



## Assignment of bachelor's thesis

<b>Title:</b>	Urban scene recognition and editing II.
<b>Student:</b>	Pavel Kříž
<b>Supervisor:</b>	Ing. Radek Richtř, Ph.D.
<b>Study program:</b>	Informatics
<b>Branch / specialization:</b>	Web and Software Engineering, specialization Computer Graphics
<b>Department:</b>	Department of Software Engineering
<b>Validity:</b>	until the end of summer semester 2022/2023

### Instructions

The thesis goal is to create a tool that allows utilizing geolocation data (accelerometer, gyroscope, etc.) to recognize the urban scene and find a particular position of the user and display a historic building in its right position. The historical appearance of the selected buildings will be placed in the recognized scenes, and the primary 3D models of the buildings of the VMČK project will be placed.

- 1) Research existing approaches for recognizing the content of urban scenes concerning the project's primary goal and including the work of J. Šefčík.
- 2) Analyze the searched methods.
- 3) Design a suitable prototype for determining the exact building location based on historical data and local geolocation data only. Focus on comparing the existing solution and do it.
- 4) Implement a prototype.
- 5) Thoroughly test the resulting solution.
- 6) Discuss the suitability and limitations of the prototype for the Dowry Towns of Czech Queens project.





**FACULTY  
OF INFORMATION  
TECHNOLOGY  
CTU IN PRAGUE**

Bachelor's thesis

## **Urban scene recognition and editing II.**

*Pavel Kríž*

Department of Software Engineering  
Supervisor: Ing. Radek Richtr, Ph.D.

May 13, 2021





---

## Acknowledgements

I want to thank my bachelor thesis supervisor Ing. Radek Richtř, Ph.D. for how he led me to the solutions and gave me answers for my questions. I also want to thank Ing. Jiřĩ Chludil for helping me understand all the technical specifications regarding the *Dowry Towns of Bohemians Queens* project that I needed. Last but not least, I want to thank my father and sister who helped me to correct my grammar and improve other features of my text.



---

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 13, 2021

.....

Czech Technical University in Prague  
Faculty of Information Technology  
© 2021 Pavel Kříž. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

#### **Citation of this thesis**

Kříž, Pavel. *Urban scene recognition and editing II.*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

---

# Abstrakt

Naše práce je součástí projektu *Věnná města českých královen*, jehož cílem je přiblížit historii široké veřejnosti za použití moderních technologií, jako je rozšířená realita. K tomu slouží aplikace pro mobilní zařízení se systémem Android, která za běhu zobrazuje historické prvky do scény fotoaparátu. Tato práce se na jejím vývoji podílí tím, že navrhuje lokalizační řešení pro aplikaci za použití obrazových a geolokačních dat. V úvodu jsou popsány a analyzovány potenciálně použitelné technologie a práce, které v rámci uvedeného projektu naší práci předcházely. Tato práce pak konkrétně navazuje na bakalářskou práci *Rozpoznávání a editace urbanistické scény*. Výsledky předcházející práce jsme adaptovali pro potřeby naší práce, ve které jsme realizovali nový pokročilejší prototyp založený na výpočtu globální polohy zařízení. Nový prototyp je pak testován na funkčnost i na praktickou použitelnost s ohledem na hardwarové možnosti zařízení a porovnáván se stávajícími možnostmi zařízení. V závěru práce je posuzováno zapojení prototypu do mobilní aplikace projektu a jsou předloženy podněty pro další budoucí vývoj.

**Klíčová slova** počítačové vidění, lokalizace, rozšířená realita, OpenCV, obrazové příznaky, Věnná města českých královen

# Abstract

Our work is part of the *Dowry Cities of Czech Queens* project, which aims to bring history closer to the general public using modern technologies such as augmented reality. This is done using an application for Android mobile devices, which displays historical elements in the camera scene at runtime. This work contributes to its development by proposing a localisation solution for the application using image and geolocation data. The introduction describes and analyses potentially applicable technologies and works that preceded our work within the project. This work then specifically follows on from the bachelor's thesis *Recognition and Editing of the Urban Scene*. We adapted the results of the previous work for the needs of our work, in which we implemented a new more advanced prototype based on the calculation of the global position of the device. The new prototype is then tested for functionality and practical applicability with respect to the hardware capabilities of the device and compared with existing device capabilities. At the end of the work, the involvement of the prototype in the mobile application of the project is proposed and suggestions for further future development are presented.

**Keywords** computer vision, localisation, augmented reality, OpenCV, image features, Dowry Towns of Bohemian Queens

---

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Research</b>	<b>3</b>
1.1 From light to the image . . . . .	3
1.1.1 Human perception of colors . . . . .	3
1.1.2 Representation of colors . . . . .	5
1.1.3 Image . . . . .	5
1.2 Image pre-processing . . . . .	6
1.2.1 Gamma correction . . . . .	6
1.2.2 Converting image to grayscale . . . . .	7
1.2.3 Histogram equalisation . . . . .	7
1.2.4 Scaling . . . . .	9
1.2.5 Crop . . . . .	9
1.3 Local image features . . . . .	9
1.3.1 Categories of features . . . . .	9
1.3.2 Local features properties . . . . .	10
1.4 Feature detection and description . . . . .	11
1.4.1 LBP . . . . .	12
1.4.2 SIFT . . . . .	12
1.4.3 BEBLID . . . . .	13
1.5 Feature matching . . . . .	14
1.5.1 Methods of feature matching . . . . .	15
1.5.2 Methods of filtering matches . . . . .	16
1.6 Perspective-n-Point problem . . . . .	16
<b>2 The context of our work</b>	<b>19</b>
2.1 The Dowry Towns of Bohemian Queens project . . . . .	19
2.1.1 History of the dowry towns . . . . .	20
2.1.2 Project goals . . . . .	20

2.2	Urban scene recognition and editing . . . . .	20
2.2.1	Design of Šefčík's work . . . . .	21
2.2.2	Realisation and testing in Šefčík's thesis . . . . .	21
2.3	Dowry Towns of Czech Queens - Core . . . . .	22
2.3.1	Previous state of the project . . . . .	22
2.3.2	Design and realisation . . . . .	24
<b>3</b>	<b>Analysis</b>	<b>25</b>
3.1	Previous theses . . . . .	25
3.1.1	Urban scene recognition and editing I . . . . .	25
3.1.2	DTBQ - Image recognition module . . . . .	26
3.2	Computer vision libraries . . . . .	30
3.2.1	OpenCV . . . . .	30
3.2.2	ML Kit . . . . .	31
3.2.3	ARCore . . . . .	32
3.2.4	Conclusion . . . . .	33
3.3	Programming language . . . . .	34
3.3.1	C++ . . . . .	34
3.3.2	Java . . . . .	34
3.3.3	Python . . . . .	34
3.4	Feature detecting, extracting and matching methods . . . . .	35
3.4.1	Feature detecting and extracting . . . . .	35
3.4.2	Feature detecting and extracting in OpenCV . . . . .	35
3.4.3	Feature matching . . . . .	37
3.4.4	Feature matching in OpenCV . . . . .	38
<b>4</b>	<b>Prototype development</b>	<b>39</b>
4.1	Design . . . . .	39
4.1.1	Programming language . . . . .	39
4.1.2	Configurability . . . . .	39
4.1.3	Database . . . . .	40
4.1.4	Output . . . . .	41
4.1.5	Elimination based on location . . . . .	41
4.2	Implementation . . . . .	41
4.2.1	Input and output formats . . . . .	41
4.2.2	Processing pipeline . . . . .	44
4.2.3	Configurability . . . . .	46
4.3	Experimental evaluation . . . . .	47
4.3.1	Methodology . . . . .	47
4.3.2	Results . . . . .	48
4.4	Installation . . . . .	52
4.4.1	Installing OpenCV . . . . .	52
4.4.2	Installing Boost C++ Libraries . . . . .	54
4.4.3	Running and compiling our prototype . . . . .	55



<b>5</b>	<b>Future outlook</b>	<b>57</b>
5.1	Contextualization . . . . .	57
5.2	Design of the localisation module . . . . .	57
5.2.1	Integration of the already finished code . . . . .	60
5.3	Future improvements proposals . . . . .	60
5.3.1	Future modules data requirements . . . . .	61
	<b>Conclusion</b>	<b>63</b>
	<b>Bibliography</b>	<b>65</b>
<b>A</b>	<b>List of used abbreviations</b>	<b>69</b>
<b>B</b>	<b>Contents of the attached data storage</b>	<b>71</b>
<b>C</b>	<b>Test results</b>	<b>73</b>



---

## List of Figures

1.1	Gamma corrections examples with different gamma values . . . . .	6
1.2	Example of histogram equalisation . . . . .	8
1.3	Examples of all possible features . . . . .	10
1.4	Demonstration of a feature with good repeatability . . . . .	11
1.5	Illustration of creating the BEBLID descriptor . . . . .	14
1.6	Illustration of feature matching . . . . .	15
1.7	Probability density function (DF) of correct and bad matches . . .	17
1.8	Illustration of the Perspective-n-Point problem . . . . .	18
2.1	Location of dowry towns in the Czech Republic . . . . .	19
2.2	Activity diagram of prototype realisation of Scene Recognition and Editing bachelor thesis by Šefčík . . . . .	22
2.3	Scheme of the whole DTBQ project . . . . .	23
3.1	Performance of Šefčík's prototype . . . . .	27
3.2	Localisation module class diagram of the thesis named <i>DTBQ - Image recognition module</i> . . . . .	29
3.3	Result of a demo from OpenCV tutorial on how to find known object location in picture . . . . .	31
3.4	ML Kit <i>image labeling</i> example . . . . .	31
3.5	Example of ML Kit <i>detection and tracking</i> . . . . .	32
3.6	Example of an application based on ARCore . . . . .	33
3.7	Early evaluation of our prototype. . . . .	36
4.1	Illustration of creating a reference building image . . . . .	42
4.2	Activities diagram of our prototype for default configuration . . .	45
4.3	Illustration from side of a camera coordinates correction. . . . .	46
4.4	Testing results of the prototype . . . . .	49
4.5	Results of understanding of the spatial object/building characteristics	50
4.6	Example of building's reference image size importance . . . . .	51

4.7	Detection of background building instead of foreground object . . .	52
5.1	Visualisation of the localisation with a 3D model of a tower Kropáčka	58
5.2	Sequence diagram of the communication between the application, localisation module and native code . . . . .	59
5.3	Possible gain of accuracy in case of improving reference images . .	61
C.1	Test result 1 . . . . .	74
C.2	Test result 2 . . . . .	74
C.3	Test result 3 . . . . .	74
C.4	Test result 4 . . . . .	75
C.5	Test result 5 . . . . .	75
C.6	Test result 6 . . . . .	75
C.7	Test result 7 . . . . .	76
C.8	Test result 8 . . . . .	76
C.9	Test result 9 . . . . .	76
C.10	Test result 10 . . . . .	77
C.11	Test result 11 . . . . .	77
C.12	Test result 12 . . . . .	78
C.13	Test result 13 . . . . .	78
C.14	Test result 14 . . . . .	79
C.15	Test result 15 . . . . .	79
C.16	Test result 16 . . . . .	79
C.17	Test result 17 . . . . .	80
C.18	Test result 18 . . . . .	80
C.19	Test result 19 . . . . .	80
C.20	Test result 20 . . . . .	81
C.21	Test result 21 . . . . .	81
C.22	Test result 22 . . . . .	82
C.23	Test result 23 . . . . .	82
C.24	Test result 24 . . . . .	83
C.25	Test result 25 . . . . .	83

---

## List of Tables

3.1	Comparison of OpenCV feature detection and description methods	37
3.2	Comparison of two matching methods. . . . .	38



---

# Introduction

Augmented reality is a term for the method of extending a real world image by some computer-generated content. The difference of augmented reality to virtual reality is that the immersion to the computer-generated content is significantly lower and instead of creating a new world it rather extends the already existing one. Nowadays, the most common form of augmented reality is inserting content into the image of a smartphone camera in real time. In the *Dowry Towns of Bohemians queens* project one of the goals is to create a mobile application that will allow to display 3D models in the scene of the mobile device. The application will display models of historical buildings and other objects. By doing that will the application create a view to the history of the place where the application will be used. Such a window to the history will allow seeing buildings that were already demolished or their look was changed significantly in their original form at the original places.

In that point occurs a problem for which a solution has to be found. The historical buildings cannot be displayed randomly all over the place in the scene of the smartphone camera. The buildings have to be placed on the exactly determined positions where they have stood before so the display has a historical value and accuracy. In order to place the buildings into the correct position the precise location of the smartphone has to be known. Nowadays positioning (GPS) sensors guarantee accuracy in meters and sometimes they lose the location accuracy completely. But the accuracy of the positioning is key to place the buildings in correct positions and a few meters make a big difference especially in case of smaller buildings.

The solution of that problem is searched in the area of computer vision. When the building and its location were recognised in the image of the smartphone camera, the location towards the building could be calculated. Prototype solution of the image recognition was already created in the preceding thesis *Urban scene recognition and editing* [1]. But beside the solution of the image recognition it is also needed to create a solution for calculating the global location or relative location towards the recognised building.





---

# Research

In this chapter are explained the theoretical basics of the image processing that we consider to use in this project. In our project the image processing includes image pre-processing, image identification and comparison of the images based on their identification. At the end we explain the perspective-n-point problem.

## 1.1 From light to the image

In this section it is explained what colors are and how to represent them digitally or what is the image considered to be. More detailed description of the following topics can be found in the Czech learning book [2]. This whole section is based on that book.

### 1.1.1 Human perception of colors

All known kinds of electromagnetic radiation like x-ray, microwave or visible light are included in the electromagnetic spectrum. Specifically, visible light, also called the visible spectrum, is a radiation with wavelengths in the range of about 380 to 720 nm. Specific color is then radiation with specific wavelength from the visible spectrum. Light with wavelength about 550 nm is perceived as green and light with wavelength 720 as red. The so-called white light is then composed of rays from the entire visible spectrum (with all possible visible wavelengths). Such light is produced by the sun or a light bulb and is called neutral light (achromatic light). When light hits a surface, some rays of light are absorbed and some are reflected depending on the physical properties of the surface. The resulting composition of the reflected rays creates what we understand as color of the surface. The dominant frequency of light influences how the color is perceived by humans. For example, a color with a majority of rays with frequencies about 550 nm would be called green.

The following terms are important for describing colors:

**Luminance** is a physical quantity describing the amount of light.

**Brightness** is a subjective perception of the light intensity.

**Saturation** purity of the light color is described by this term. The tighter is the frequency spectrum of light the more saturated the color is.

**Lightness** in light with some dominant frequency is the lightness, the amount of neutral light.

### Human eye

Rod cells and cone cells are sort of sensors in the human eye which are responsible for perception of light and color. The rod cells are approximately ten times more sensitive to light than the cone cells and thus the rod cell's purpose is mainly to measure the luminance. The cone cells complement the rod cells in a way that they measure the colors.

