Technical report of project no. VI20172020068

Tools and Methods for Video and Image Processing to Improve Effectivity of Rescue and Security Services Operations (VRASSEO)

Selected Video-Processing Methods and System Architecture Design

Vítězslav Beran, Marek Šolony, Adam Herout, Vojtěch Bartl, Pavel Zemčík, Svetozár Nosko, Jaroslav Zendulka, Vladimír Bartík, Vojtěch Fröml

Brno University of Technology
Faculty of Information Technology
Božetěchova 1/2
Brno, 612 66, Czechia

December 2017

Contents

Abstract

This technical report describes image-processing and classification methods for extracting information from image data in order to increase the automation of processing data from the surveillance cameras in real time. Another considered source of image data is mobile imaging platforms such as drones etc.

Presented research focuses on specific methods for processing images with lower quality under realistic conditions. Selected techniques utilizes the high-dynamic-range (HDR) images in addition to the standard image representation, the development of image information over time, or the reconstruction of 3D information from multiple images. As a result of these procedures, specific methods are presented focused horizon estimation, 6DOF camera localization, or LPR detection in difficult light conditions.

The report further describes the design of a system for managing video-data and metadata and their subsequent use in order to extract context information and detects predefined situation patterns in an online manner. The system architecture part presents key parts of the system, its functionalities and designed API.

1 HDR images deghosting

HDR video acquisition is a very important feature of modern surveillance, traffic monitoring, and other applications that exploit static cameras. Typically, for both economical and technological reasons, the HDR video is acquired using multi-exposure using sensors with limited dynamic range

Unfortunately, one of the most difficult problems of today's digital photography and video is very limited ability of capturing the dynamic range occurring in the captured scenes. By selection of the aperture and shutter speed, it is possible to select which part of dynamic range is captured and which is lost – and, of course, the loss is an adverse affect in the image and video capture process; however, efforts exist to increase the capability of capturing in High Dynamic Range (HDR).

The most common approach for HDR acquisition is based on merging in radiance domain, in the meaning of real illumination in the given scene. Debevec and Malik [1] proposed an algorithm which can fuse multiple photographs into a high dynamic range radiance map, whose pixel values are proportional to the true radiance values in the scene. The contribution of each pixel is determined from the weight function. HDR image H is then calculated as a weighted average of the individual exposures L:

$$H = \frac{\sum_{i=1}^{N} w(L_i)\frac{L_i}{t_i}}{\sum_{i=1}^{N} w(L_i)} \tag{1}$$

where N is the number of exposures, t_i exposure times and function w the triangle weight function.

The standard HDR merging algorithms are suitable for static scenes only. The movement of objects during sequence capturing is causing adverse effect called ghosting. Various methods to detect and remove ghosting from HDR images have been developed to date. These methods are called as deghosting algorithms.

We proposed a ghost detection algorithm based on prediction of pixel value. It is based on the similar principles as are introduced by Grosch [2]. Since we know the exposition time of each image from sequence, we can predict a pixel values in subsequent (or precedent) images:

$$L_x = L_y \cdot (t_x/t_y) \tag{2}$$

where t_x and t_y are exposition times of images. This equation can not be complied within the over or under-exposed patches in image, where the information is missing. The ghost detection is performed before the HDR merging phase, resulting in ghost pixel mask (further called as *Ghostmap*), where the positions marked in *Ghostmap* by *GhostFlag* and are treated differently during the HDR merging.

The function ω tests the two images whether their pixels follows the prediction:

$$\omega(L_i, L_{i+1}) = \begin{cases} 1 & L_i * \frac{t_{i+i}}{t_i} / \alpha > L_{i+1} \\ 1 & L_{i+1} / \frac{t_{i+1}}{t_i} * \alpha < L_i \\ 0 & \text{else} \end{cases} \tag{3}$$

where the term α represents the tolerance, which must be taken into account, since the sensor noise, quantization errors and CRF precision may influence the predicted value and thus cause the false ghost detections. According to our experiments, we use $\alpha = 1.2$ as default, but it is dependent on target sensor or another image source respectively. In general, decreasing of this ratio leads to more strict ghost detection, where more pixels are marked as ghosts, which eventually lead to lose of dynamic range. Increasing of ratio, on the other hands, decreases the chance of successful ghost detection.

