*Acta Polytechnica CTU Proceedings* **6**:28–33, 2016
doi:10.14311/APP.2016.6.0028

# Mapping the Surroundings as a Requirement for Autonomous Driving

Martin Steininger*, Christoph Stephan, Christian Böhm, Fabian Sauer, Roland Zink

*Faculty of Electrical, Media and Computer Engineering, Deggendorf Institute of Technology, 94469 Deggendorf, Germany*

* corresponding author: martin.steininger@stud.th-deg.de

Abstract. Motivated by the hype around driverless cars and the challenges of the sensor integration and data processing, this paper presents a model for using a XBox One Microsoft Kinect stereo camera as sensor for mapping the surroundings. Today, the recognition of the environment of the car is mostly done by a mix of sensors like LiDAR, RADAR and cameras. In the case of the outdoor delivery challenge Robotour 2016 with model cars in scale 1 : 5, it is our goal to solve the task with one camera only. To this end, a three-stage approach was developed. The test results show that our approach can detect and locate objects at a range of up to eight meters in order to incorporate them as barriers in the navigation process.

Keywords: Microsoft Kinect, Mapping, 3D Mapping, Robotour, Autonomous Driving.

## 1. Introduction

In addition to the extensive launch of full electric vehicle (fev) autonomous driving is deemed to be a key research and development trend in the automotive industry. The entire industry is working on the topic and has already developed first solutions for automatic braking, line keeping (i.e. AUDI "Active lane assist" or BMW "Active Assist") or even autopilots for driverless cars like Tesla Motors' "Autopilot" or Google's "Driverless Car" [1]. The used navigation and self-driving algorithms run on maps [2] and require very accurate digital (geo-)data of the environment [3]. Therefore, the spatial coverage of the surroundings as a basis for autonomous navigation of the vehicle is common to all applications.

Consequently, autonomous driving – regardless of the respective task such as automatic braking, adaptive cruise control, lane keeping or autonomous driving – requires detection and imaging of the space which it will pass through. Photo cameras, LiDAR-, RADAR- or other sensors continuously detect and measure the surrounding of the car, creating a digital spatial image of the area (map). Combined with positioning services like GPS or Galileo, the location of the vehicle can be determined and set in relation to surrounding objects such as other vehicles or obstacles. However, it is not an ordinary cartographic map resulting from the permanent mapping process but digital images that are stored temporarily as server frames for the navigation. The navigation of the car finally relies on mathematical functions (algorithms) to pass through the digital and real space on the shortest (spatial) or fastest (time) way.

This paper addresses the issue of the space detection which is considered as an essential basis of autonomous driving and robot navigation (see for example [4]) and describes the use of a Microsoft X-Box One Kinect stereo camera for autonomous driving for the outdoor delivery challenge Robotour 2016 (robotika.cz). We present an approach for mapping the surroundings including the three steps (1) 3D capturing, (2) projection and filtering and (3) object detection. The hardware and the approach are implemented on a four wheel drive off-road buggy, scale 1 : 5.

The main hardware consists of a Congatec TS180 COM Express module. For the main processing, it communicates with the X-Box One Kinect and a Raspberry Pi and displays the crucial information on a Krämer V-800 Touchscreen. The Raspberry Pi reads the sensor values from the GPS and the Bosch BNO055 and transmits them to the Congatec module. It receives the steering and speed values from the Congatec module, which are transmitted to the Freescale KL25Z microcontroller, which in turn controls the RC-Car model via PWM signals.

## 2. Approach

The development path of the approach to map the surroundings for an autonomous driving vehicle should be well-defined to avoid inconsistencies. The first step is answering the question which hardware will be used to detect objects. With the goal to use only a Kinect camera as sensor, the approach relies on the processing of the depth, and standard images. Both data represent a 2D respectively 3D model of the environment. The approach is characterised in three development steps:

- 3D capturing,
- projection,
- object recognition.