Each cone cell has one of three different photopigments. Each photopigment is more or less sensitive to specific wavelengths. Blue photopigment is a photopigment that is most sensitive to light rays with a wavelength about 445 nm. Green and red is then most sensitive to rays with wavelengths about 535 nm or 575 nm respectively. Green and red photopigments, compared to blue ones, are more sensitive to the whole visible spectrum. The photopigments are also not evenly distributed among the cone cells. In the eye there are the most cone cells with red photopigment, bit less with green one and the least with blue one.

### Color perception

In the eye, colors are perceived as three basic colors: red ( $R$ ), green ( $G$ ) and blue ( $B$ ) but the information about the colors is delivered to the brain in different forms. In the connection between the eye and the brain, the optic nerve, the colors are combined in the following way: The color information is represented in two channels. The first one is made by the subtraction of blue from yellow  $Y - B$  and the second one by subtraction of green from red  $R - G$ . There is also a third channel created as an addition of red to green  $G + R$  and the channel indicates the brightness.

### Eye adaptation to luminance

An important feature of the human eye is the adaptation to luminance. It is a phenomenon of perceiving the light intensity changes (seeing surfaces with different brightness in one view). The light intensity change is not perceived linearly by humans but logarithmically. Thus, the change of light intensity is perceived as bigger in lower light intensities than in the higher ones. Thanks to

the luminance adaptation, the human eye is able to perceive a wide luminance range.

### 1.1.2 Representation of colors

Colors are digitally represented in various ways, which are called color models. All colors that can be represented by a color model, create a color space. One of the most used color models is RGB. The name of the model is an abbreviation derived from the three colors: red, green, and blue. These colors are considered basic colors and they are defined as a specific color in the spectrum of colors visible by a human. In RGB, colors are expressed as a combination of the basic colors. RGB is an additive color model. This means that by adding intensity in each channel, we get a brighter color. So the maximum intensities of basic colors are combined to create white and the minimum intensities to create black. Value in each channel is usually represented in range  $\langle 0, 1 \rangle$  as a floating-point number or it is represented as an integer in range  $\langle 0, 255 \rangle$ . The last case is related to the fact that the value is usually stored on 8 bits. A maximum of 256 different values can be stored on 8 bits ( $2^8 = 256$ ).

There are many other color models and spaces. One of them is the color space CIE 1976 L\*a\*b\* (also called CIELAB), that is closer to how humans perceive colors than RGB. This color model also has three channels but one of them stores the lightness of the color (channel L\*). The other two channels are axis in the color space, the first one is an axis that goes from green to red (channel a\*), and the second one goes from blue to yellow (channel b\*). There are other models similar to this one, such as the YUV color model. This color model also has three color channels (Y, U and V) from which two (U and V) determine the chrominance (color) and the third channel (Y) stores the brightness.

### 1.1.3 Image

In society, an image is usually understood as a painting, a drawing, or what people see on the screen, or as light that comes into the human eye. Mathematically, however, we can define an image with a continuous function  $f$  with two variables  $x$  and  $y$ ,  $x, y \in R$ . Such a function is called an image function and can be defined as follows:

$$z = f(x, y) . \tag{1.1}$$

Such a function considers the infinite size of the image, which is not practical. Finite image function with range  $I$  within some specific boundaries  $\langle x_{min}, x_{max} \rangle$  and  $\langle y_{min}, y_{max} \rangle$  can be defined as follows:

$$f : (\langle x_{min}, x_{max} \rangle \times \langle y_{min}, y_{max} \rangle) \rightarrow I . \tag{1.2}$$

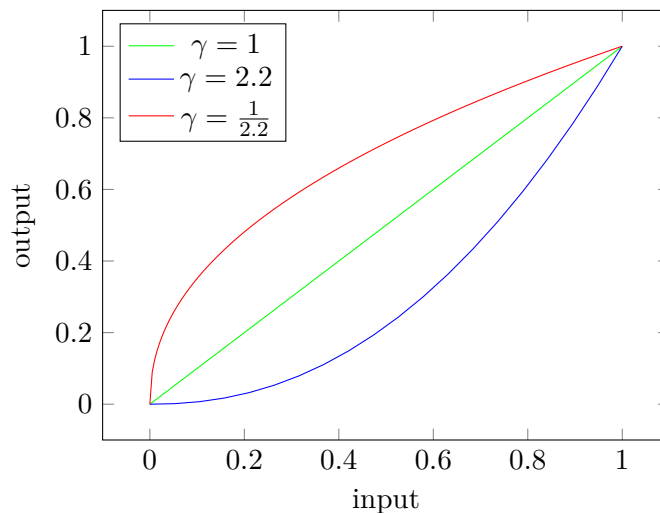


Figure 1.1: Gamma corrections examples with different gamma values

To represent the image digitally, the variables  $x, y$  have to be integer numbers. For discrete image functions, the value  $z$  is called *pixel* (picture element). It is also important to mention that the value  $z$  does not have to be a number. In the case of RGB, the value  $z$  would be a vector made of three numbers.

## 1.2 Image pre-processing

It is appropriate to think about some pre-processing of the image before the main image processing. In our case, such pre-processing means to put the image in a state in which the undesirable influences are diminished as much as possible. Ideally, the image should be in the same state after the pre-processing independently on the conditions in which the image was taken.

### 1.2.1 Gamma correction

Gamma correction is how is called a following non-linear operation [3]:

$$V' = V^\gamma, \quad (1.3)$$

where  $V'$  is the output and  $V$  is the input and  $\gamma$  is some real number. If  $\gamma < 1$ , then such an operation is called gamma compression and the  $\gamma$  is called *encoding gamma*. If  $\gamma > 1$ , the operation is then called gamma expansion and the  $\gamma$  is called *decoding gamma*. Examples of some gamma corrections may be seen in figure 1.1.

As mentioned above, the luminance is not perceived linearly by the human eye, but logarithmically, so the human eye is more sensitive to changes in darker tones of colors in comparison to lighter ones. Therefore, gamma

correction is used. When the data is stored using gamma compression, then more possible values are given for darker tones. This processing has the advantage that only 8 bits instead of 11 are needed to store the same amount in the spectrum of dark shades, while maintaining the same visual quality. But in the end it is needed to transfer the data back to the linear space by processing all the values with the gamma expansion [3].

### 1.2.2 Converting image to grayscale

There are several color models that have one channel determining the luminance of the color. Such color models are YUV for example. When the Y channel is separated, a grayscale image is created. But not all color models have such simple conversion.

When converting colors from the RGB color space, it is important to know that different colors contribute to the perceiving of the brightness of the color with different amounts. This is due to the distribution of photopigments among the cone cells, as explained above. Thus, when the colors are converted, specific weights are used in a way that gives us the following formula [4]:

$$Y_{lin} = 0.2126 \cdot R_{lin} + 0.7152 \cdot G_{lin} + 0.0722 \cdot B_{lin} , \quad (1.4)$$

where  $Y_{lin}$  a  $R_{lin}, G_{lin}, B_{lin}$  are values of lightness of the basic colors stored in the linear space.

If the colors are stored in gamma compressed space, the values are first transferred back to the linear space by using gamma expansion. Then is done the conversion to the grayscale and the shades of gray are gamma compressed again [5]. Different weights may be used for the conversion for different standards. The weights are also not scientifically determined, but determined experimentally, so they provide satisfactory results. For simplicity, a linear conversion from a non-linear color space to non-linear space of shades of grey is sometimes used [4].

### 1.2.3 Histogram equalisation

Histogram equalisation is a method to change the global contrast in an image. A histogram is a distribution of the values in the image. Contrast can be explained as the difference of values between two objects (points, areas or lines) [6]. If the contrast is too low, it is difficult to distinguish two objects in image, see the picture 1.2a with the histogram in the image 1.2b. The values are crammed in the middle of the histogram, leading to smaller differences between the values in the image. After applying the equalisation, the values are evenly distributed among the range of values, see the picture 1.2c with the histogram 1.2d. The image with an equalised histogram has a higher contrast

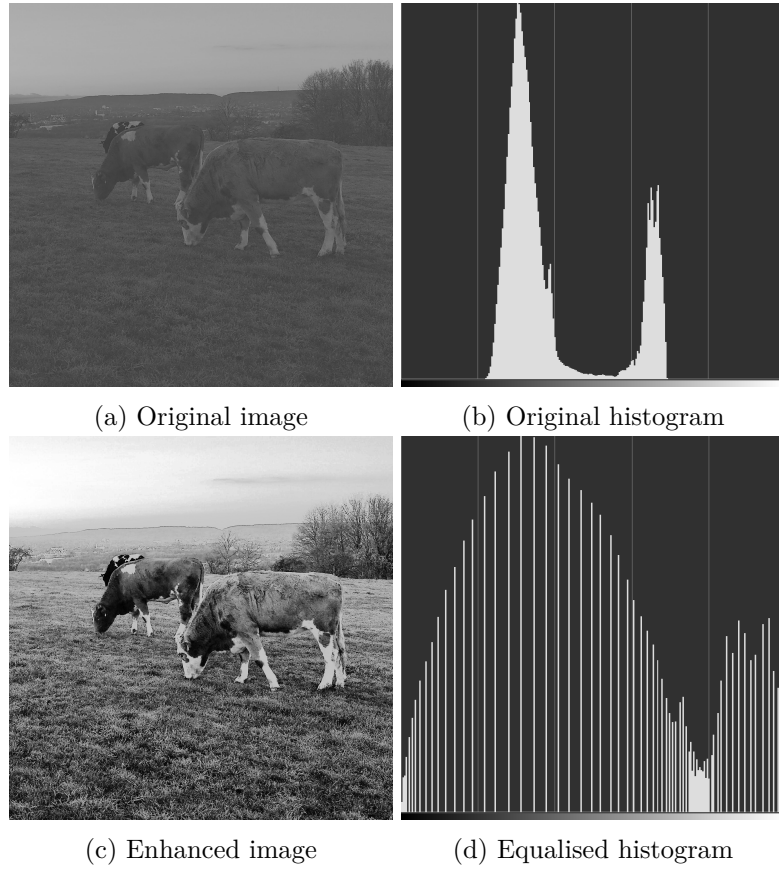


Figure 1.2: Example of histogram equalisation. Photo: author

than the original image. Increasing the contrast helps the texture and objects in the image to stand out more.

For an image of size  $M \times N$ , where  $f(x, y)$  is a pixel at coordinates  $x, y$ , is the image with an equalised histogram computed as follows. In the histogram  $H$ , the quantity of each value from the range  $L$  is placed as follows [6]:

$$H(i) = \sum_{x=1}^M \sum_{y=1}^N \begin{cases} 1 & f(x, y) = i \\ 0 & \text{otherwise} \end{cases} . \quad (1.5)$$

The pixel  $g(x, y)$  in the output picture at coordinates  $x, y$  is then calculated as follows:

$$g(x, y) = \frac{CDF(f(x, y)) - CDF_{min}}{M \cdot N - CDF_{min}} \cdot (L - 1) , \quad (1.6)$$

where  $CDF$  is the cumulative distribution function with minimum  $CDF_{min}$  which is defined as follows:

$$CDF(j) = \sum_{i=1}^j H(i) . \quad (1.7)$$

The explained process takes into account the global histogram for the whole image. Sometimes, such an adjustment is not sufficient and it is necessary to use a more complex method. A method created by a simple modification in this method, which gives significantly better results, is called the adaptive histogram equalisation (AHE). The adjustment is not to use the global histogram, but a local histogram created from the neighborhood around the calculated pixel. Another additional option of modification is the contrast limited adaptive histogram equalization (CLAHE), which does what AHE does, but also clips the quantities to a certain limit [6].

#### 1.2.4 Scaling

Scaling is such image processing in which the size of the image is changed.

#### 1.2.5 Crop

Crop of the image is a method of removing unwanted parts of the image. By using crop the wanted part of the image stands out more.

### 1.3 Local image features

Local features are such a pattern that is clearly distinguishable within its closest neighborhood. Usually, the feature is some change of a property of an image, despite the fact that the feature does not have to lie directly on the change. Usually, the properties of an image are: luminance, color or texture. Different examples of features can be seen in the image 1.3. The features may be edges or small image patches as can be seen in the left outline drawings. Features are often points, such as points (marked with small circles) in the image on the right. The features neighborhood is usually described after the detection of the features in the image and the products of that are the descriptors of the features. The process of describing the features is then called description or extraction of features. The descriptors are then what is usually used for various purposes [7].

The features are sometimes also called keypoints, or interest points. The image features are studied for a long time and the first paper was published in 1954. Feature detection has evolved a lot since then. One of the interesting properties is that the combination of features detected on straight lines and rectangular corners is often an indication of structures made by humans [7].

#### 1.3.1 Categories of features

There can be three categories of use of algorithms that have something to do with features detection. This is not the only way to categorise these algorithms, but their categorisation in this way helps to determine which proper-

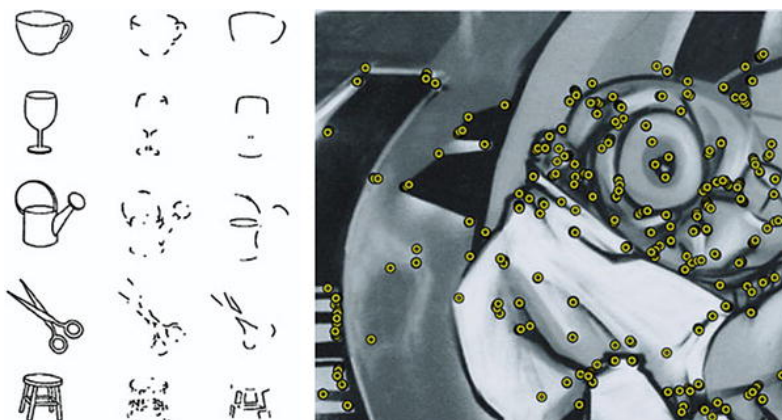


Figure 1.3: Examples of all possible features. Features might be lines or surfaces as in outline drawings or points as in the grayscale image. Illustration is taken from [7].

ties of features are important in which situations. The first case of use (first category) is when meanings are assigned to the features. A good example is when it is assigned the meaning of being roads or railroads to the line features in the aerial photos. In the second case of use, the features are used as anchors. Such features have to be reliably detected (all the time at the same location) among multiple detections. Good examples of such use are searching for matches, object tracking or creating panoramas. In the third category, the features are used as a representation of the content in the image. Here it is not the meaning of individual features or accurate location that is important, but rather the properties of all the features together as a whole. Categorization of the image content is a good representation of this category [7].