The Ghostmap is defined as follows:

$$G = 1 \Leftrightarrow \exists e = 1;\ e \in \{\omega(L_i, L_{i+1}) \mid\ i \in < 1; n-1 >\} \tag{4}$$

The algorithm description is simplified by pixel range control - all under and overexposed pixels are omitted from prediction. Anyway, all pixels are evaluated against extreme changes of value. This algorithm is working per-pixel and and uses simple arithmetic operations, so it is suitable for implementation on embedded devices. It is also applicable on arbitrary number of images, but our pipeline is designed for sequences of three images. The follow-up HDR merging algorithm is modified and in the areas, where *Ghostmap* indicates movement, incorporates only the pixels from reference image. The reference image is, also in most other related works, the middle image in the sequence.

The results of our deghosting method are presented on Figures 1. Our method is applicable for any surveillance purposes and even more; deghosting results are presented on traffic monitoring task, where the main goal is to preserve as most details as possible for evidence purposes, especially the license plate of approaching car has to be readable. Figure 1 contains standard movement of car approaching camera by approx. 50km/h.

2 Horizon Detection for Camera-Pose Estimation

Estimation of viewpoint is critical for understanding a given scene. Quick (and possibly not quite accurate) guess of the viewpoint is an important component of the *gist* of the scene Creating such a representation of an image can be beneficial not only for human visual processing but also in computer vision. One of the most mentioned viewpoint aspects is the image's *horizon.* Although horizon is a fairly intuitive characteristic of the viewpoint, in some

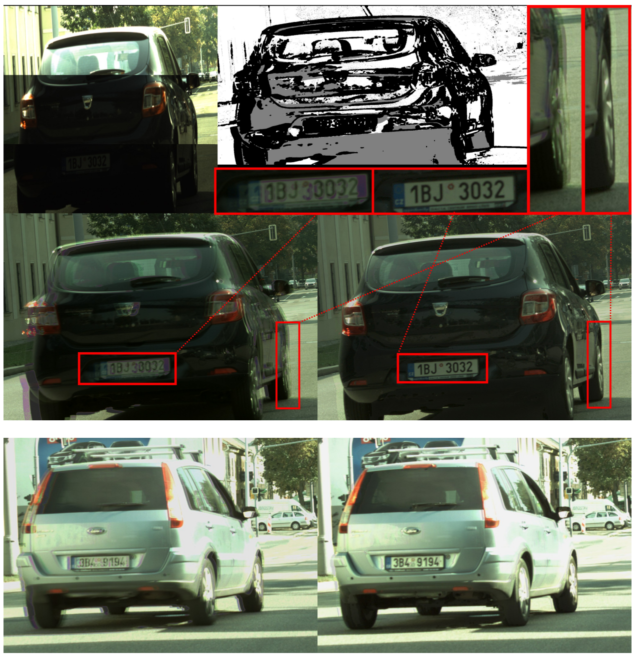
Figure 1: Deghosting method results. Top left quarter contains stripes of original images, where significant car movement could be observed. The bottom left image reflects movement without our deghosting technique. Top right image contains Ghostmap used for image recovery (gray signs the under/overexposed patches). The right bottom image shows the resulting deghosted image. Under the Ghostmap, the upscaled licence plates are shown.

complicated scenes (urban, occluded, . . .), establishing exact horizon can be complicated for humans and even more so for machines.

Horizon is very often used for camera calibration because two horizontal vanishing points lie on the horizon line. With the knowledge of the horizon line and the third (vertical) vanishing point, camera calibration can be done. An approach which estimates the vanishing points by connecting corresponding points of the same object and then constructs the horizon is often used for camera calibration with human tracking , where pedestrian's head and feet are connected by lines as the person moves across the scene and their intersection lies on the horizon line. Similarly, vanishing points and the horizon line can be localized for example by detecting pedestrians' toes in the ground plane. Although camera calibration methods which avoid using horizon exist, they typically have some constraints, for example, known pedestrian height distribution , 'Manhattan world' scene , or dominant motion only in one coherent direction Horizon can also be used for estimating 3D scene geometry and object detection support.

Our goal is to detect the horizon (in the sense of the ideal line of the ground plane perpendicular to the gravity) in a single static (often surveillance) uncalibrated camera stream based on motion of objects in the scene without any a priori constraints except that the majority of motions happen in horizontal planes (which share the same horizon by definition). Such a method can be later used for automatic camera calibration, scene understanding, and other computer vision tasks.

We assume that the scene contains arbitrary objects (pedestrians, cars, dogs, cattle, machinery, . . .) with an arbitrary height/size distribution. The scene does not need to be manhattanian and the motion can appear in any direction and in virtually any place in the scene. Existing methods for horizon estimationuse a single static image without assumption of a movement in the scene. Although motion can be used for horizon estimation for example in the form of cloud motion together with wind velocity , in our work we assume surveillance cameras without any constraints, no clouds thus need to be present in the scene and no additional information is provided.