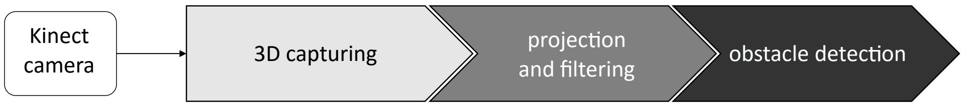

Figure 1. spatial mapping of the current surroundings.

It is developed to simplify complex processes, describe them in a proper way and visualize all of the main development steps in a manner that is easy to understand. According to the development steps, the approach is divided into three processes, see Figure 1. At first, the Kinect camera films the space in front of the vehicle in order to gather potential obstacles. The second step contains the projection of the depth images on a 2D ground plane. At last, the obstacles or object will be recognized during the third step. The three main steps will be explained below.

## 3. Related work

In 2002 Sebastian Thrun (see [5]) showed that the problem of terrain mapping is to generate a spatial model of a robot's or vehicle's environment using sensors and cameras. Concerning this point there are several more subproblems:

- measurement noise,
- robot must choose its way during mapping process,
- environment changes over time,
- possibly the hardest problem — the so called data association problem: correspond sensor measurements to the same physical object in the real world.

"Mapping is largely considered the most difficult perceptual problem in robotics." [5, p. 6]. In 1989 Herbert et al. already presented the so called locus algorithm to build terrain maps based on a single active range sensor, by "computing the intersection of the surface observed by the sensor with the vertical line passing through $(x, y)$" [6, p. 998]. Frankhauser et al. [4, p. 2] introduced an elevation mapping approach from a "robot-centric perspective" with relative distances to the robot and a permanent update process of the entire elevation map with information about the motion of the robot.

Kleiner and Dornhege presented their autonomous driving vehicle for Urban Search and Rescue (USAR) with two different methods. First the rapid mapping of a large-scale environment by wheeled robots, and second the mapping of rough terrain by tracked robots [7].

Hornung et al. [6] developed an open source library to represent the environment in memory using octrees. The cells of the discretized space can have one of the following states: unknown, free or occupied. To map the sensor data into the octree they use a probabilistic sensor model.

Thrun et al. [8] describes Stanley, the winner of the 2005 DARPA Grand Challenge. The robot was built by Stanford University researchers and uses laser range finders as well as RADAR sensors for sensing the environment. The evaluation of the sensor data is supported by artificial intelligence and machine learning algorithms.

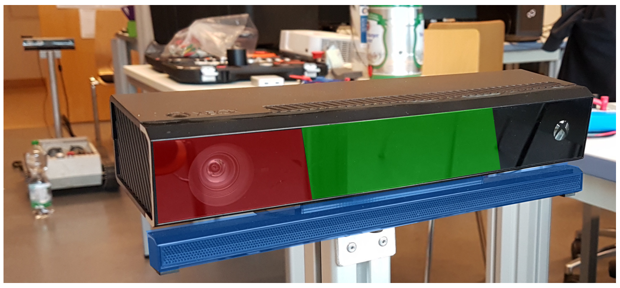

Figure 2. Overview of the Kinect camera.

## 4. 3D capturing

As a 3D sensor is a rather expensive component, the Kinect is a cheap alternative compared to the functions it provides. The camera is doing most of the calculations on its own SoC, thus reducing the processing effort for the main unit. The first version of Microsoft Kinect camera was developed by Prime Sense, an Israeli 3D sensing company, which was bought by Apple in 2013. After that, Microsoft created its own system based the existing Prime Sense Technology [9].

