threads, processing only specifically tagged image pairs. For an explicit Multi-GPU implementation the threads can be assigned to the respective GPU identifiers. This scheme, as outlined in Fig. 4, allows a very finely tuned exploitation of the available CPU and GPU computing resources.

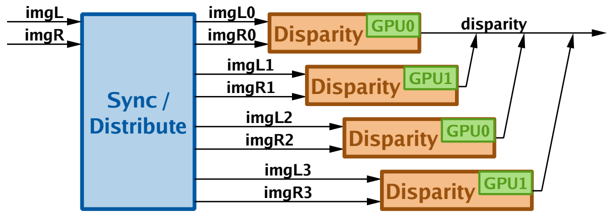

Fig. 4. Synchronisation/distribution and multithreaded execution, with 4 threads on 2 GPUs and symbolic time-shifted execution in shown example.

IV. Experimental Results

Many experiments were conducted during the development of the implementation variants - concerning the ONNX export parameters, TRT engine build parameters, data types, etc. Throughout the data type models, the parameter combinations with the optimum result regarding quality *and* throughput were selected. We hence present here typical results for these models that will also help to select the best fitting model for a specific setup and task. Since the processing speed is independent of the input data (only dependent on the model), the timing results will be considered seperately.

Our *reference* model here (**REF**) is a ROS module with the original "serial" HSM algorithm processing FullHD (1920×1080 px) data at the inputs, the output, and in the network. We compare this reference to the following models:

- **DS**: "Serial" model, data *downscaled* to $1/16$ resolution
- **32**: TensorRT engine & DALI pipeline, FP32 data only
- **16**: TRT&DALI, mixed-precision data FP16 & FP32
- **816**: TRT&DALI, mixed-precision INT8, FP16 & FP32
- **8**: TRT&DALI, mixed-precision INT8 & FP32.

All implementations, except **DS**, work with FullHD data throughout the module.

A. Quality results

Here we compare results of the disparity outputs for different implementation models to those of the reference model (index ref). As the key measure, we selected the relative disparity error $e_d = |(d - d_{\mathrm{ref}})/d_{\mathrm{ref}}|$, with d - disparity. Since $d = fB/Z$ (with Z - depth, f - focal length and B - stereo baseline), it is very similar to the relative error in depth $e_Z = |(Z - Z_{\mathrm{ref}})/Z_{\mathrm{ref}}| = |(d - d_{\mathrm{ref}})/d|$, but avoids large singular errors (outliers) due to small erroneous disparities.

We calculate e_d at every pixel location, and from that the *mean* of e_d over the whole image $\overline{e}_d$, and the *maximum* of e_d in each image $\hat{e}_d$. In addition we restrict the error calculation in each image to the region of interest, defined by the depth of field (DOF) of the endoscope cameras. This limits the observations to the range of depths over which point cloud data will be used for the 3D reconstruction.

In Fig. 5 the mean relative disparity errors in the entire image and in the DOF, over the whole data set, are displayed for the above-named models in a boxplot. In Tab. I the errors $\overline{e}_d$ and $\hat{e}_d$, averaged over the whole data set, are listed for the same implementations. We selected a typical data set of 926 stereo images, with a disparity range of [3.24, 560].

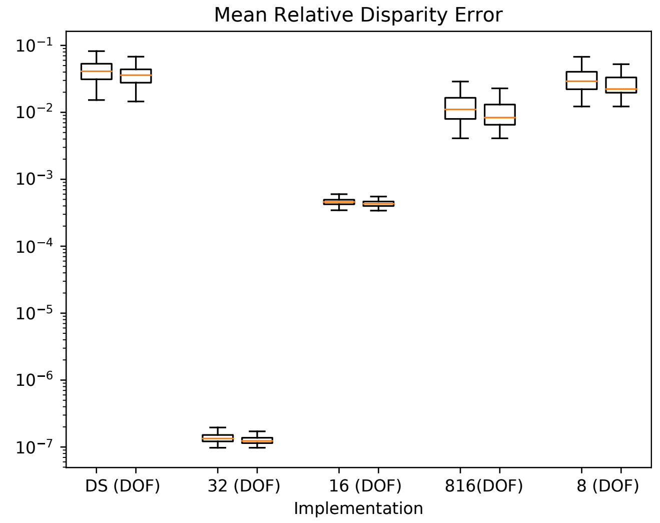

Fig. 5. Mean relative disparity error over data set for different models

They were recorded with the EinsteinVision endoscope [19], whose DOF of [20mm, 200mm] reduces the disparity range to [48.5, 485]. Other available data sets render very similar results and are omitted here due to space restrictions.