Although datasets with horizon position in image exist (HLW , ECD , YUD , they only contain static images without any motion in the scene. To evaluate the method and to allow for future comparison (to our knowledge, there is no existing dataset dealing with this issue), we collected a dataset based on publicly available IP cameras. We manually constructed a ground truth by using geometric properties of objects in the scene and we also collected human annotations which cast a light on the algorithm's performance and allow for its comparison to human (well trained and routinely used) *gist* of the scene mentioned earlier.

Most of the recordings were taken from publicly available IP cameras, some recordings were captured by a camcorder. One scene was used from the PETS dataset but it is not a very suitable one because of its short duration.

The recordings differ in many aspects – places, camera positions, day time, scene type, duration, resolution, . . . The collection includes scenes from traffic, indoor, outdoor, pedestrians, etc. Some recordings were taken during night so different light conditions are also available. The duration of the recordings is in the range from 5 minutes to 30 hours, mean length is about 2.9 hours. The resolution is largely varying with the given IP camera's quality in the range from 320×240 to $1\,920 \times 1\,080$ pixels. In total, 47 different usable scenes were obtained. Some scenes were re-captured under different conditions (lighting, crowd density, . . .), yielding 66 recordings in total.

Obtaining horizon ground truth turned out to be a challenging problem mostly due to very frequent occlusion of the natural horizon in the scenes (buildings, horizon out of frame, . . .). In order to obtain a geometrical estimation of the horizon, we extracted one representative frame from the video recording and manually annotated groups of lines that are parallel in the orig-

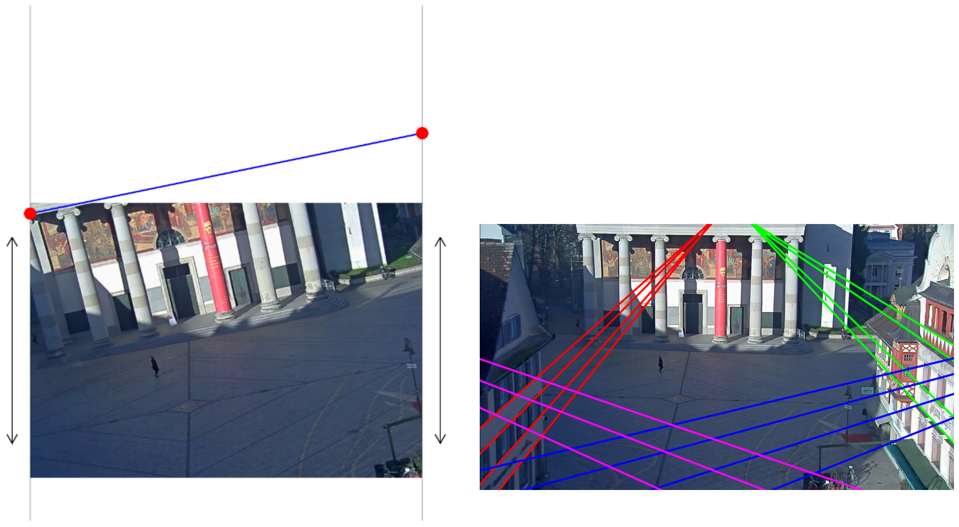

Figure 2: Scene horizon annotation principles. **left:** Web annotation tool for crowsourced data collection. **right:** 'Geometric' ground truth annotation by using scene parallel lines.

inal 3D scene (edges of a house's windows, markings on the streets, patterns in the pavings, etc.). Each of these groups of lines provides one estimated vanishing point; all vanishing points should be collinear – coincident with the line of the horizon. The horizon is obtained by using the least-squares linear regression on the set of the estimated vanishing points obtained as the minimal error intersections of the lines in the individual groups.

Aside from the geometric horizon estimation, we collected horizon annotations by humans. We created a web annotation tool (Figure 2 left) and knowledgeable people were asked to estimate the horizon in the scene frame as precisely as possible. People are able to localize the horizon in a given image with small error after a short description of what horizon really is. To prevent people of simply assuming a horizon line being always horizontal in the image (though this is common in many camera shots), the images given to the users for annotation were rotated by $\pm 20°$ and the maximal rectangle was slightly cropped as in Figure 2 left.