### 4.1. Overview

The Kinect camera has an array of sensors, which can be divided into three categories: a depth sensor (green), a colour sensor (red) and an audio sensor (blue), as seen in Figure 2. The audio sensor is not used for the 3D reconstruction. The colour sensor consists of a CMOS camera with an infrared light filter. The main use of the colour sensor is the track detection. The colour image can be mapped with the depth information, although the resolution of the colour sensor is higher than that of the depth sensor. That is the reason why there are multiple colour pixels per one depth pixel. If there are more depth pixel, than colour pixels, the colour pixels need to be split into multiple smaller pixel, with the same information. The depth sensor consists of a CMOS camera and an infrared led blaster. The CMOS camera is filtered so that it just sees near-infrared light which is emitted by the infrared led blaster. All the information is processed in a specially designed SoC (Figure 3). This allows the SoC to process a depth map containing all the depths for every pixel and map the colour pixel to it. The result of the processing is a data stream containing an audio, a video, and a depth stream.

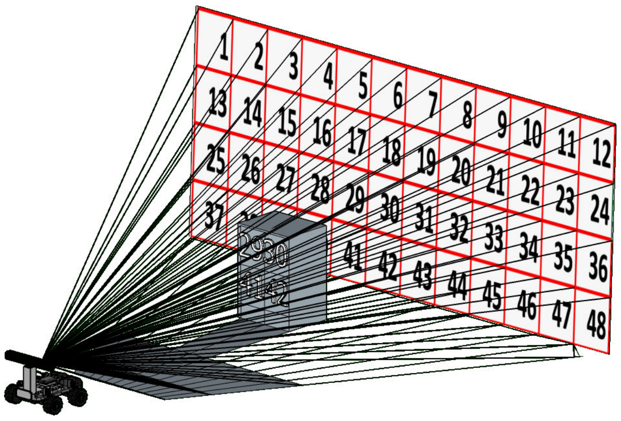

FIGURE 3. 3D reconstruction of the Kinect [11, 12].

### 4.2. 3D reconstruction

There are multiple ways of reconstructing the depth information from an image. The classic way needs two calibrated cameras at a defined distance. This allows for calculating the distance to the object with trigonometry formulas. The difficulty will be in differencing the objects resulting in the so called correspondence problem which “refers to the problem of ascertaining which parts of one image correspond to which parts of another image, where differences are due to movement of the camera, the elapse of time, and/or movement of objects in the photos.” [10].

The Kinect overcomes this problem by using a method called “Time of Flight”. In this method the travel time of infrared light, which is emitted by a led, is measured for each pixel, as seen in 3.

“A strobed infrared light illuminates the scene, the light is reflected by obstacles, and the time of flight for each pixel is registered by the infrared camera. Internally, wave modulation and phase detection is used to estimate the distance to obstacles (indirect ToF)” [13].

The problem is that the infrared light can be washed out by other near-infrared light sources like sunlight, resulting in the Kinect not being able to detect the infrared light and losing the depth information.

## 5. Projection and filtering

The Kinect camera is mounted on the chassis in an arbitrary fixed angle, tilted to the cars chassis. The exact angle in which the camera is mounted to the ground is not known. It can be measured, but to be independent from that setting, the ground plane coefficients, which are returned from the Kinect, are used. The coordinate system of the returned 3D point cloud (as camera coordinate system in the following; indicated with a $_{\text{car}}$ subscript) does not match with the coordinate system of the car (as car coordinate system in the following; indicated with a $_{\text{camera}}$ subscript). In Figure 4 both coordinate systems are shown.

The body-frame which can be requested from the camera contains a four-dimensional vector representing the floor coefficients by $\vec{p}_{\text{coeff}} = (A; B; C; D)$.