### 1.3.2 Local features properties

Local features are such features that have a specific spatial meaning in regard to their neighborhood. Such neighborhoods may be any area in the image. As a result, the local features have reasonable invariance across unfavorable visibility conditions. Ideally, however, the features should represent some meaningful geometric objects. But this is complicated and it is not even considered nowadays. However, quality algorithms of feature detection should have the following properties (not necessarily all of them at the same time) [7]:

**Repeatability** If there are multiple photos of the same object and a feature is detected in one picture, then the feature has to be also detected in other photos (if the feature is properly visible). There are two ways to achieve repeatability. The first way is the invariance of transformations, such as rotation or scaling. The second way is the robustness, which is





Figure 1.4: Demonstration of a feature (highlighted with the circle) with good repeatability. Feature is detected despite different angles of view, noise and slight rotation. Photo: author

meant as insensitivity to noise or other minor influences. Examples of both ways and high repeatability can be seen in image 1.4.

**Distinctiveness** Feature with this property can be well distinguished from other features detected in the image or in other images. To some extent, this property is in contradiction to invariance and robustness. The invariance and robustness are usually done by ignoring some information and that leads to less information by which the feature can be distinguished.

**Locality** Is the property of the feature depending only on the neighborhood of the feature.

**Quantity** The number of features should be big enough to cover sufficiently even small objects in the image. In some cases the high number of features is not convenient so a limit of the features quantity has to be set.

**Accuracy** If the same feature is detected in two different images, then the location has to be in the same place in relation to the spatial configuration of the two images.

**Efficiency** Short computation time is preferred especially in real-time applications.

## 1.4 Feature detection and description

In our thesis, several feature detection and description methods are briefly described. A more detailed description of some of these and other methods can be found in the previous bachelor thesis [1].

### 1.4.1 LBP

Local binary patterns (LBP) are features that are calculated for each pixel in the image as follows [8]:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c) 2^p \quad \text{where } s(x) = \begin{cases} 1 & x \leq 0 \\ 0 & x > 0 \end{cases}, \quad (1.8)$$

$g_p$  means an adjacent point (the point has the value of the pixel on which it is) value with index  $p$ ,  $g_c$  is the value of the evaluated pixel (central point). The adjacent points lie evenly on a circle around the central point. Parameters  $P$  and  $R$  determine the number of adjacent points and the radius  $R$  (measured in pixels) of the circle. For  $R = 1$  the adjacent points lie in the direct neighborhood pixels of the evaluated pixel.

A partial rotation invariance can be created by applying a bit shift to the right on all the calculated values. The right shift is applied until 1 is in the least significant position. With the help of this modification, the LBP is invariant to rotations of multiples of the value  $360/P$ .

### 1.4.2 SIFT

Scale-invariant feature transform (SIFT) was patented by University of British Columbia [9] and published by David Lowe [10]. The patent expired in the year 2020.

Further description of the SIFT is freely taken from work [11]. SIFT is invariant to lighting changes, orientation, uniform scaling and partially invariant to affine transformations. These properties guarantee robustness preventing many negative influences. Although it is an old method, it has good results compared to other methods according to this comparison study [12].

The output of the SIFT algorithm is a list of  $N$  keypoints. The way of computing the keypoints are the following steps:

- Creation of images in many different scales created from the original image. This process is often called a creation of an image pyramid. The levels of this pyramid are the images in different scales,
- Detecting extremes across all the images from the pyramid. The detected points are candidates of being keypoints,
- Keypoints localisation,
- The canonical rotation is assigned to these keypoints,
- Generation of descriptors of these keypoints.

The output structure of keypoints  $x_i$  is defined as follows  $x_i = \{x, y, \sigma, o, f\}$ , where  $x$  and  $y$  are the coordinates of the keypoint in the image,  $\sigma$  is the scale

in which the keypoint was calculated,  $o$  is the canonical orientation,  $f$  is the SIFT descriptor. The descriptor is a vector of 128 floating point numbers.

### RootSIFT

RootSIFT is a modification of the SIFT algorithm that increases the accuracy without greater memory storage requirements or significantly higher processing cost [13]. The modification of the algorithm is not complicated and it consists of a few steps after the keypoints description. The steps are as follows:

**First step** The first step is to L1 normalise the SIFT descriptors (originally they are L2 normalised).

**Second step** Square root all the descriptors

**Third step** L2 Normalise all the descriptors.

L1-normalised vectors must meet following condition:

$$\sum_{i=0}^n x_i = 1 \text{ and } x_i \geq 0. \quad (1.9)$$

L2-normalised vectors must meet following condition:

$$\sum_{i=0}^n \sqrt{x_i^2} = 1. \quad (1.10)$$

Descriptors in this way are then called as RootSIFT descriptors, and comparing them the same as SIFT descriptors leads to comparing them effectively by the Hellinger kernel. According to the original work that leads to an improved accuracy. The Hellinger kernel for two L1-normalised histograms(vectors) is defined as follows:

$$H(x, y) = \sum_{i=0}^n \sqrt{x_i y_i}. \quad (1.11)$$

Resulting number is a coefficient that can be used for the comparison.

### 1.4.3 BEBLID

Boosted efficient binary local image descriptor (BEBLID) is a feature descriptive method that is a binary modification of Boosted efficient local image descriptor BELID (based on real values) [14]. According to the experiments in the study, where the BEBLID was introduced [14], it beats the accuracy and speed of other top performing descriptors ORB, BinBoost and LATCH at the time (BEBLID was published in 2020).

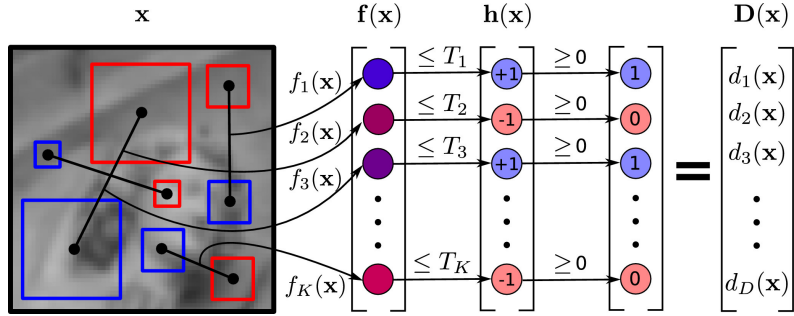


Figure 1.5: Illustration of creating the BEBLID descriptor for image patch  $x$  around detected feature. The image was taken from the BEBLID paper [14].

The feature detection works on comparing pairs that are created from the learned sampling patterns. Sampling patterns are predetermined pairs of points to be compared around the place that is going to be described. There is also a process that creates the descriptors invariant to Euclidean transformations.

In the whole process, a term weak learner is used, which means a function that creates responses to an image patch around a feature. The responses are marked  $h(x)$  and they create a vector of  $K$  learning responses to the weak learner and  $h_k(z) \equiv h_k(x; f, T)$  is called the  $k$ -th weak learner. The weak learner depends on the feature extraction function  $f$  and threshold  $T$  and the  $h_k(x)$  is defined as follows:

$$h(x; f, T) = \begin{cases} 1 & \text{if } f(x) \leq T \\ -1 & \text{if } f(x) > T \end{cases} . \quad (1.12)$$

This weak learner is defined for the BELID algorithm and to modify it to make it create binary outputs,  $-1$  has to be mapped to  $0$ . The feature extraction function is based on comparing pairs of mean grey values of areas around pixels. These pairs are determined by the sampling patterns. The descriptor BEBLID is then vector created by the weak learner, as seen in the figure from the original study 1.5

## 1.5 Feature matching

Feature matching is usually a process of finding the best matches for features from the reference set of features in the second set of features. Not only one best match has to be searched, some processing needs more than only the best match. Sometimes it is necessary to have  $k$  the best matches (sorted according to their similarity with the reference feature). The output of this process is a set of best matches from the second set of features for each feature from the reference set of features. Descriptors are usually used for the matching, rather



Figure 1.6: Illustration of feature matching between two images of the same building taken from different angles. The green ones are matched correctly and the red one is not correct. Photo: author

than the keypoints themselves [15]. Illustration of such a process can be seen in the image 1.6.

If the scenario in the image 1.6 compares the content of both images, then the result will probably be that similar content was found in both images. Filters can be applied to the output matches in order to reduce the number of miss matched features (as the red match in the picture 1.6). More on that topic can be found in the previous work [1].

### 1.5.1 Methods of feature matching

A match for two features is such a pair in which the features are the most similar ones from two sets of features. The Euclidean distance is used in comparing the SIFT features [16]. For binary-based descriptors (like ORB), the Hamming distance is usually used [17]. The smaller the distance, the more features are considered to be similar. Features with zero distance are then considered identical [16].

#### Brute force matching

Straightforward and simple way of feature matching is to use brute force. When using this method, each feature from the first set of features is compared to each feature from the second such set [17]. Because the time complexity is quadratic, time increases rapidly with the number of features.

#### FLANN based matching

For faster matching of large sets of features, a method based on a freely accessible library called Fast Library for Approximate Nearest Neighbors (FLANN)

is usually used. With this method, the match is found with the use of hierarchical structures (randomized kd-trees, hierarchical k mean trees or hierarchical clustering algorithm for ORB). It is important to note that for some structures the search does not guarantee the best match. It produces an approximation of the best match [18].

### 1.5.2 Methods of filtering matches

As mentioned above, some features might be miss matched. In this section it is explained how it is possible to try to remove these mismatches.

#### Cross-check Test

A filter with a simple but working approach is the Cross-check Test. The idea is to match the features in two directions. First, searching the best matches for features from the first set of features in the second set, and then doing the same in the opposite direction. Only the matches that have been found in both directions are not filtered out [17].

#### Lowe's ratio test

Another method of filtering matched features is the Lowe's ratio test. This method introduced in the study [19] filters the matches based on the ratio the closest match distance to the second closest match distance. For this filter, it is necessary that at least two found matches per feature. If the ratio of distances  $\frac{\text{best match}}{\text{second best match}}$  is greater than some threshold, the match is filtered out. This filter guarantees that the match is the best by large margin, so there is not any similar possible match. Lowe states in his study that the threshold should be set between 0.7 to 0.8, as can be seen in the image 1.7. There is a graph of multiple tests of this filter and its results.

#### RANSAC

Random Sample Consensus (RANSAC) is a method used for interpreting and smoothing data containing a large number of gross errors. It is important to note that this is not a filtering method, but it rather just extracts the correct interpretation of the data as a whole. This method is often used in computer vision because the feature detection algorithms are prone to errors [20].

## 1.6 Perspective-n-Point problem

Perspective-n-Point problem (usually referred only as PnP problem) is about finding the correct transformation of points from world space to local camera space when specific information is known. The information must contain the geometry of the points in world space (3D coordinates) and where they have

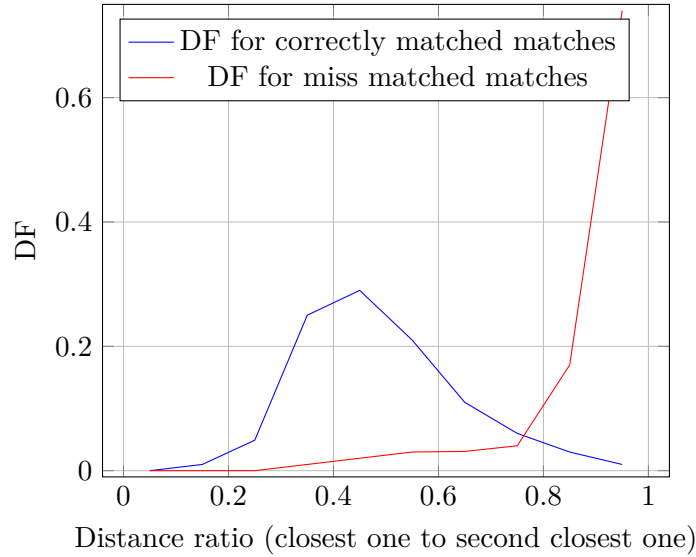


Figure 1.7: Illustration of probability density function (DF) of correct and bad matches depending on the ratio measured by D. Lowe in the study [19].

been projected in the image taken by the camera (2D coordinates). The focal length of a camera, the sensor size, and the resolution has to be known as well [21].

An illustration of the problem can be seen in the image 1.8. Where  $c_i$  are 3D points in world space,  $u_i$  are their projections, the translation matrix  $t \in \mathbb{R}^{3,1}$  and rotation matrix  $R \in \mathbb{R}^{3,3}$  create together the resulting transformation [21].

To project the 3D points into the image, the transformation has to be used with the projection matrix  $A$  in the following way:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \quad (1.13)$$

The final step is to divide the resulting vector by the  $s \in \mathbb{R}$ . The matrix  $A$  can be created from the information about the camera as follows:

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad c_x = \frac{w}{2}, \quad c_y = \frac{h}{2}, \quad f_x = \frac{w \cdot f}{s_x}, \quad f_y = \frac{h \cdot f}{s_y}, \quad (1.14)$$

where  $w$  is a width,  $h$  is a height of the image resolution,  $s_x$  is a size of the camera sensor in horizontal direction,  $s_y$  is then a size in vertical direction and  $f$  is the focal length of the camera [21].

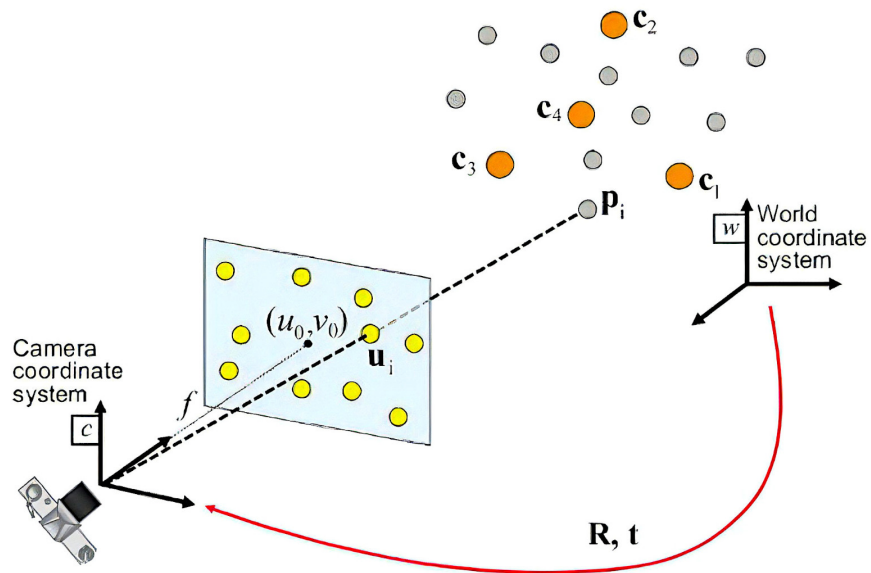


Figure 1.8: Illustration of the Perspective-n-Point problem. Image was taken from OpenCV tutorial [21].



---

## The context of our work

This chapter describes the context of our thesis within the *Dowry Towns of Bohemian Queens* project (further called also as DTBQ). Many theses were created within this project. Two of these and the project itself are introduced in this chapter.