Participants estimated the horizon by moving a visual line with markers on its sides as precisely as possible – the users could try different positions of the controlled line and look for the best match. Every participant marked 20 least annotated scenes. The annotations were filtered to rid of annotations clearly skipped or carelessly performed. Finally, 16 – 21 annotations for each of the 47 scenes are available (mean number 18.42 annotations per scene by different human subjects).

The proposed algorithm is fully automatic in the sense that no user input is needed per-camera and it works with various scenes (indoor, outdoor, traffic, pedestrian, livestock, etc.). The algorithm detects moving objects, tracks them in time, assesses some of their geometric properties related to the object dimensions and infers observations related to the position of the horizon. We collected a dataset of 47 public camera streams observing suitable scenes of various nature. We annotated ground truth horizons based on geometric properties in the images and by direct human input. We evaluate the proposed algorithm and compare it to human guesses – it turns out that the algorithm is on par with humans or it outperforms them in the difficult scenes.

3 Camera-Pose Estimation based on SLAM

In robotic applications, to solve the problems such as automatic navigation or obstacle avoidance, the map of the environment needs to be estimated. The ability to build a map allows the mobile robotics to perform various tasks in complex, unknown environments without relying on the external reference system such as GPS. The estimation of the map and simultaneous localization is known as SLAM problem.

The SLAM needs to solve the "chicken and egg" problem, where the robot needs a map to localize and at the same time to use the pose to build and update the map. The robotic sensors such as cameras, range scanners or odometric sensors are inherently subjected to noise. To deal with the uncertainty of the sensor measurements, various approaches can be applied.

An intuitive way to represent SLAM problem has been proposed in [3] as a graph based formulation. This formulation represents SLAM problem as an optimization of graph, where the vertices represent robot poses and landmark positions and edges represent measurements. Although the measurements are affected by noise, the solution to the graph is a configuration of the nodes that is maximally consistent with the measurements. An efficient technique for solving this SLAM representation has been introduced as soothing and mapping [4]. Soothing methods estimate the full trajectory of the robot as well as all landmarks from the set of measurements. The sparse nature of the SLAM is exploited and efficient implementation and manipulation with sparse matrices is employed for solving factorization of either information matrix (containing inverse covariances) or measurement Jacobian.

The general SLAM formulation allows additional sensor measurements such as odometry, GPS or IMU to be incorporated into the SLAM system to further improve the accuracy of localization and mapping.

We aim to evaluate existing versions of SLAM solutions able to run on Micro Aerial Vehicle (MAV) equipped with NVIDIA Jetson [1] platform. Jetson platform contains powerful GPU for embedded systems applications. It

[1]http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

provides a fully functional NVIDIA CUDA platform for quickly developing and deploying compute-intensive systems for computer vision, robotics, medicine, and more. Additionally it is able to compile and run OpenGL 4.4, and Tegra-accelerated OpenCV software.

The main qualities that important for an autonomous operation of MAV include range of supported sensors, performance on NVIDIA Jetson platform, compatibility or operation under Robotic Operating Syetem (ROS) and modularity. We investigated available SLAM softwares and and evaluated them on multiple datesets - aerial drone dataset, to evaluate robustness and real-time capabilities and dataset with ground truth to evaluate accuracy. These algorithms create map of the environment from sensors, and at the same time estimate the position and rotation of the camera in the scene. Following algorithms were examined:

- LSD-SLAM [5]
- ORB-SLAM [6]
- Google Cartographer [7]

3.1 LSD-SLAM

LSD SLAM is a monocular visual SLAM system estimating the map and pose of the camera from a subset of images called keyframes. The movement of the camera is tracked along the keyframes through direct image alignment. This approach does not use keypoint or feature detection, the consistency between frames is achieved using intensity information of whole image, applying photometric error minimization. This allows for creation of semi-dense inverse depth maps storing information about inverse depth to 3D points corresponding to image pixels. LSD SLAM is capable to perform loop closure of the camera trajectory, reducing the camera drift caused by the noise in sensor measurements.

The map is optimized using graph optimization approach, where keyframes are represented as nodes and the measurements as a edges of a graph. LSD SLAM runs in two threads, one for camera tracking and one for map building and measurement incorporation.

3.2 ORB-SLAM

ORB-SLAM is a visual SLAM system that supports monocular cameras as well as stereo cameras or RGBD cameras. It utilizes ORB features, which are multi-scale FAST corners with 256 bit descriptors associated. The advantage of ORB features is its fast feature and descriptor extraction and good viewpoint invariance, which is especially useful for wide baseline registration. Loop closure is also supported in ORB SLAM and it is based on a bag-of-words database, in which the algorithm can query for actual feature state to detect previously visited areas.