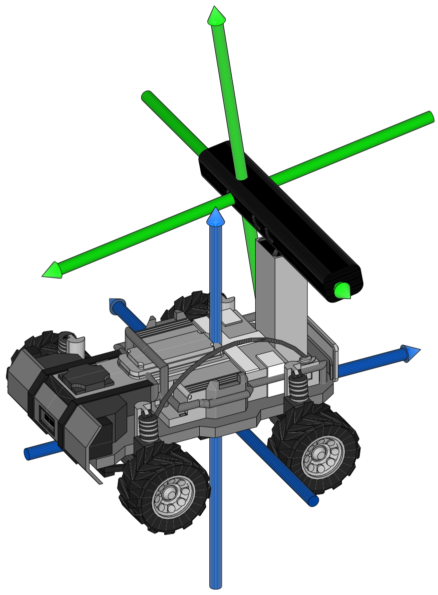

FIGURE 4. The camera coordinate system (green) and the car coordinate system (blue) [11, 12].

These coefficients are combined in (1) to receive the plane equation for further mathematical operations:

$$0 = Ax + By + Cz + D \tag{1}$$

In a first step, the distance of every point to the ground is calculated. A point is considered to be a part of the ground if a certain limit is not exceeded. Therefore they are removed from the point collection if the distance to the ground is smaller than a certain distance. The remaining points are processed in the following steps.

In the next step, each frame of the Kinect has a resolution of 512 by 424 and can thereby result in a large amount of 3D-pixels. A pixel is returned, if it is in the specified range of the camera. The nearest distance the camera returns a pixel for is about half a meter. The maximum distance is about eight meters. So there are quite a large number of points waiting to be computed. The rather small computing system with limited processing power on the car must handle this in real-time. For the presented project, a real-time reaction is defined with a latency of a few hundred milliseconds. To reach this goal, one issue is making the algorithm faster. This could be realised by:

- removing as many points as possible to save energy and processing effort converting the points from one coordinate system to another,
- adjusting processes how the data is processed step by step,
- using a programming language with small processing overhead.

To reduce the further processing effort the points get filtered next. A new parameter named *MinDistance* is introduced in the GUI (Graphical User Interface) where the points must have a greater distance from the camera than that value, to prevent recognising parts of the car as obstacles. This is needed due to the current setting of the car. There may be points above the height of the car so that the car can drive beneath. These points get removed if they are above a maximum height from the ground plane which is defined as a parameter named *MaxHeight*. Obstacles far away are also not interesting for the process at the moment. In the same way points far away are also not interesting and get filtered with an allowed maximum distance named *MaxDistance*.

In this section some functions are introduced to simplify the explanation of the coordinate transformation. The first three equations deal with projections. Project a vector $v$ onto another vector $v_{\text{onto}}$ (2), also a vector $v$ onto a plane $E : 0 = (\vec{x} - \vec{r}) \cdot \vec{n}$ (3) and a point with position vector $\vec{x}$ onto a plane (4):

$$f_{\text{v2v}}(\vec{v}_{\text{orig}}, \vec{v}_{\text{onto}}) : \vec{v}_{\text{proj}} = \left(\vec{v}_{\text{orig}} \odot \frac{\vec{v}_{\text{onto}}}{|\vec{v}_{\text{onto}}|}\right) \tag{2}$$

$$f_{\text{v2p}}(\vec{v}_{\text{orig}}) : \vec{v}_{\text{proj}} = \vec{v}_{\text{orig}} - f_{\text{v2v}}\left(\vec{v}_{\text{orig}}, \vec{n}\right) \cdot \vec{n} \tag{3}$$

$$f_{\text{p2p}}(\vec{x}_{\text{orig}}) : \vec{x}_{\text{proj}} = \vec{x}_{\text{orig}} - f_{\text{v2v}}\left(\vec{x}_{\text{orig}} - \vec{r}, \vec{n}\right) \tag{4}$$

Next the normalisation is defined in (5):

$$f_{\text{n}}(\vec{v}_{\text{orig}}) : \vec{v}_{\text{norm}} = \frac{\vec{v}_{\text{orig}}}{|\vec{v}_{\text{orig}}|} \tag{5}$$

Since the source coordinate system of the Kinect is in three dimensional space and the destination system is only two dimensional, it is necessary to project every detected 3D-pixel onto the ground plane. In the three dimensional coordinate space of the Kinect we define a two dimensional coordinate space for the new coordinate system. The new coordinate system can be seen as a plane in the original space. Therefore an origin and also two vectors for the two axes of that coordinate system are needed.