### 2.1 The Dowry Towns of Bohemian Queens project

This project was named after a group of towns that were owned by Czech queens in the past. The institution of the towns owned by Czech queens was established at the beginning of the 14th century and ceased to exist in 1918. In group of these towns belong Hradec Králové, Chrudim, Vysoké Mýto, Polička, Jaroměř, Dvůr Králové, Trutnov, Nový Bydžov and Mělník. As can be seen in the map 2.1, the towns except Mělník are located in the east of Bohemia [22].



Figure 2.1: Location of dowry towns in the Czech Republic. Image is taken from website of the DTBQ project [22].

### 2.1.1 History of the dowry towns

Several queens had their court in Mělník. One of them was Queen Barbora Celská, the wife of King Zikmund Lucemburský, or Queen Johana z Rožmitálu, the wife of King Jiří z Poděbrad. Hradec Králové was a favorite town of Queen Alžběta Rejčka, the first queen to have dowry towns. She got the towns when her husband Václav II registered her 20 000 hryvnias of silver on the selected East Bohemian towns before his death. Hradec Králové was also a favourite town of Queen Alžběta Pomořanská, the wife of King Karel IV. However, she was very unpopular in the town because she created big debts on the account of the town. Thanks to this rich history, the dowry towns are strongly connected to the Czech history and until nowadays they are considered to be part of the Czech regional identity [22].

### 2.1.2 Project goals

The goal and content of the project is to study the presentation of dowry towns with the help of modern technologies of computer graphics and tools of historical geography. The main outputs of the project will be mobile application and a website that is planned to be used as a historic guide for the dowry towns. On the website, users will be guided through the urban landscape of dowry towns using specialised maps of the entire region or each town. A 3D reconstruction of buildings, which stood or still stands, for Hradec Králové will be created and offered to the users in virtual reality with examples from the life of people in the past. Set of products will be published to the public to help to inform about dowry towns and popularize the history of these towns [22].

## 2.2 Urban scene recognition and editing

The *Urban scene recognition and editing* thesis, on which this one continues, is written by Jan Šefčík [1]. Within the DTBQ project, this work deals with technical problems in the field of smartphone application, especially with the problem of localisation based on computer vision. Based on this, building images can be displayed in the scene of smartphone cameras.

Main goals of the thesis are established as follows:

- Researching the possibilities of urban scene recognition,
- Analyse the found possibilities and their restrictions within the DTBQ project,
- Design and realisation of a prototype that determines the location using the geolocation and image data,

- Testing the realised prototype and comparing it to the limitations within the DTBQ project.

The first two goals, researching and analysing methods of image pre-processing, feature detection and extraction, and feature matching and filtering are met in the theoretical part of the thesis. Their suitability within the DTBQ project is discussed too and in the chapter of analysis, there are some useful notes for improvements.

### 2.2.1 Design of Šefčík's work

The practical part of Šefčík's thesis is devoted to a mobile application design and a realisation. Application properties and functional and non-functional requirements are specified within the design.

The requirements can be simplified as follows:

**Functional requirements:** The application runs in real-time, weather adaptation, accuracy of the localisation, easy administration of the database of the historical objects, saving data in the application.

**Non-functional requirements:** The application runs on the Android operating system version 9.0 and higher, modularity, quality documentation in the form of technical documentation and documentation for programmers, user manual and video manuals for administrators.

### 2.2.2 Realisation and testing in Šefčík's thesis

The design included a much wider area than the realisation itself. The state of finished implementation is well illustrated by the activity diagram in the image 2.2. The realised implementation is therefore a prototype of the image recognition and localisation of the object in an image.

The prototype was implemented using the Python programming language and OpenCV, the library for computer vision, and the Pillow library. In the implementation, the feature detecting and extracting method SIFT and the matching based on the FLANN were used. The Lowe's ratio test and RANSAC were used to filter and smooth out the matching output.

A simple file system database of images with buildings of different architectural styles was created to test the prototype. Šefčík's thesis was concluded with that prototype testing, where the performance of the prototype was evaluated. The importance of image pre-processing was shown in the testing and the good results of CLAHE were specifically mentioned there as the only worthy method of pre-processing. It was also mentioned that using the RANSAC algorithm helps a lot with recognition accuracy. A big advantage of the RANSAC algorithm was that it neglected the bad matches that are created in repeated patterns (same windows on building).

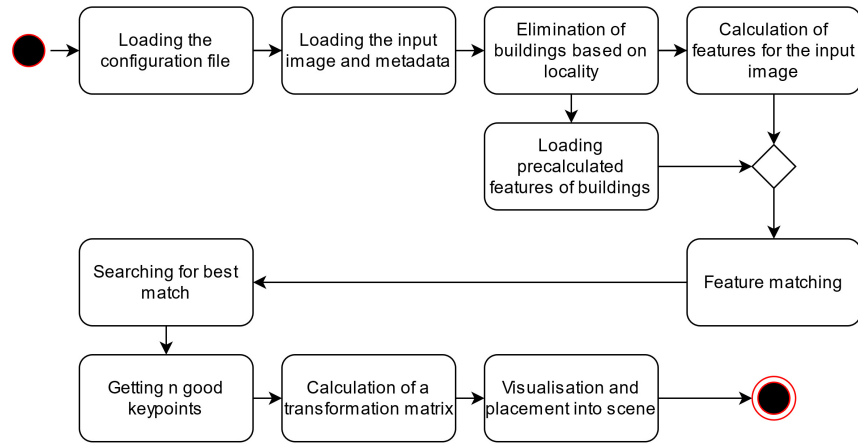


Figure 2.2: Activity diagram of prototype realisation of Scene Recognition and Editing bachelor thesis by Šefčík. The diagram is translated from the Šefčík’s original [1].

## 2.3 Dowry Towns of Czech Queens - Core

The *Dowry Towns of Czech Queens - Core* thesis by Daniel Vančura (in original called *Věnná města českých královen - jádro*) [23] is important work for the project. This section describes the parts of Vančura’s bachelor thesis in regard to ours. His thesis is then divided into chapters, in which the first chapters describe the current state before the beginning of his works and important technologies in the project. The next chapters talk about design and realisation.

The main goals of Vančura’s thesis have been:

- Finishing the application programming interface (API),
- Refactoring the data storage,
- Creating documentation for the entire technical area of the project. Overall, the thesis has to be the documentation itself for other students working within the DTBQ project.

### 2.3.1 Previous state of the project

Among others, a principle of 3D model approval and creation is explained in Vančura’s thesis. The 3D models are models of selected historical buildings that will be later prepared for the display in virtual and augmented reality. Before that, these models have to be first created with highest historical accuracy.

### 2.3. Dowry Towns of Czech Queens - Core

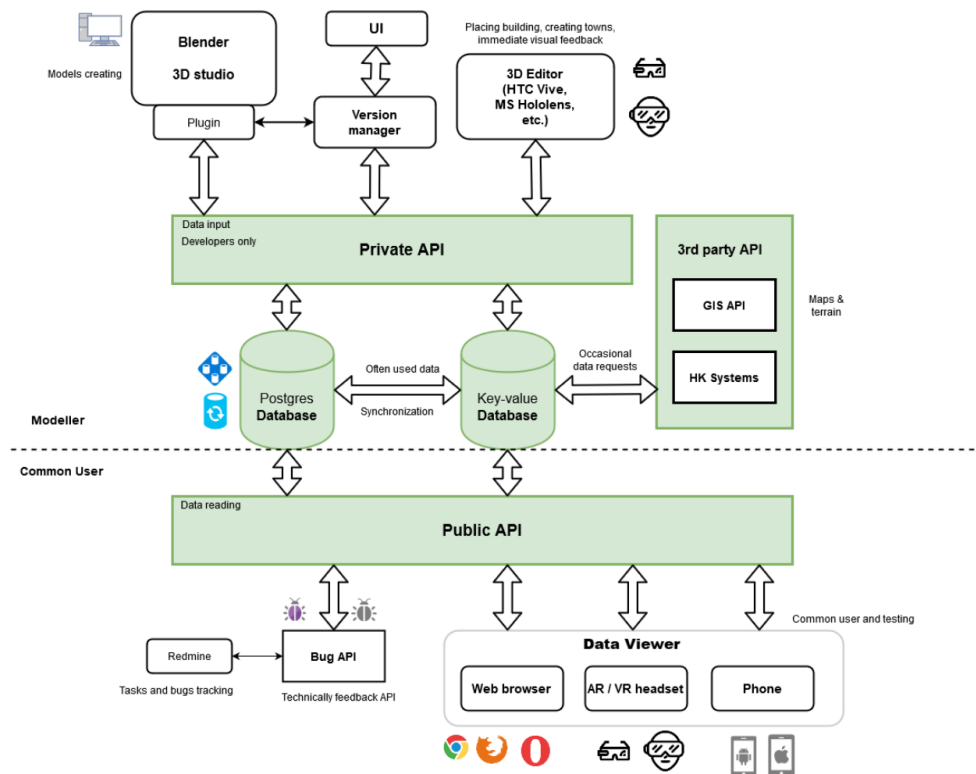


Figure 2.3: Scheme of the whole DTBQ project. Vančura’s thesis covered the green highlighted parts. The scheme is from Vančura’s thesis [23].

The approval process was created, because the 3D model creator and the historian are not expected to be one person. In this process, the 3D model creator creates a model, which is then approved by a historian and graphic designer who check the correctness of the model. This process is facilitated with version control for the models, so if the model is not approved, a new version can be created. The models have to be produced for many different conditions like snowy, sunny or night. It is not suitable to display buildings with snow on it in the summer.

There is also mentioned the concept of not only using a geolocation system but also computer vision for better accuracy. Vančura’s thesis specifically mentions the solution by using 360°photos. The state description of the project at the time before his thesis is elaborated here in detail. The parts on which he worked are clearly visible in the schema 2.3 as the highlighted parts. These parts are mainly databases and private API. The public API is then simplified and the Bug API is added to the public API. The Bug API serves the purpose of logging problems, errors and other data.

At the time Vančura’s work began, the technologies had already been determined as follows [23]:

**Docker** Tool that creates containers (environments) where the app can run independently of the outer platform,

**Node.js** JavaScript multi-platform environment,

**Express** Framework for Node.js that simplifies creating of the web application,

**Typescript** It is a technology introducing type variables into Javascript,

**TypeORM** Framework for object-relational mapping of object into database,

**JSON Web Token (JWT)** Security format standard,

**PostgreSQL** Open-source object-relational database system used in the project for storing metadata,

**MongoDB** Database for storing binary data in the key-value style. This technology was used for storing 3D models, textures, etc.,

**Geographical Information System (GIS)** This system includes maps and other data describing terrain like aerial photos or the plan of housing. This model is used to display the terrain.

### 2.3.2 Design and realisation

Within Vančura' design, the lacks of the previous solution were corrected, which were mentioned in the analysis in his thesis. The metadata database has been redesigned and the same has been done in the approval process. The version system for the models has been introduced, with the possibility to add tags on which base it would be easy to search through the objects. Last but not least, a vision of future development was outlined.

Regarding the API, the lack of the private API has been fixed, like no possibility of editing or deleting records in the database. After the correction, it is possible to do all the basic CRUD (create, read, update, delete) operations. The API allows to change supported formats, but now it supports the following formats:

**3D models:** blend, 3ds, max, obj, fbx, sfb, sfa;

**Images:** jpg, jpeg, png;

**Configuration files:** txt, config.

It is important to note that a simple manual for administrators and developers has been created within Vančura's realisation. This manual also contains an installation guide. The thesis with the manual can be considered a complete documentation of the project at that time.

---

## Analysis

In this chapter we analyse the methods from the Research chapter. We briefly test some of the methods to make a better image about them. Further, we analyse possible technologies for the development of localisation programs.

### 3.1 Previous theses

In this section, we specifically analyse previous works in the DTBQ project that have a direct influence on our work. Vančura's thesis was important for us as an introduction to the whole project and all its characteristics, but we do not analyse it. Instead of it, we analyse the works directly preceding our.

#### 3.1.1 Urban scene recognition and editing I

As mentioned above, the *Urban scene recognition and editing* thesis by Jan Šefčík [1] is the direct predecessor of our thesis which continues in Šefčík's footsteps. Within analysis, we take a look at what is worth keeping and what is good to further analyse, improve or do differently.

#### Analysis

The analysis describes which pre-processing methods are useful and which are not. We see this part as important, because the pre-processing is a part of the image processing and it should not be omitted, as Šefčík's thesis mentions. Šefčík also mentions that the only pre-processing method that makes a viable positive difference is the *Contrast Limited Adaptive Histogram Equalisation* (CLAHE) because it helps to compensate for the bad exposition or shadows. The feature detecting and describing methods have been well analysed too. Further, in his thesis only the SIFT algorithm, because it was evaluated as the most accurate one.

#### **Design and realisation**

In the analysis, we lack an explanation why FLANN was used for feature matching, so we will analyse if it is an ideal choice. Besides many other pieces of advice that we follow, there was the design and realisation of an image recognition prototype. The prototype was created in Python, which is not a typical programming language used to develop mobile applications. We believe that Python was chosen mainly for the purposes of the prototype and not for possible use in a mobile application. Based on that, we decided to further analyse what technologies and methods to use.

Šefčík created an image database as part of the prototype implementation. It contains buildings of different sizes or architectural styles. The database also contains additional information about the photos, such as the GPS coordinates of the place where the photo was taken. Šefčík mentions that GPS coordinates are key for speeding up the detection process. Based on the location, it is then possible to disqualify many buildings, so there will be significantly less buildings to search by using the computer vision.

#### **Testing**

In the end there was done testing on the database. The best results of Šefčík's work may be seen in the image 3.1. The results seem accurate in most cases. There are only a few images of the whole database so we consider that it is the showcase of the best results.

However we see testing as the main weakness of Šefčík's work. Every reference image was created from an image that is queried to be recognised only by blackening the parts of the image that are not the building itself. So for every queried image with a building, there exists a perfect matching reference image and thus no transformation invariance is tested except scale. This is in contrast with the use outside of the experimental environment.

#### **3.1.2 DTBQ - Image recognition module**

Šefčík's thesis is not the only important one on which ours continues. Another important one is the master thesis of Jaroslav Štěpán [24]. There are three goals that are important to us, namely creating the localisation module, the tracking module and the manual on how to make a module in the application. In the localisation module and tracking module goals is hidden another goal which is to create the Application Programming Interface (API) that would be used by the modules. The API would give the solutions addressing common problems of the modules.





Figure 3.1: Performance of Šefčík's prototype. Photo collection was made by Jan Šefčík [1].

## API

The API would provide support in the following areas: access to the camera of the phone, access to the phone sensors, connection to the network and access to the stored data. All the provided interfaces by the API are visible in the diagram in the figure 3.2. The diagram is a part of a design of the localisation module that was developed as a part of his thesis. At the top of the diagram are the interfaces that provide help to the localisation module, but also the interfaces that have to be implemented by the localisation module.