The map is represented by sparse 3D points. The optimization is also based on graph optimization algorithm, where the reprojection error of the observed 3D points in minimized. ORB-SLAM works in 3 threads, camera tracking, map building and loop closure detection. Implementation of this approach is available either stand-alone or incorporated to ROS system.

3.3 Google Cartographer

Google cartographer is a 2D/3D SLAM algorithm based on laser scan measurements. Data from odometry and Inertial Measurement Unit (IMU) are used to track and extrapolate robot position and laser scans to build the map. The SLAM works at local and global level. At local level, the laser scans are inserted into a *submap*, which is considered accurate for small areas. To cope with the accumulation of error, the map is continually optimized with pose graph optimization.

Loop closure is also supported by Google Cartographer, by scan matching actual scan against all finished submaps.

3.4 Datasets and Evaluation

All SLAM algorithms (ORB-SLAM, LSD-SLAM, Cartographer) were run on NVIDIA Jetson TK1 hardware, and evaluated on MAV dataset [2] and TUM dataset [8]. TUM dataset contains sequences of camera movements with known ground truth, therefore the error between the estimated camera poses and true camera poses can be computed.

Google Cartographer SLAM was tested using 2D laser scan dataset provided with the implementation to test the capabilities of the Jetson hardware. 3D version of Google Cartographer could not be successfully evaluated yet.

Table 1 shows the overview of the SLAM algorithms, and Table 2 shows the accuracy and performance results of LSD-SLAM and ORB-SLAM algorithms.

According to the results, ORB-SLAM achieves better accuracy and framerate performance than LSD-SLAM. On the other hand, LSD-SLAM provides semi-dense 3D occupancy map, which is much more suitable as an input for robotic navigation and path planning. Further experiments including MAV datasets with known ground truth need to be performed to obtain results in real life scenarios.

[2]https://ivul.kaust.edu.sa/Pages/Dataset-UAV123.aspx

Algorithm	Input Sensors	Map type	Loop closure
LSD-SLAM	monocular camera	3D occupancy semi dense map	yes
ORB-SLAM	monocular camera, sereoscopic camera, depth camera	sparse 3D point cloud, keyframe graph	yes
Google Cartographer	laser range scanner, odometry, IMU	probability grid map	yes

Table 1: Overview of SLAM algorithms.

	TUM-xyz		TUM-desk		Drone	Cartographer
	Error[cm]	FPS	Error[cm]	FPS	FPS	FPS
LSD-SLAM	15.1	12	16.6	14	11	-
ORB-SLAM	6.8	22	6.2	23	20	-
Google Cartographer	-	-	-	-	-	8

Table 2: Error and performance evaluation.

4 Data Storage and Management

The hardware architecture of the system consists of sensing devices, field stations and a main central server.

1. **Sensing device** - The task of the sensing device, typically a camera, is to acquire video data for further processing. Input video data will be captured by one or more cameras, will be streamed by the system's field station. At the same time, however, these video data on the input side may be pre-processed and streamed together with extracted metadata to the field station.

2. **Field stations** - Here, images/video and metadata extracted from external or internal modules are stored in the system. Field station system allows moving data to a central server and query data stored on the central server.

3. **Central server** - the server stores the previously captured data, namely images/videos and their metadata.

The logical architecture of the system consists of ViAn Server (Video Analysis Server), processing and analytic modules and operational applications.

1. **ViAn Server** provides management of video data and metadata extracted from them, registers processing and analytic modules. It provides database and basic analytical services. Its task is to support the

management of video data and extracted metadata, including the support of some analytic tasks over these metadata. It uses some parts of the existing VTApi application interface providing supportive functionality for deploying computer vision algorithms. This interface has been extended to support algorithms working with real-time video-data streams by extending its Videos API. ViAn service provides the following set of APIs:

   (a) **Dataset** API - manipulation of entire datasets encapsulating video data, metadata, active processing tasks and computed results.
   (b) **Videos API** - management of videos, images and video streams and its metadata
   (c) **Processing tasks API** - allows definition of computing jobs to be assigned to computing processes and generic querying of results
   (d) **Processes API** - control of running processing tasks
   (e) **Events API** - specific event-based querying of results

2. **Processing modules** primarily serve to extract metadata from data captured by sensing devices. An example of such a module may be one extracting information about moving objects in the observed area.