The two dimensional coordinate system for the obstacle detection is implemented within the three dimensional coordinate system of the Kinect camera. The camera position perpendicular above the ground is origin: $C_{\text{camera}} = f_{\text{p2p}}(O)$ with $O$ as the origin of coordinate system. The axes are defined as $e_x = f_{\text{n}}\left(f_{\text{v2p}}\left((0; 0; -1)\right)\right)$ and $e_y = f_{\text{n}}\left(f_{\text{v2p}}\left((-1; 0; 0)\right)\right)$. In Figure 4 you see the orientation of the axis.

Now, (6) can be used to project all points into the new car coordinate system:

$$P_{\text{car}} = \begin{pmatrix} f_{\text{v2v}}\left(P_{\text{camera}} - C_{\text{camera}}, e_x\right) \\ f_{\text{v2v}}\left(P_{\text{camera}} - C_{\text{camera}}, e_y\right) \end{pmatrix} \tag{6}$$

Next, the obstacle detection will use these projected points and create a barrier along the nearest edge of the recognized obstacles.

## 6. Obstacle detection

When implementing algorithms for autonomous vehicles it is mandatory to detect and tag obstacles in front of the craft. As described, a depth image is provided by the Kinect camera from where each single depth pixel is projected to a ground plane with each having the correct distance and position to the camera's point of view during the first stage of image processing.

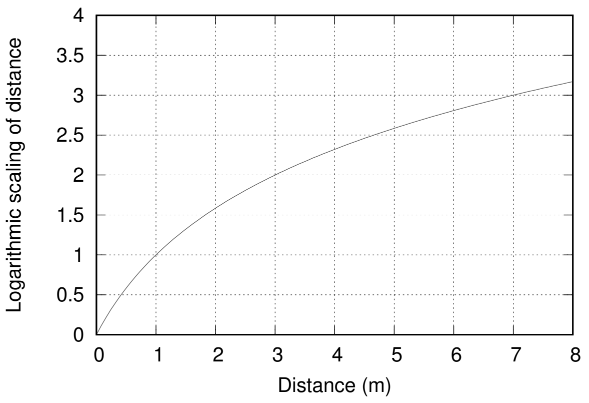

Figure 5. Logarithmic distance scaling for image size reduction purposes.

As a first step, the depth points contributed by the Kinect camera and the projection to the ground plane are transformed into a 2D image. The next three steps described in this section include:

- distance scaling,
- Gaussian functions,
- and the actual object and obstacle detection.

### 6.1. Distance scaling

The project team faced problems concerning computing power: the large number of pixels coming from a depth frame provided by the Kinect camera exceeded the available and limited hardware.

For the purpose of reducing the computing effort for processing the Gaussian function (see subsection 6.2) the project team decided to scale the projected image's size by its height logarithmically. As shown in Figure 5, the scaling of distances allows to have more detailed image information in a closer range to the camera's origin than in a larger distance to this specific point.

The distance we get from the Kinect Camera is a floating point number, the index we need to reference a pixel within and on the image is an integer. If the distance is only rounded to the next integer value there is one pixel per meter. Because of this a parameter is introduced which represents the resolution of the image's distances. The scaled value is divided by this value to get the index as an integer, after rounding of this new integer we get the correct index value for the image.

For example, 0.5 m in reality are mapped to an index value of 0.585 after the scaling, while 8 m in the real world, which is the Kinect camera's maximum recognition distance, correspond to an index value 3.171 logarithmically scaled.