Specifically, the `INetworkProvider` interface provides the module the possibility to communicate with remote servers or to establish network connections. Another `ICameraProvider` interface provides the access to the camera or the properties of the camera in the form of camera intrinsic parameters. The other two supporting interfaces are `ISensorProvider`, which provides

easy access to the data from phone sensors, and `IStorageProvider`, which accesses the stored data to the module. All these interfaces have the same purpose and that is to make it easier to develop modules. This is done in this way, these interfaces are simple and easy to use. We see this as very useful, because we do not have to have a lot of knowledge about Android programming when developing our module. These interfaces take care of that.

#### Localisation module interface

But probably the most important interface is the `LocalizationModule`, which has to be implemented by the localisation module. We see the `getGlobalPosition` method as the main of the three methods there, because it is the method called by the application to get the location of the phone. The other methods are there to provide the application information about the module, which is also important, but is not the core of the module's purpose.

#### Realisation of a simple localisation module

In the diagram in the figure 3.2 also contains the design of the localisation module. The design in the part of Recognition is rather simple, but it sufficiently outlines the main properties of the main processing part like feature (called keypoint in the diagram) detection and feature description. There is also a method that is named `findObject` and its arguments are the features. We do not see this as completely clear, because features are not compared themselves, but their descriptors are what should be compared to find similarities in images.

There is a part of implementation that deals with pre-processing. All the pre-processing methods have to implement the interface. Based on Šefčík's thesis, we think that the filters could be scaling, CLAHE, and conversion of color images into grayscale ones. Štěpán states that the image is given by the mobile device in the YUV model and it is needed to convert it to RGB and then to shades of grey. He does not explain why the conversion to RGB is needed and we do not see the need to do so. The conversion to the shades of grey is necessary because all the considered feature detection and extraction algorithms are defined only for grayscale images. The separation of the Y channel (brightness) from the YUV color model should be a sufficient and simple way to create input grayscale images. However, it is important to note that it would be needed to store the descriptors of objects' pictures (of buildings or other objects to be found) in the database that would be computed also from the Y channel (Y from the YUV color model). This detail might not be necessary, but the disunity could lead to lower accuracy.

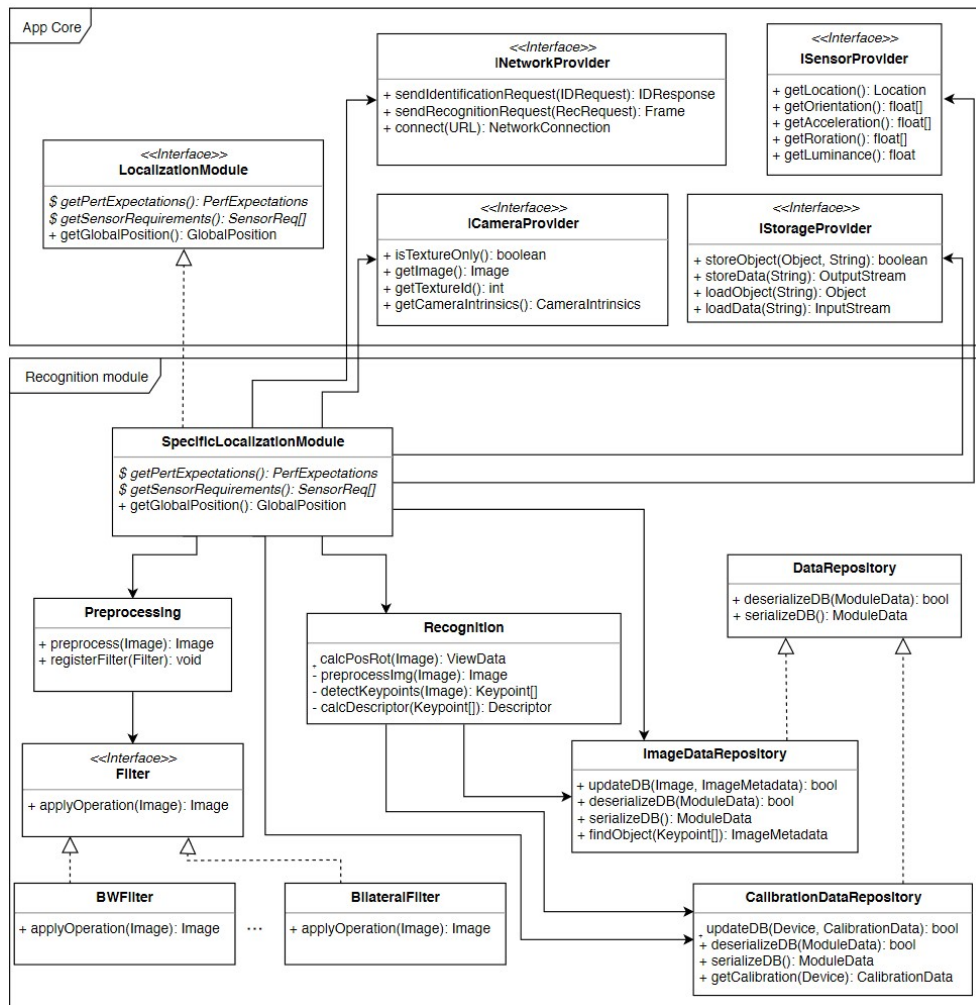


Figure 3.2: Localisation module class diagram of the thesis named textit-DTBQ - Image recognition module. Diagram is taken from the thesis created by Jaroslav Štěpán [24].

#### Localisation module developing

At the end of the realisation in the Štěpán's work, there are instructions on how to create a module for the DTBQ application. Thanks to these instructions the requirements of knowledge of Android development are lower. The manual expects the developer to use the C++ programming language for the main computations, so the manual also contains part on how to exchange data between code in Java programming language (Java is the main programming language of the application) and the C++ programmed part.

## 3.2 Computer vision libraries

In the preceding work by Jan Šefčík [1] was not done any analysis regarding the libraries, platforms or software development kits in the area of computer vision. OpenCV was introduced without any comment on exactly why to use this library, thus we analyse not only this library but also some others. Finally, we can make a reasoned decision on what to use in realising our project.

### 3.2.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source library in the field of computer vision and machine learning. OpenCV is designed to provide the infrastructure for computer vision applications. OpenCV is very suitable for use in our thesis, because it provides infrastructure in the same area as the application module that would be done in our work [25].

OpenCV versions up to version 4.4.0 are licensed under the 3-clause BSD license and versions since 4.5.0 are licensed under the Apache 2 license [26]. Both licenses allow both non-profit and commercial use including possibility for modifications [27], [28].

What we see as the benefits of OpenCV is a huge community of 47 thousand developers and more than 2500 implemented algorithms [25]. We are mainly interested in feature detecting, extracting, and matching algorithms. All of the algorithms that we are interested in are implemented in OpenCV. Functionality very similar to something we want to achieve is implemented in a tutorial in the official OpenCV documentation [29]. There is a small demo of how to find a known object in a picture. The result of this demo can be seen in this image 3.3. This is the major thing that we want to achieve in our work.

It is important for our project that OpenCV supports Android and besides that it also supports Windows, Linux and Mac OS. There is also a wide range of programming languages for which there exists an official interface. These languages are C++, Python, Java and MATLAB [25].

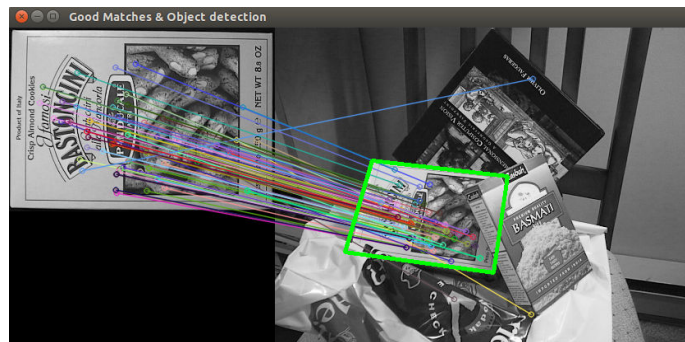



Figure 3.3: Result of a demo from OpenCV tutorial on how to find known object location in picture. Picture is from the OpenCV tutorial [29].



Labels	Text	Confidence
1	Stadium	0.9205354
2	Sports	0.7531109
3	Event	0.66905296
4	Leisure	0.59904146
5	Soccer	0.56384534
6	Net	0.54679185
7	Plant	0.524364

Figure 3.4: Example of the ML Kit *image labeling* of an image with soccer match. Example is taken from the website of the ML Kit.

### 3.2.2 ML Kit

Google offers multiple products to ease up mobile application development. One of these products is the ML Kit package [30]. ML in the name means machine learning. The ML Kit is a software development kit (SDK) designed to provide solutions for the common problems in computer vision. The package supports Android and MacOS platforms and it is offered for free under the terms of Google.

The ML Kit provides solutions in multiple areas, two of which are interesting to us. The first one is *image labeling*. The labels assigned to images are in the form of pairs of text and probability, as can be seen in figure 3.4. The text describes things that might be in the image or what happens in the image. The probability then represents the self-confidence of the algorithm with the label. This is an interesting functionality, but not very suitable for the localisation.

Another interesting area for us is the *detection and tracking*. In this functionality, the ML Kit provides more information about what is in the image than in the *image labeling*. The output of detecting and tracking can be seen in the figure 3.5. The ML Kit provides tracking and detection of multiple

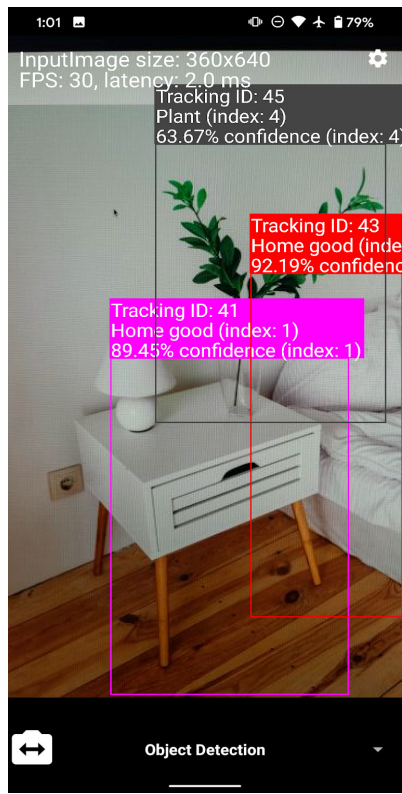


Figure 3.5: Example of ML Kit *detection and tracking*. Example is taken from the samples provided by Google on the ML Kit website.

objects in one photo. The output consists from the tracking ID of the object, boundaries of the area where the object is detected, and a label (similar to one in the *image labeling*). There is one main disadvantage, because the boundaries are only in the form of a bounding box. The bounding box provides too low accuracy for our intentions. There is no other positioning information than the scale and approximate location. Also, the use of the detecting and tracking is not in line with what we expect, because we want to detect objects from our own database and the ML Kit detects objects categories only from its internal database.

The ML Kit shows as capable but it provides less specialised solutions than we need. So we do not consider using ML Kit any further.

### 3.2.3 ARCore

ARCore is a platform created by Google for the implementation of augmented reality projects [31]. Augmented reality is also what the DTBQ mobile application is about. The first from the ARCore's three main elements of performance is the *motion tracking*. It covers the area of keeping the knowledge of



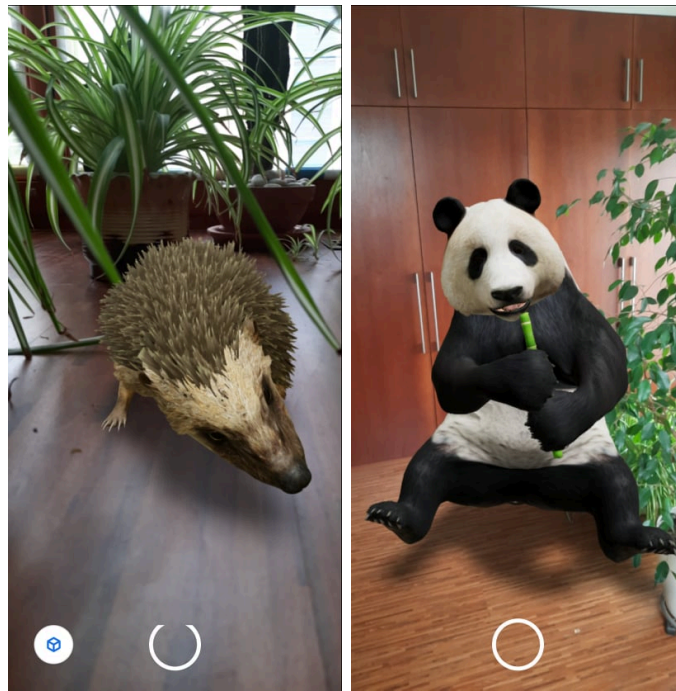


Figure 3.6: Example of an application based on ARCore. Images are screenshots from a phone application created by Google LLC.

the relative position of the camera in the surroundings. *Environmental understanding*, the second element of ArCore, is responsible for understanding the shapes of the surroundings. It mainly detects planes and the geometry of the world around. Performance of *light estimation*, the last element, is clearly visible in the image 3.6 where the animals create shadows on the floor in a realistic style. The images 3.6 are taken from a typical mobile application that uses ARCore. The application renders animals in the images of the real world in real-time. It also uses two other elements of the ARCore *motion tracking* (users can move phones around an animal) and *environmental understanding* (animal stands on flat ground).

ARCore is ideal for a tracking module of the DTBQ mobile application. That is probably why it was chosen for the tracking module in the work of Jaroslav Štěpán [24]. But we want to localise buildings in the picture and such functionality is not implemented in the ARCore, thus we no longer consider ARCore in our work.

### 3.2.4 Conclusion

We have analysed the OpenCV, ML Kit and ARCore, but the only acceptable option was the OpenCV, because the other two products have not met our requirements.

### 3.3 Programming language

Since we have decided to use OpenCV, we have to use some programming language that is supported by OpenCV. The supported languages are C++, Python, Java and MATLAB [25]. We eliminated MATLAB as not suitable and it is not mentioned here. Main reason is non-existent possibility of application development on the Android platform.

#### 3.3.1 C++

In our work we are implementing a module to the DTBQ application. Jaroslav Štěpán has already introduced instructions on how to create a module for this application using C++ [24]. Therefore, if we used C++, we would not need to research how to create the module ourselves completely but we could just follow Štěpán's instructions. Another advantage is that the DTBQ application is a real-time application and C++ is a viable choice for real-time applications. The programming language is generally known for its efficiency, so in case of programming our own processing methods the high computation speed is easily achievable.

#### 3.3.2 Java

The DTBQ mobile application for Android is developed using Java [24]. Java is also an official language for developing Android applications. This is a big advantage that speaks for the use of Java. But Java is an interpreted language and such languages generally run slower than compiled languages like C++. Therefore, if we want to code some algorithms not implemented in the OpenCV, it would lead to longer computation times. Our project creates a module in real-time application, in which speed is critical, and in the mentioned case of implementing our own algorithms could be the use of Java a big problem. But according to Štěpán's instructions at least some code would be needed in Java as an intermediary between the application and the C++ in case of developing application module [24].