TABLE I
AVERAGE RELATIVE DISPARITY ERRORS

Model	$\overline{e}_d$	$\hat{e}_d$	$\overline{e}_d$ (DOF)	$\hat{e}_d$ (DOF)
DS	0.060	1.015	0.049	0.704
32	$1.58{\cdot}10^{-7}$	$5.41{\cdot}10^{-6}$	$1.55{\cdot}10^{-7}$	$4.41{\cdot}10^{-6}$
16	$5.22{\cdot}10^{-4}$	$1.43{\cdot}10^{-2}$	$4.95{\cdot}10^{-4}$	$1.14{\cdot}10^{-2}$
816	0.018	0.771	0.016	0.472
8	0.038	1.079	0.031	0.549

The measurements show very good quality for the models **32** and **16**. The **816** and **8** models exhibit quite large maximum errors, but perform better than the "downscaled" model **DS**.

B. Timing results

Experiments and measurements were performed with the same data sets as above, in two different environments:

- a laboratory setup with two workstations **LA**, **LB**, where only pre-recorded data can be used,
- an Experimental Operating Room with a body phantom, a stereo endoscope, and two workstations **XA**, **XB**, where "live" experiments using the phantom are also conducted.

These cover a broad range of CPU and GPU configurations:

	LA	**LB**	**XA**	**XB**
CPU	i7-2600	i9-9820X	i9-10900X	2×Xeon 4216
GPU	RTX2080Ti	4×RTX2080Ti	RTX2080S	4×A5000

During implementation and optimisation of the processing modules, timing analyses were most important. We utilised several profiling tools to measure the run times (*latencies*) of the functions and code sections. Considering the whole processing chain, however, the *throughput* (average volume of data per time) is much more significant – it also levels out irregularities and variations of the load, and takes account of parallel data processing (multithreading).

TABLE II
THROUGHPUT OF IMPLEMENTATION MODELS (FRAMES / S);
MODEL **8** OMITTED (VERY SIMILAR TO **816**).

Target workstation	**LA**	**LB**	**XA**	**XB**
REF FullHD (1 thread)	2.4	2.8	2.4	2.9
DS 1/16 resolution (1 thread)	20.2	18.0	21.5	14.9
32 (1 thread / 1 GPU)	5.2	5.3	4.1	8.3
16 (1 thread / 1 GPU)	20.1	20.0	16.4	21.5
816 (1 thread / 1 GPU)	23.6	23.9	19.5	27.3
16 (2 threads / 1 GPU)	18.8	18.7	15.7	20.2
16 (2 threads / 2 GPU)	-	40.3	-	42.9
16 (4 threads / 2 GPU)	-	37.9	-	41.7
16 (3 threads / 3 GPU)	-	60.6	-	64.9
16 (4 threads / 4 GPU)	-	80.0	-	74.1
816 (2 threads / 2 GPU)	-	47.8	-	51.0
816 (4 threads / 2 GPU)	-	45.9	-	54.3
816 (3 threads / 3 GPU)	-	71.9	-	60.2
816 (4 threads / 4 GPU)	-	89.3	-	90.1

The throughput measurements (see Tab. II) for the reference HSM network at FullHD (**REF**) are compared to the above-listed models running on different workstations. The results of model **8** are very similar to those of **816**, and are excluded here. For the TRT&DALI models **16** and **816**, multi-threaded and multi-GPU implementations were measured in addition.

Even single-threadedly, the TRT/DALI models achieve significant accelerations to the slow **REF** model, but are below full framerate. However, the multi-threaded solutions with multiple GPUs accomplish dramatic improvements: Already model **16** with just 2 GPUs fully satisfies our requirements of 30 frames/s at FullHD resolution. The results with even more threads and GPUs scale well, and suggest a means for acceleration on less capable GPUs. The utilisation of more threads than GPUs is of no advantage – similar results with more threads are therefore omitted.

V. SUMMARY AND CONCLUSIONS

We presented an algorithm and implementation for fast and accurate estimation of dense disparity maps. We successfully improved and accelerated a state-of-the-art high-resolution network to full frame rate (on 2 GPUs), by employing TensorRT, DALI, and multi-core/multi-GPU execution.