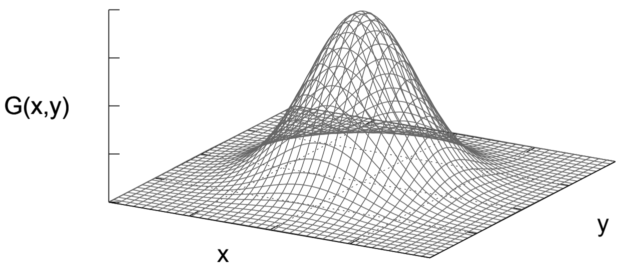

Figure 6. Gaussian distribution in 2D.

As can be seen in Figure 7 the opening angle of the camera is segmented with a specific resolution, which can be set as a parameter. Every segment is mapped to a single rectangular column into the projected image.

The advantage of this approach is that near distance pixels are richer in detail than more distant pixels.

## 6.2. Gaussian function

The next step is to smoothen the depth image to reduce single error pixels or noise and smoothen pixel clouds for further and more precise computations and calculations. Although there are several options for smoothing images, the chosen solution is the Gaussian filter

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (7)$$

In (7), the Gaussian function for two dimensions is shown. Where $x$ is the distance from the origin on the horizontal axis and $y$ is the distance from the origin on the vertical axis, and $\sigma$ is the standard deviation of the Gaussian distribution. The distribution can be seen in Figure 6 for two-dimensional Gaussian functions.

This formula produces a surface of concentric circles with a Gaussian distribution from the center point. The new value of each pixel is set to a weighted average of that specific pixel's neighborhood. The original pixel's value receives the highest Gaussian value and neighborhood pixels receive smaller weights as their distance to the original pixel increases. This results in a blur effect [14, 15].

Figure 7 shows three identical depth frames which are already projected to a ground plane (you see the frames in a top view) and transformed into a gray image. On the left-hand side, the single frame is displayed without any filtering or blurring. There are many false pixels, noise and distortion.

A Gaussian filter is already used in the middle frame, nevertheless quite a lot noise and other false pixels can be seen. The approach works in practice but still needs a little fine tuning. Therefore a smoother configuration of the Gaussian settings was used.

The third frame on the right-hand side displays very little distortion, noise and false pixels, but provides clearly identifiable pixel clouds. These pixel clouds mostly represent objects or other obstacles in the range of the Kinect camera.

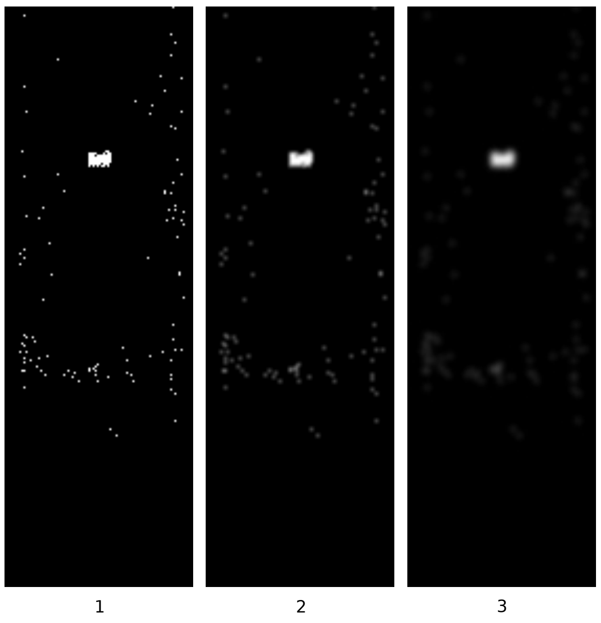

Figure 7. Three identical frames with no Gaussian filtering in frame 1, minimal Gaussian filtering in frame 2 and for the project's obstacle detection best Gaussian filtering settings in frame 3.

## 6.3. Detect obstacles

The last step before actually being able to detect obstacles is removing unnecessary pixels which still are in the image after applying a Gaussian filter. In the scope of the project a parameter was added to set a tolerance for still existing false or error pixels: *threshold*. It's inevitable to do so, due to the fact that there are not only black and white pixels anymore (values 0 and 255) because of the Gaussian filtering but also values in between.