#### 3.3.3 Python

In the case of Python, the advantage for us lies in the fact that Python was already used by Jan Šefčík [1] and his thesis is the direct predecessor of ours. Thanks to that, it would be possible to continue on the basis of Šefčík's code and develop it further. That is a major advantage. However, there is also one main downfall. In case of using Python, we would need to port that to Android ourselves, because the DTBQ application now supports only C++ or Java modules [24] and there are no instructions on how to create modules for the DTBQ application in Python.



## 3.4 Feature detecting, extracting and matching methods

### 3.4.1 Feature detecting and extracting

There are many methods for detecting and describing features. In the previous thesis by Šefčík [1] were analysed ORB, SIFT and SURF. His thesis concluded that the most accurate method of feature detection is SIFT. However, it is also stated that it is the slowest one among the three methods. The fastest one is ORB. The SURF method is commented as not suitable for the DTBQ project, because the method is patented and it does not bring any significant advantages over the other two methods that are free to use.

#### Methods not yet considered

What is not mentioned in Šefčík's work is the SIFT modification called Root-SIFT and feature description method called BEBLID.

RootSIFT is considered more accurate than SIFT with neglectable computation time increase. It is not included in OpenCV, but it is not complicated to implement it when the SIFT implementation is available, so we could implement it ourselves.

BEBLID was introduced recently, in the year 2020. At the time Šefčík did his thesis, this method was not included in the OpenCV like other methods. This method is implemented in OpenCV in modules with experimental and non-free algorithms, because it is considered an experimental method. But the method itself is free and it was published with the 3-clause BSD license [32]. To use this method, a compilation of the OpenCV with this module is needed. According to the documentation, this compilation with extra modules is also possible for Android. But anyway, the algorithm is published in the online Git repository in the form of a demo [32]. The demo was created using OpenCV and C++, so possible integration into our work should be without any problems in a case of using C++. The descriptors can be used to describe features detected by ORB or SIFT methods.

We do not consider to use any deep learning methods of feature detecting and extracting in our work. The application has to run in real-time on smartphones, where the resources are limited and the computational costs of running deep learning methods are increased significantly in comparison to *handmade methods*[14].

### 3.4.2 Feature detecting and extracting in OpenCV

All the feature detecting and describing methods in OpenCV can be easily exchanged, because they implement the `Feature2D` interface. So the choice of the method can be done later or the choice can be configured at runtime.



Figure 3.7: Early evaluation of our prototype. Correctly displayed object (building) in the scene on the left and badly displayed on the right. Building photos are from Jan Šefčík’s image database [1].

But since we are interested in a real-time application, the speed of the feature detection and description is important.

### Testing

An early prototype was implemented using OpenCV to test the speed of the methods. For testing we used a subset of images from the image database of the Šefčík’s project [1]. We did 5 tests and in each one there was one scene image (urban scenery) and a database of reference building images, which were searched in the scene.

The number of detected features was limited to 1000. The number was slightly lower for several images, because the methods were not always able to find 1000 features (we observed that ORB is more reliable in detecting at least 1000 features than SIFT). After detecting the features, the features were described with a predetermined algorithm and the time that the process took was carefully measured.

Each of the object descriptor vectors was then compared with the calculated one from the scene image. The object that best suited (according to simple metric) to the scene was chosen and displayed in the scene. After that we could visually evaluate if the display of the object in the scene is correct or not the same, as can be seen in images 3.7. This test was primarily prepared in mind to measure the time. The accuracy measurement was simplified to a binary evaluation: fail or success.

### 3.4. Feature detecting, extracting and matching methods

Detection:	SIFT	SIFT	SIFT	ORB	ORB
Description:	SIFT	RootSIFT	BEBLID	ORB	BEBLID
1 success	yes	yes	yes	yes	yes
1 time (ms)	7758 (13775)	8064 (13543)	7729	1361	897
2 success	yes	yes	yes	yes	yes
2 time (ms)	10025 (17958)	10220 (17824)	9931	1626	1022
3 success	no	yes	yes	no	no
3 time (ms)	9843 (17706)	10556 (17940)	10120	1700	1086
4 success	yes	yes	yes	yes	yes
4 time (ms)	7608 (13687)	8274 (13639)	7794	1395	916
5 success	yes	yes	yes	yes	yes
5 time (ms)	7739 (13500)	8203 (13903)	7636	1349	882
Total success	4/5	5/5	5/5	4/5	4/5
Average time	8595 (15325)	9063 (15370)	8642	1486	961

Table 3.1: Comparison of OpenCV feature detection and description methods. All tests included processing of 48 images with the limit of detected features set to 1000. There are two values for SIFT-SIFT and SIFT-RootSIFT columns, because time depends on the usage of the OpenCV methods.

#### Test results

The test results can be seen in the table 3.1. In case of the SIFT-SIFT and SIFT-RootSIFT columns, two different values can be explained in the following way. The feature interface offers to detect and describe features in one step (values outside brackets), or to split the calculation to two steps (values in brackets). In the case of SIFT, the splitting into two calculations leads to a significant speed decrease. We think this behavior is specific to the OpenCV SIFT implementation.

The conclusion from this testing is that SIFT is by a magnitude order slower than ORB. The RootSIFT modification is only 5% slower than SIFT and is more successful. Also the combination of SIFT feature detection and BEBLID description has 100% success rate (in our test), the same as RootSIFT and it is only slightly slower than SIFT-SIFT. Method combinations with ORB ended up as less successful in terms of correct results than combinations with SIFT. But on the other side, they are multiple times faster. The combination of ORB and BEBLID achieves even a 35% increase in speed in comparison to pure ORB and still maintains the same success rate.

#### 3.4.3 Feature matching

In Šefčík’s work [1], it is stated that another option is to use matching methods based on FLANN. The advantage of these methods is that it creates data structures which speeds up the searching of a match for the feature. But these

### 3. ANALYSIS

---

Distance: Matching method:	Hamming Brute force	Hamming FLANN	L2 norm Brute Force	L2 norm FLANN
1 success	yes	yes	yes	yes
1 time (ms)	86	531	119	898
2 success	yes	yes	yes	yes
2 time (ms)	91	539	121	922
3 success	no	no	no	no
3 time (ms)	87	584	119	915
4 success	yes	yes	yes	yes
4 time (ms)	88	541	120	921
5 success	yes	yes	yes	yes
5 time (ms)	90	526	123	902
Total success	4/5	4/5	4/5	4/5
Average time (ms)	88	544	120	912

Table 3.2: Comparison of two matching methods. All tests included processing of 48 images with the limit of detected features set to 1000.

structures have some additional costs, such as the cost of the structure built at the beginning. The number of features has to be therefore large enough, so the FLANN based matching is faster than the brute-force matching. The question is which method is faster in our case.

#### 3.4.4 Feature matching in OpenCV

Both of the above mentioned matching methods are implemented in OpenCV. We therefore prepared a test, similar to that one mentioned above, to find out which matching method is the fastest in our case. We were interested in how the matching methods performed, according to the used distance to compare the descriptors.

The test result can be seen in the table 3.2. We can see that the brute-force matching is clearly faster than the FLANN-based matching. The number of features is the key in which method is faster. So we also performed a test without maximum detected feature limit (it led to about 5000 features per image) and this test also favoured brute-force matching. The type of distance used influences the matching time as well. Hamming distance made both matching methods to run faster than L2 norm (Euclidean) distance. Based on this test we could consider to use only brute-force matching. In case of change for another matching it would not be a problem because all OpenCV feature matching implement the same interface.

---

# Prototype development

As a main part of our work is the creation of a prototype, which is intended to be a development, debugging, and testing platform for a localisation solution. Such a prototype would be similar to that one that was developed in Jan Šefčík's work [1]. The goal of doing this is to push what Šefčík has done further and to extend the process by calculating not only the transformation matrix between the reference image space and the image space of the image queried to be recognised.

## 4.1 Design

We will follow the design of Jan Šefčík[1]. So in this section, we will describe the changes from the Šefčík's design that we will make.

### 4.1.1 Programming language

The main change is that we will choose C++ programming language instead of Python. To be able to work with a file system and parse JSON files we decided to use Boost (set of C++ libraries).

C++ will be used, because of its efficiency and the code itself will be more similar to that one in the application localisation module. The C++ is planned as a programming language for the computation parts in the localisation modules. We expect that in that way it will be possible to integrate big code parts from our prototype.

### 4.1.2 Configurability

We will continue using JSON files for storing additional image data. However we plan to create a more sophisticated configuration system than Šefčík's, where the initial configuration Python source files were used. That is not

possible in C++ without compiling. We want to approach it in a more user-friendly way by moving the configuration into JSON files. This configuration will be divided into more JSON files, so they can be easily exchanged. They will be all linked in main JSON configuration file that will have following form:

```
{
  "reference_images" : "filepath/references.json",
  "scene_images" : "filepath/scenes.json",
  "parameters" : "filepath/parameters.json",
  "output_root" : "Where/to/save/output/",
  "run_name" : "some name"
}
```

The `reference_images` field will point to JSON with file paths to the reference images of buildings and in a similar way for other fields. The JSON file with parameters will be a file with parameter fields similar to that mentioned above. Based on our testing of feature detecting and extracting methods we observed that the configuration data changing the most across tests is the input scene image. The JSON file with scene images file paths will have following structure allowing smooth workflow:

```
{
  "index" : 0,
  "scene_images" : [
    "filepath/scene1.jpg",
    "filepath/scene2.jpg"
  ]
}
```

where the `index` determines the used scene from the `scene_images` array. The most frequent expected use case is including going through the scene images and in this case it is needed to increment the index only.

### 4.1.3 Database

Same as in Šefčík's work we will use a file-system database to store images. All the needed image data other than the image itself will be stored in JSON files within the same directory as the images and the same name but the suffix. Information about reference images of buildings will contain localisation data and information about the scene images will be the camera intrinsic parameters and eventually the global position, where the image was taken.

The file paths of the images will be stored in a dedicated JSON file. To fill our database with images we will use image sets created by us and the images from the work of Tatiana Popova [33].

#### 4.1.4 Output

The current mobile application DTBQ interface for localisation modules requires global coordinates and rotation to the object in the form of a quaternion, so that will be realised as well. Beside this, the prototype will calculate a 3D transformation from the world space to the local space of the camera.

The result will be provided in the form of image visualisation and text information. Both will be provided in the real-time while running the prototype or by saving the output into files. This will create the options to debug the application and also create and store test results. The text information will include the information about the processing and the result.

#### 4.1.5 Elimination based on location

An elimination of images based solely on global coordinates will not be a part of our prototype. The intention of the prototype is to test the algorithms and methods and create the solution for given problems. The location elimination would significantly decrease the input to these methods.

## 4.2 Implementation

In this section we write about the realisation of the localisation prototype. In this section we first describe what inputs have been defined and which outputs have to be expected. We then explain what processing is done to calculate the output from the input.

### 4.2.1 Input and output formats

In this part we describe the necessary inputs and expected outputs of our localisation prototype. We expect that our localisation prototype will be similar to other possible localisation solutions in the DTBG project. So this information could be used in further development in the DTBQ project.

#### Reference building images

The input images on the side of buildings with known locations to be recognized in the camera scene are called reference images. Such an image of a building has to be created in a way that it looks like a parallel projection from the front of a building. The image has to be without the perspective deformation as seen in the image 4.1. In this illustration is visible how the reference image was created. The perspective is removed so the real world horizontals are horizontal also in the image and the same for the vertical direction. In this way, the deformation is only partially removed, because the original image from which the image in the database was created is influenced by the perspective, that is well visible in the middle picture in the figure 4.1.

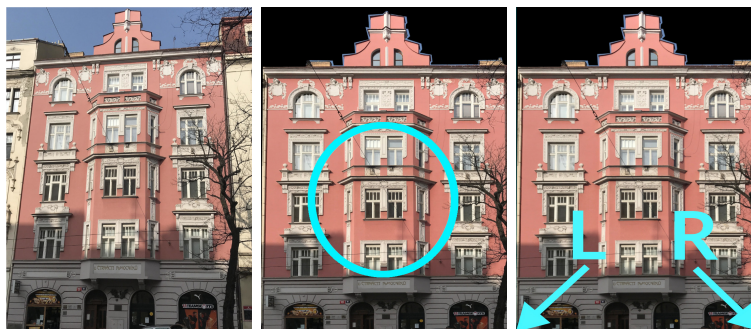


Figure 4.1: Illustration of creating a reference building image. On the left is the original picture, in the middle is the created database reference image. The cyan ellipse shows the remaining perspective deformation (the right side of the building extension is wider in the image than the left side). On the right is an illustration of the global location requirements. Original photo: Tatiana Popova

A photo that was taken straight in front of the object/building from a large distance is ideal for creating the image for the database.

With the reference building images, the *JSON* files have to be created, with the same name and information about global coordinates (often just called GPS) of the left and right corner as in the right image in the figure 4.1. The position data has to be given exactly for the places in the left and right corner of the image, the lowest to the ground as possible. The *JSON* data structure looks like as follows:

```
{
  "right_corner": {
    "longitude": 14.419302807,
    "latitude": 50.086761105
  },
  "left_corner": {
    "longitude": 14.41927048,
    "latitude": 50.086600324
  }
}
```

### Camera scene images

Camera scene images are images taken by phone that are to be recognised. There are two following necessary requirements that has to be met to obtain valid results:

- The scene image must not be cropped (it can be uniformly scaled).



- The following information about the camera from which the images have been taken:
  - Focal length,
  - The size of the camera sensor in both directions.

The following JSON content is example of a file attached to every scene image:

```
{  
  "camera_name" : "Xiaomi Redmi 5 Plus",  
  "focal_length" : 4,  
  "sensor_size_x" : 4.96,  
  "sensor_size_y" : 3.72  
}
```

The following sentences are guidelines for best results. The localisation has better results, if the scene image has such a resolution that the resolution of the building in the scene image matches the resolution of the reference building. This is usually slightly higher than a reference image resolution. A scene image is better recognised if taken parallel to the ground (rotated images might not work as well).

### **Resolution of the images**

The size of the image affects the computation time and size of allocated memory for the feature detection and extraction. When we used high resolution ( $\geq 12$  MP) the size of used memory was about 3 GB at the peak. The resolution about 750x1000 (0.75 MP) works for us as a good compromise, because about 240 MB is allocated at the peak and the computation time is shorter.

### **Global location**

The global location in the global coordination system and rotation towards the object/building are the first possible result of the localisation. They are in the following format:

- Global location of the camera defined as follows:
  - Latitude in degrees,
  - Longitude in degrees.
- Quaternion representing rotation of the direction vector pointing towards the detected object from the east.

### Rotation-translation matrix

The second type of output is a rotation-translation matrix, which transforms points from world space to the local camera space, the same matrix that was described in section 1.6. The biggest advantage of this output is that it does not depend on the global location. The output structure is following in this case:

- Which object is in the scene image,
- Rotation-translation matrix.