The subsequent improved 3D reconstructions will lead to more accurate registrations, and the increased speed will deliver better usability from the perspective of the surgeon. These benefits build towards the development of an IGS and its translation into the operating room.

The presented solution can of course prove beneficial in other medical or robotics applications where fast high-resolution depth from stereo is indispensable. The relatively high hardware demands of the current implementation could be mitigated with new GPU generations (requiring only a single GPU) or in applications where accuracy is less critical (e.g. automotive sensing), and where our results can support a trade-off between quality and speed.

The next challenge in this context is the lack of reference data – an inherent problem of intraoperative methods. For a human body phantom, however, the method obtains accurate depth data. This will also allow us to numerically assess the effect of disparity variations on 3D reconstruction.

References

[1] A. A. Fretland, D. Aghayan, and B. Edwin, "Long-term survival after laparoscopic versus open resection for colorectal liver metastases," *Journal of Clinical Oncology*, vol. 37, no. 18_suppl, pp. LBA3516–LBA3516, 2019.

[2] S. Speidel, S. Bodenstedt, F. Vasconcelos, and D. Stoyanov, "Chapter 29 - Interventional imaging: Vision," in *Handbook of Medical Image Computing and Computer Assisted Intervention*, ser. The Elsevier and MICCAI Society Book Series, S. K. Zhou, D. Rueckert, and G. Fichtinger, Eds. Academic Press, 2020, pp. 721–745. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012816176000034X

[3] C. Schneider, M. Allam, D. Stoyanov, D. Hawkes, K. Gurusamy, and B. Davidson, "Performance of image guided navigation in laparoscopic liver surgery - a systematic review," *Surgical Oncology*, vol. 38, p. 101637, 2021.

[4] R. Docea, M. Pfeiffer, J. Müller, K. Krug, M. Hardner, P. Riedel, M. Menzel, F. Kolbinger, L. Frohneberg, J. Weitz, and S. Speidel, "A laparoscopic liver navigation pipeline with minimal setup requirements," 2022, unpublished.

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

[6] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[7] G. Yang, J. Manela, M. Happold, and D. Ramanan, "Hierarchical deep stereo matching on high-resolution images," *CoRR*, vol. abs/1912.06704, 2019. [Online]. Available: http://arxiv.org/abs/1912.06704

[8] M. Pfeiffer, C. Riediger, S. Leger, J.-P. Kühn, D. Seppelt, R.-T. Hoffmann, J. Weitz, and S. Speidel, "Non-rigid volume to surface registration using a data-driven biomechanical model," in *MICCAI 2020*. arXiv, 2020. [Online]. Available: https://arxiv.org/abs/2005.14695

[9] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[10] R. Szeliski, *Computer vision algorithms and applications*. London; New York: Springer, 2011. [Online]. Available: http://dx.doi.org/10.1007/978-1-84882-935-0

[11] Baeldung, "Normalizing inputs for an artificial neural network," https://www.baeldung.com/cs/normalizing-inputs-artificial-neural-network, October 2020, online, accessed 2022-06-01.

[12] NVIDIA, "NVIDIA TensorRT documentation," https://docs.nvidia.com/deeplearning/tensorrt/, online, accessed 2022-06-01.

[13] D. Kirk, "NVIDIA CUDA software and GPU parallel computing architecture," in *ISMM*, vol. 7, 2007, pp. 103–104.

[14] NVIDIA, "NVIDIA data loading library (DALI)," https://developer.nvidia.com/dali/, online, accessed 2022-06-01.

[15] J. A. Guirao, K. Lecki, J. Lisiecki, S. Panev, M. Szolucha, A. Wolant, and M. Zientkiewicz, "Fast AI data preprocessing with NVIDIA DALI," in *GPU Technology Conference*. GPU Technology Conference, January 2019.

[16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, January 2009.

[17] The Linux Foundation, "Open neural network exchange - the open standard for machine learning interoperability," https://onnx.ai/, online, accessed 2022-06-01.

[18] N. Valigi, "Concurrency in ROS 1 and 2: from AsyncSpinner to MultithreadedExecutor," in *ROSCon Macau 2019*. Open Robotics, October 2019. [Online]. Available: https://doi.org/10.36288/ROSCon2019-900895

[19] B. Braun SE, "EinsteinVision 3D camera system," https://www.bbraun.co.uk/en/products/b50/einstein-vision.html, online, accessed 2022-06-01.