The final step is to find objects in the distance-scaled and Gaussian filtered image. After preparation of the given depth image with Gaussian filtering and logarithmic distance scaling, the final step for finding and detecting objects in range is to find pixel clouds with a defined minimal distance (parameter in algorithm: *minDistanceBetweenObstacles* in meters) to other clouds and therefore to other obstacles.

When a pixel cloud (respectively an obstacle) is detected with no other obstacle in range of *minDistanceBetweenObstacles*, it is marked as a detected obstacle.

## 7. Conclusions

The autonomous driving requires the continuous mapping of the surroundings. The presented approach and the hardware setting show that the mapping can be done with a Kinect camera in the case of model cars in a scale up to 1 : 5. With the 3D capturing of the surroundings, the projection and filtering and obstacle detection, it is possible to get a temporary map of the car's environment as a basis for the navigation process. The navigation itself relies on the A*-algorithm and uses the projected map including the detected obstacles as the digital weighted spatial information. In a next step, we are planning to test the whole configuration and the navigation process under different conditions, i.e. different surfaces or different weather conditions. The aim is to obtain feedback on the use and the performance of the stereo camera and the navigation algorithm. Beyond the self-driving car, the presented approach can be implemented in other vehicles, like unmanned aerial vehicles (UAV), autonomous underwater vehicles (AUV) or even in robots for different services.

## References

[1] A. M. Kessler. Elon Musk Says Self-Driving Tesla Cars Will Be in the U.S. by Summer - The New York Times, 2015.

[2] A. Fisher. Google's Road Map to Global Domination - The New York Times, 2013.

[3] A. Geiger, P. Lenz, R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3354–3361. 2012. doi:10.1109/CVPR.2012.6248074.

[4] P. Fankhauser, M. Bloesch, C. Gehring, et al. Robot-centric elevation mapping with uncertainty estimates. In *International Conference on Climbing and Walking Robots (CLAWAR)*, EPFL-CONF-198746. 2014. doi:10.1142/9789814623353_0051.

[5] S. Thrun, et al. Exploring artificial intelligence in the new millennium. vol. 1, chap. Robotic Mapping: A survey, pp. 1–35. 2002.

[6] M. Herbert, C. Caillas, E. Krotkov, et al. Terrain mapping for a roving planetary explorer. In *IEEE International Conference on Robotics and Automation*, pp. 997–1002. 1989. doi:10.1109/ROBOT.1989.100111.

[7] A. Kleiner, C. Dornhege. Real-time localization and elevation mapping within urban search and rescue scenarios. *Journal of Field Robotics* **24**(8-9):723–745, 2007. doi:10.1002/rob.20208.

[8] S. Thrun, M. Montemerlo, H. Dahlkamp, et al. Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics* **23**(9):661–692, 2006. doi:10.1002/rob.20147.

[9] PrimeSense, 2016. Page Version ID: 710008223.

[10] Correspondence problem, 2015. Page Version ID: 661209488.

[11] S. DESIGN. RC-XD (Call Of Duty: Black Ops) - 3d Warehouse.

[12] t. plus. Kinect from Microsoft - 3d Warehouse.

[13] P. Fankhauser, M. Bloesch, D. Rodriguez, et al. Kinect v2 for mobile robot navigation: Evaluation and modeling. In *International Conference on Advanced Robotics (ICAR)*, pp. 388–394. 2015. doi:10.1109/ICAR.2015.7251485.

[14] M. S. Nixon, A. S. Aguado. *Feature extraction and image processing*. Newnes, Oxford ; Boston, 1st edn., 2002. OCLC: ocm49045100.

[15] L. G. Shapiro, G. C. Stockman. *Computer vision*. Prentice Hall, Upper Saddle River, NJ, 2001.