The global location of the left  $L$  and right  $R$  corner are exchanged in runtime (not in files) for the world space coordinates (virtual world space) which are defined as follows:

$$L = (0, h, 1, 1)^T, \quad R = (w, h, 1, 1)^T \quad (4.1)$$

where  $w$  is the width and  $h$  is the height of the reference image.

### 4.2.2 Processing pipeline

In this section, we closer specify the processing pipeline. The whole processing pipeline is documented in the activity diagram in figure 4.2. Some parts are very similar to the activity diagram by Šefčík [1] in figure 2.2. We redone all these parts, but we followed similar principles, so we do not explain these parts, but we explain what we have added. Above all, we have added the possibility of pipeline configuration, calculation of 3D transformation, global localisation, and interpreting of the matching data.

### Choosing the best match

During the development of the prototype, we searched for the best option for how to choose the correct match between multiple matches. We found three viable options, namely the average of values from the Lowe's ratio test, the average distance of feature matches and the ratio between matches and filtered feature matches. We discovered strong positive correlation between these three values. We chose the way of searching for the highest ratio between matches and filtered matches. It has an asymptotic complexity of  $O(1)$  in comparison to the linear asymptotic complexity of the averages depending on the number of matches.

### Calculation of 3D transformation

In this calculation, the PnP problem has to be solved and the rotation-translation matrix has to be found. We used the OpenCV iterative implementation of that problem in the OpenCV module *calib3d*. As an input for

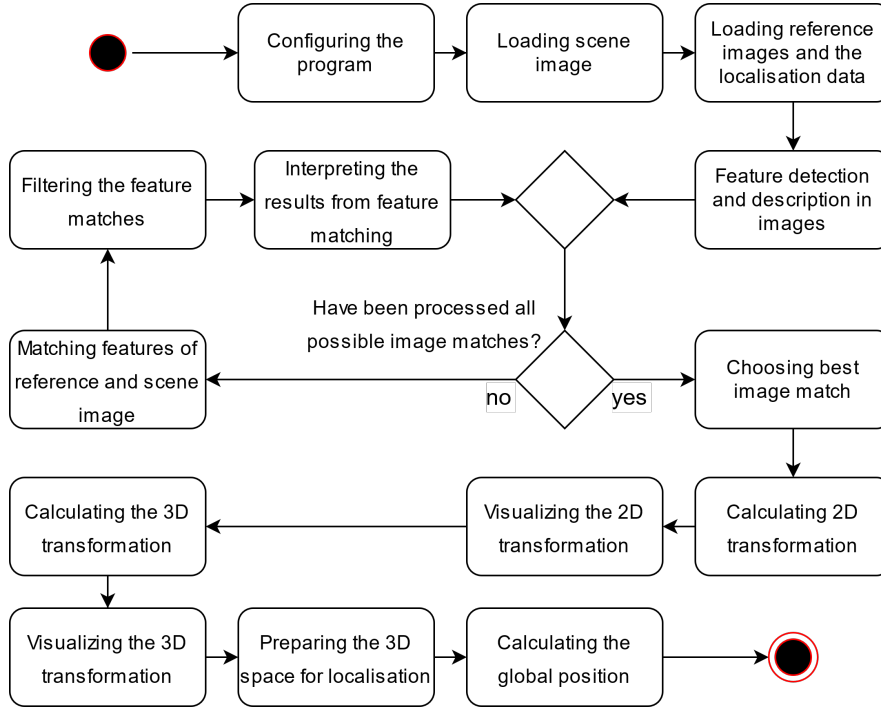


Figure 4.2: Activities diagram of our prototype for default configuration

this PnP problem solution we provide a 3D plane geometry (building facade) of the image and 2D projection of the reference image corners in the space of the scene image. These corners can be obtained by transforming the original reference corners with the transformation matrix from the 2D solution.

The 3D plane geometry  $G$  for image of size  $h \times w$  is defined as follows:

$$G = \{(0, 0, 1, 1)^T, (w, 0, 1, 1)^T, (w, h, 1, 1)^T, (0, h, 1, 1)^T\}. \quad (4.2)$$

### Localisation

When the corners location of the building plane (facade) are known in the camera space and the global coordinates are known for the two bottom corners of the building, we can calculate the global coordinates of the camera. For our prototype, we only considered a flat ground and the computation will be less accurate in the case of a slope along the building facade and inaccurate in case of a slope in different directions. But because the dowry towns are mostly on flat ground, it is not a significant problem.

There is one more problem and it is the height at which the camera of the phone is above the ground. It is similar to the problem with the hilly terrain, but it has an easier solution because we can expect some average phone holding height. For this problem, we created an illustration in the figure 4.3. The problem is that the real location of the camera global coordinates are not

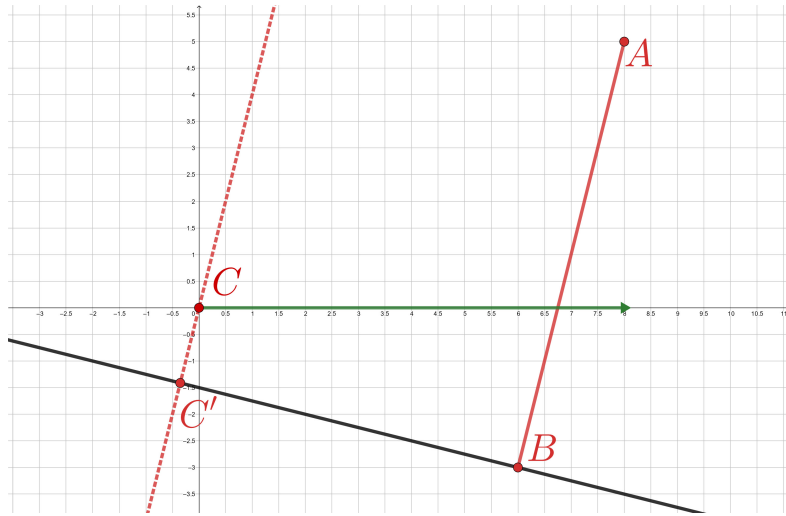


Figure 4.3: Illustration from side of a camera coordinates correction. The green line stands for vector in the direction of the camera view, the black line stands for the ground, the red line then for building facade from the side, the red striped line is then real world up vector in place of camera  $C$  and  $C'$  is a intersection of the up vector and the ground.

at the origin (where is the camera located) but it lies on the intersection of ground and the real world up vector passing through the origin. As seen in the image, there is a small distance between the intersection and y-axis which makes the difference.

The up vector is computed from the building facade. Therefore, if it is not guaranteed that the facade of the building will create a real world up vector (as in the case of objects other than buildings), we recommend turning off this feature in the configuration of our prototype.

### 4.2.3 Configurability

Our prototype was realised with the flexibility in mind. It is possible to configure which methods are used for feature detection, feature extraction, feature matching and possibility to configure these methods by adjusting the OpenCV parameters. List of these methods consists of SIFT, RootSIFT, BEBLID, ORB, brute-force matching and FLANN based matching. The Lowe's ratio is also possible to easily set in configuration. Other possible settings to disable or enable are global coordinates calculations, optimisation for bigger accuracy when measuring time and optimisation for calculation of the global location.

The last configurable thing is if the log of the whole process with images is displayed in real-time or saved to files. The prototype creates a log with information about what it processed, how and with what results and values.

All the parameters of the feature detection and extraction methods (except detected feature limit) are configurable only in the header file before compilation. They are in the header file, because they are determined to be changed only by a knowledgeable person. However the prototype is built so they can be changed without any problems.

## 4.3 Experimental evaluation

We have done testing in which we focused mainly on the comparison with the phone location sensor (GPS sensor). We did this because the use of the prototype (a future phone module) has to provide a more accurate location to that one that is provided by the phone sensors.

### 4.3.1 Methodology

We prepared a database of reference building (or other objects) images and then created a set of 25 tests. In each test, we queried images of the scenes. Some of the images were taken from the image database created by Tatiana Popova for her bachelor thesis [33]. We created the rest of the images to cover our needs for testing which could not be covered with images from Popova's database. We made the tests with the intention to create similar conditions to these ones that would be present when using the mobile application.

#### Reference building images

There were prepared all the objects that covered a wide variety of objects, but most of the objects are buildings, because that is mainly what the prototype was optimised for. We chose buildings with different architectural styles. With our tests we covered the following areas:

- Buildings with one front facade,
- Corner buildings,
- Towers,
- Free-standing columns.

#### Scene photos

The image with the scene is such an image containing the building/object that we put into the database. The style of every scene image is such that it could be taken by any tourist/anyone. This means that the image was taken by phone camera from such places where people usually move like pavements. In more detail, the tests were created in a way that they cover all the following areas:

- Ideal camera shots from front of the building,
- Camera Shots from angles up to 50°,
- Different weather and lighting conditions,
- More reference objects in the scene.

### Used algorithms and methods

We decided to use SIFT without modification, because it was tested a lot during its existence and considered to be reliable and as a *golden standard*. We needed reliability, because we also needed to test other parts and thus the whole prototype. But for any future testing, we encourage the testing of a combination of SIFT and BEBLID or RootSIFT modification of SIFT. The parameters of the SIFT algorithms were set to default values except for a number of features that were set to 1000. For feature matching, we used the brute force matching and Lowe's ratio test and RANSAC for filtering.

### Measuring results

We saw three ways on how to measure the results. The first is to compute the global location and project it on the map with the global location given by the phone sensors. With this, we were able to decide which one better suits the photo scene and as a score we measured the difference between a more accurate and a less accurate solution. So the result will be in form of a number measuring increase of accuracy in comparison to phone sensors (negative number in case of accuracy decrease in comparison to phone sensors).

The second way is to binary decide if the building was correctly recognised. That means if it was recognised which object/building is in the photo scene.

The last way is to evaluate the understood relative location to the object/building in the form of a projection matrix. With such a matrix, we could render into the scene image a wire frame of a prism. This prism represents the understanding of the spatial characteristics of the object/building by our prototype.

#### 4.3.2 Results

We created a representation of the results in the form of a histogram 4.4, where the results of the first measuring method can be seen. The results are in the form of gained accuracy in meters in comparison to the phone positioning sensors. We assign -56.58 (the additive inverse of the maximum positive difference) as a value to the failed cases (when our prototype failed to identify the building in a scene image) and we obtain an average accuracy increase of 0.98 meters. This result is a small gain in accuracy from the results of the mobile device sensor. If we omit all the failing cases, the average increase

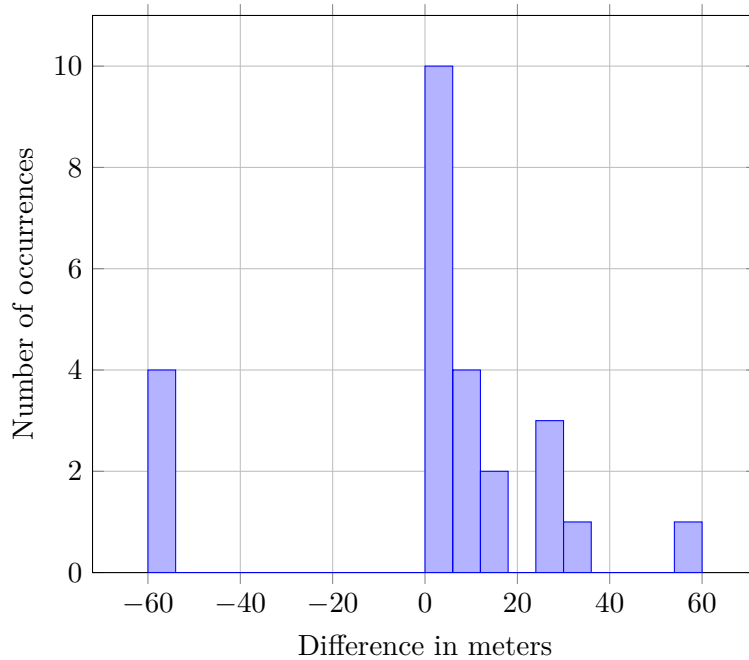


Figure 4.4: Testing results of the prototype in the form of a histogram. The values in the histogram are the differences between the phone sensor’s and our prototype’s location output in meters. If the result was more accurate for our prototype, then the numbers are representing the accuracy increase compared to data provided by phone sensors. Special value is -56.58, this has been assigned to 4 cases where our prototype completely failed. Zero was assigned to 5 undecidable cases.

in accuracy is 11.94 meters. If we further omit the undecidable cases (cases when we could not tell if the phone was more accurate or our prototype), we obtain an average accuracy increase of 15.68 meters (higher value is better). Thus, for that measurement, we can conclude that there are cases where our computer vision localisation solution fails completely, but if it does not fail, it provides superior accuracy in comparison to the phone sensors. It is important to note that no front facing test (ideal conditions) has failed.

The second measurement results are 21 (84%) successes and 4 (16%) failures.

The last measurement method has similar results as the previous methods, because when the object was recognised (success), then at least a somewhat accurate understanding of the spatial object/building characteristics was created. Usually such a result is the same or better than the image 4.6b. The images in the figure 4.5 represent a perfect match and a fail. In the image with the fail was the volume of the column missed, but on the other side, in the successful image, the object is perfectly matched with the prism.

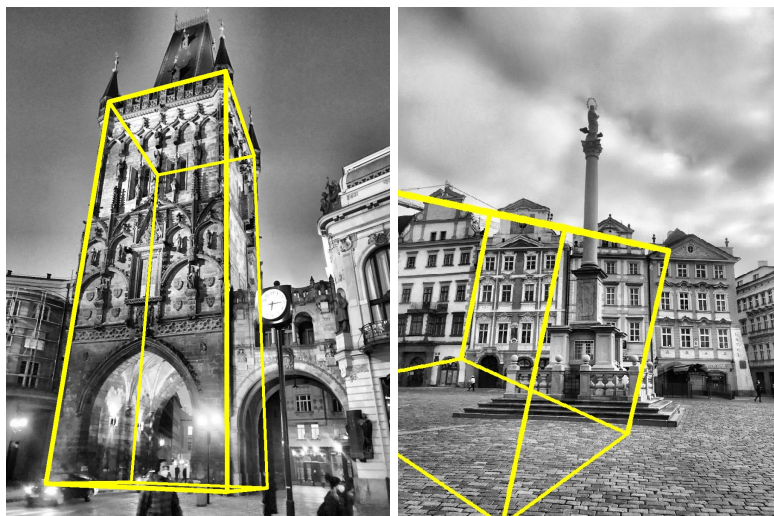


Figure 4.5: Results of understanding of the spatial object/building characteristics. On the left is the best result, where the prism perfectly matches the tower. On the right is then a totally spatially misunderstood object (fail). Author of original photos: Tatiana Popova

### Notes to testing

During the testing, we improved the reference pictures database by adding a night version of objects or improving the images. In all tested cases, the daily reference pictures led to failure when testing with the night scene image. But after adding the night reference image, the object was detected and in one case ideally matched as seen in perfect match in figure 4.5. Improvement of the reference images is well seen in the set of pictures 4.6, where it is well visible that the correct size (height to width ratio) of the reference image is very important to get right for the accurate matching.