3. **Analytic modules** perform more advanced analyses of previously extracted metadata, such as similarity-based search for objects occurring at a given location within a given time interval.

4. **Operational applications** define the required task, manage its execution and visualize the results.

The deployment of the software components to hardware nodes is depicted in Figure 3.

ViAn Server and processing and analytic modules and operational applications can be deployed on both the field station and the main server. The field station deployments can provide limited functionality compared to the main server one, for example only some processing and analytic modules. Some modules can be physically deployed to devices other than a field station or a central server, such as a computer dedicated to more computationally intensive metadata extraction.

Field stations receive multimedia data and metadata from the sensing device and allow the user to quickly respond to the processing of this data. In addition, the user should be able to decide which data and metadata from his station should be synchronized with the main server, or query the server for relevant information (e.g., previous occurrences of the object in previous analyses).

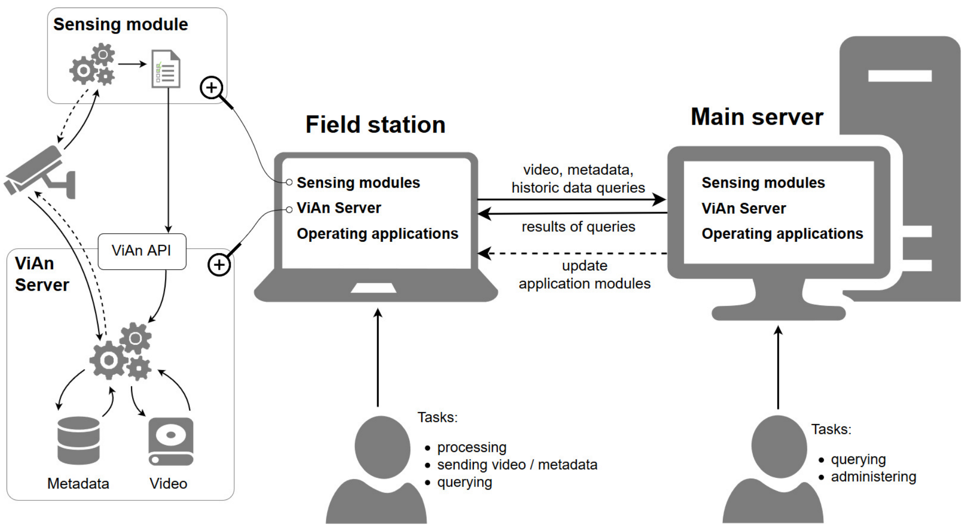

Figure 3: Basic concept of data storage in VRASSEO

The main server provides similar functionality as field stations. Moreover, it is also possible to perform offline analyses of historical data stored on the server. As a source of metadata and video data, computer nodes with sensing modules can be connected to the server, allowing distributed processing, for example, when using stationary cameras.

An important stage was the preparation of the interim development and testing server environment in which the development takes place and will take place before the new server purchased within the project is put into operation. The interim development and testing server, which was created during the first phase of the project solution, includes in particular:

1. Server - for the purposes of the central server, a proactive test server has been set up and selected libraries and support tools such as PostgreSQL 9.6.2, Libpqtypes 1.5 .1, OpenCV 3.1, PocoProject 1.7.8p2, ZeroMQ 3, Google Protocol Buffers 3.1, and more.

2. Development environment - For a simpler system development, a definition file was created to create a Docker container. The container is based on Ubuntu 16.04 and has been added to the above mentioned libraries and support resources, but in the versions available for Ubuntu 16.04 (for example some versions: PostgreSQL in version 9.5, Opencv version 2.4, ZeroMQ version 2.2 or Google Protocol Buffers version 2.6).

As a pilot task that will be used during development, video activity detection was ddeveloped and deployed.

References

[1] P. E. Debevec and J. Malik, "Recovering high dynamic range radiance maps from photographs," in *ACM Trans. Graph.*, SIGGRAPH '97, 1997.

[2] T. Grosch, "Fast and robust high dynamic range image generation with camera and object movement," 2006.

[3] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, no. 4, pp. 333–349, 1997.

[4] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *Intl. J. of Robotics Research, IJRR*, vol. 25, pp. 1181–1204, December 2006.

[5] J. Engel, T. Schöps, and D. Cremers, *LSD-SLAM: Large-Scale Direct Monocular SLAM*, pp. 834–849. Cham: Springer International Publishing, 2014.

[6] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *CoRR*, vol. abs/1502.00956, 2015.

[7] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271–1278, May 2016.

[8] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.