We tested a column at the town square where a building is visible in the background and we performed two tests with the same image. The first test was before and the second after adding the improved reference image. In the second test, the building in the background was detected instead of the column in the foreground. It is important to say that the first test failed. This case can be seen in the figure 4.7. Because of detecting the building in the background, the location was detected more accurately than from the phone sensors.





(a) Original reference



(b) Result



(c) Enhanced reference



(d) Enhanced result

Figure 4.6: Example of building's reference image size importance. It is visible that the accuracy in the top result image is worse and it is visible that the reference picture is deformed (it is too wide). Photo: author



Figure 4.7: Detection of a building in the background instead of the foreground object, after improving reference image of the detected building in the background. The test on the left image failed. Original photos: Tatiana Popova

## 4.4 Installation

In this section, we provide a manual on how to install our project in a way that would be able to compile and run the prototype. We describe how to satisfy all the dependencies of the OpenCV and Boost C++ libraries. We have developed the prototype in the Visual Studio 2019 IDE on operating system Windows and the manual is written in a way to provide installation guide for this configuration.

### 4.4.1 Installing OpenCV

To run our prototype, it is needed to install OpenCV. There are two options for installing OpenCV. The first is to download the already built libraries and the second option is to compile the OpenCV codes. We write about two ways of installing, because there are some OpenCV modules that are not included in the official prebuilt binaries. Such a module is *xfeatures2d* (extra/experimental 2D features algorithms), which is important for our project, because it contains the BEBLID algorithm.

In both cases, the version 4.5.1 and higher has to be installed, because the BEBLID algorithm was introduced in this version. Also, the algorithm SIFT was already moved from the *xfeatures2d* module to the regular *features2d* module (2D features algorithms) in this OpenCV version.

### Installing prebuilt version

Installing the prebuilt version is more straightforward than compiling the library and installing it afterwards. The following manual is for installing OpenCV 4.5.2 for developing projects in Visual Studio 2019 on the operating system Windows. The steps of the installation are as follows:

1. Download the sources of the version 4.5.2 from the OpenCV releases page [34],
2. Unpack the sources into desired location `<INSTALL_ROOT>`,
3. Run the *Edit the system environment variables* tool and click the *Environment Variables...* button,
4. When installing on the same account, create and edit the following variables in the table *User Variables* otherwise in *System Variables*,
5. Create there a new `OPENCV452_ROOT` variable with value `<INSTALL_ROOT>\`,
6. Add `%OPENCV452_ROOT%\vc16\bin` record for the `PATH` variable.

Follow next steps for installing OpenCV into Visual Studio project (the naming for `OPENCV452_ROOT` may vary in different projects, so it is further called only `OPENCV_ROOT`):

1. Create an empty C++ project in Visual Studio or download our project,
2. Open project properties,
3. Set the configuration (release/debug) that would be edited,
4. In the *C/C++* section, open the *General* subsection and add the `$(OPENCV_ROOT)include` record for *Additional Include Directories* field,
5. If there are already some records in the fields, separate the records with semicolon,
6. In the *Linker* section open the *General* subsection and add the `$(OPENCV_ROOT)x64\vc16\lib\` record in the *Additional Library Directories* field,
7. In the *Linker* section, open the *Input* subsection and add the `opencv_world452d.lib` record in the *Additional Dependencies* field for debug configuration or `opencv_world452.lib` for release configuration (configurations can be set at the top of the properties window),
8. It might be needed to restart the Visual Studio.

Further information may be found in the installation tutorial in the official OpenCV documentation [35].

### Compiling the library

For installing the library, it is best to follow the tutorial in the official OpenCV documentation [35]. To compile OpenCV with the *xfeatures2d* module, the CMAKE parameter `OPENCV_EXTRA_MODULES_PATH` has to be set to `opencv_contrib/modules` where `opencv_contrib` is a directory containing sources downloaded from the *Repository for OpenCV's extra modules*. For our project it is also necessary to set the variable `BUILD_opencv_world` to true.

#### 4.4.2 Installing Boost C++ Libraries

The second dependency of our project is the Boost C++ Libraries. We use the Boost version 1.76.0 in our project. To install Boost on Windows, the steps are following [36]:

1. Download and unpack the Boost ZIP file,
2. Go to the `tools\build\` directory,
3. Run `bootstrap.bat`,
4. Then run `b2 install --prefix=PREFIX` where the `PREFIX` is the directory to which is the Boost installed,
5. In a similar way as in the OpenCV installation manual, add a record `PREFIX\bin` to the `PATH` system variable.

In our project, we have followed the installation manual from the official Boost documentation [36]. More information can be found there. So to run our project, following steps has to be taken during installing Boost to Visual Studio project:

1. Create a new system variable `BOOST176_ROOT` with value `<PREFIX>\`
2. Open the project properties,
3. Set the configuration (release/debug) that would be edited,
4. In the *C/C++* section, open the *General* subsection and add the `$(BOOST176_ROOT)` record for the *Additional Include Directories* field,
5. In the *C/C++* section, open the *Precompiled Headers* subsection, set *Precompiled Header* field to `Not Using Precompiled Headers`,
6. In the *Linker* section, open the *General* subsection and add the `$(BOOST176_ROOT)stage\lib` record in the *Additional Library Directories* field,

7. If there are already some records in the fields, separate the records with a semicolon,
8. It might be needed to restart the Visual Studio.

### 4.4.3 Running and compiling our prototype

To run and compile our prototype, it is necessary to install the OpenCV and Boost C++ libraries. The project was developed in Visual Studio 2019, so the easiest way to start is to also use the same IDE, because the manuals are created in that way.

There are some further notes to the installation. When OpenCV without *xfeatures2d* was installed, then the `COMPILE_EXPERIMENTAL_MODULES_ENABLED` macro in the *experimentalModules.h* header file has to be disabled. When compiling OpenCV manually, the `opencv_world` has to be created, otherwise the additional library names in the project properties have to be specified differently. The system throws an error with the name of the missing library when running without them, so the names of the libraries can be found out by trying.

When setting up the project and adding the additional library information in the project properties, the values may already be there if the project was downloaded by default.



---

## Future outlook

Despite the fact that it was not the task of our thesis, we considered creating a localisation module for the mobile application of the DTBQ project. We did not implement our own complete implementation, but we did a survey on how to create such a module and how to integrate it into already created code. We write about this in this chapter together with proposals for improving.

### 5.1 Contextualization

First, we want to put into the context of DTBQ what we have already created. We have tested and developed our prototype on images taken mainly in Prague, which is not a dowry town of bohemian queens, but we want to show that it can be easily used in the dowry towns as well. We also want to show how augmented reality could look based on our localisation.

To do this, we have created a visualisation in figure 5.1. In the image is a rendered model of the Kropáčka tower, which stood in the past in the dowry town Hradec Králové [37]. The render was stylised to match the night lighting conditions. But otherwise it was rendered based on the unchanged information given by our localisation prototype. The rendered tower does not cover the whole tower that was originally in the image (Prašná brána), because the sizes of both towers are different. But thanks to this, it is possible to see that the perspective of the rendered tower corresponds to that one of the original tower.

### 5.2 Design of the localisation module

We already specified and analysed the API and realisation of the localisation module by Jaroslav Štěpán in the section 3.1.2. We follow his design of the localisation module in most of the places. The part mediating the communication between the computing part and the application will be implemented



Figure 5.1: Visualisation of the localisation with a 3D model of the Kropáčka tower. Environment photo was taken from the database of Tatiana Popova [33] and the 3D model was created as part of study [37].

in Java. The localisation module interface `ILocalizationModule` has to be implemented to create a localisation module in the application. The communication with the native code in C++ will be done via the Java Native Interface (JNI). The interface will have following methods:

- `createProxyObject()`,
- `destroyProxyObject()`,
- `updateDatabase()`,
- `getGlobalPosition()`.

That list closely corresponds to the methods used in the JNI used in the local localisation prototype by Jaroslav Štěpán [24]. The communication between the application, the Java part of the localisation module and the native code in C++ would look like in the sequence diagram 5.2. The key point of the



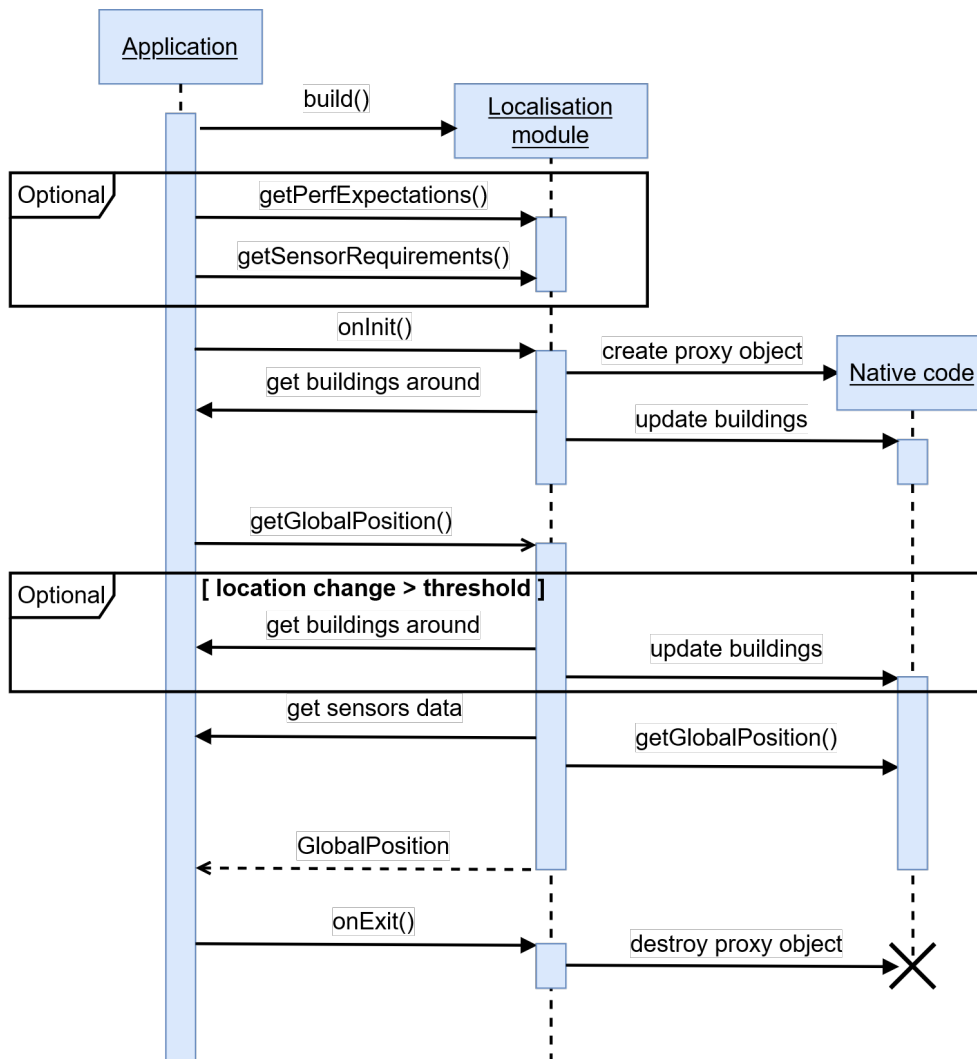


Figure 5.2: Sequence diagram of the communication between the application, localisation module and native code

communication between the Java part of the module and the native code is to use a proxy object. The proxy object will be created on the heap memory and accessed using the address in the memory of the object whenever needed until it is released. The following list is an explanation of some called functions in the sequence diagram 5.2:

**getPerfExpectations()** Method implemented by the localisation module that returns information about what hardware usage the module would have.

**getSensorRequirements()** Method returning what sensors the module needs

to operate with.

**getGlobalPosition()** Method that is called by the application to get the location.

### 5.2.1 Integration of the already finished code

A big part of the already created code could be moved into the module. The main part that would have to be redone is the input loading and processing configuration. The prototype is not planned to store the images for a longer time for multiple processing. Therefore, it would be appropriate to create a run-time database of reference images inside the native code. The database would be updated whenever the device location would be changed more than by pre-defined the threshold. In the run-time database would be prepared descriptors of reference buildings images for next calculation of the global location, so that the response would be as fast as possible. The rest of the code would be also optimised to get even better performance, mainly because some parts of the prototype are redundant for the module inside the mobile application. But the main processing part of the prototype could be just modified in a way that it would be possible to run it directly after calling the `getGlobalPosition` method of the proxy object.

## 5.3 Future improvements proposals

Here we want to present some improvement proposals of the processing that we are aware of but we did not have the time frame to realise them.

### Precise reference images

We have observed in the tests that the quality of reference images is a crucial factor, especially the image aspect ratio has to be on point. Unfortunately we do not have that much precise reference images of buildings yet, because we do not know the exact size of the buildings facades. The loss of accuracy by not having precise reference images is well illustrated in the image 5.3. By having more precise references images not only the loss will diminish but also the detection would be more successful as well. So we see that as the most important point of improvement.

### ASIFT

We discovered the ASIFT algorithm in the OpenCV documentation just as we were finishing our thesis, so we did not have enough time to re-search this method anymore. It is implemented in OpenCV under the name `AffineFeature` as a part of the *features2d* module. The ASIFT algorithm is



Figure 5.3: Possible gain of accuracy in case of improving reference images. The green quadrilateral is what has been detected and the red quadrilateral is the final product of a volume understanding. The difference between them is then the accuracy loss that was caused by having inaccurate reference images. Photo: author

a modification of the SIFT algorithm that improves the SIFT affine transformations invariance [38]. This could help to detect buildings, even if they were photographed from a sharper angle.

### 5.3.1 Future modules data requirements

In this section we write about the possible future improvement of the localisation that would require more input data than what we specified in the section 4.2.1.

#### Localisation in hilly areas

The localisation in hilly areas is problematic if we want to know the global coordinates of the mobile device, because the altitude is not included in the format of the global coordinates. In this case, it would be necessary to know the precise real world up vector, which could be obtained from the phone sensors or calculated if the detected object has a vertical facade. Another requirement would be to have access to the altitude for the given coordinates. With this information, it would be possible to calculate the exact location even in hilly areas, for example by an iterative method with an initial estimate of the altitude from the phone sensors.

### **Buildings given based on location**

Another improvement would be if the API of the mobile application gave the module only objects that are inside a circle with a given radius and center around some global coordinates.

### **Descriptors instead of images**

To detect objects, it is only needed to have the descriptors of the features detected in the reference image. So it would be better to use already precalculated descriptors of the reference images in the application. It would be appropriate if the descriptors were precalculated by multiple methods. Because then it would be possible to use faster (but less accurate) feature detection and extraction methods for devices with lower computation power and vice versa.

### **Dynamical scaling of scene images**

Despite the fact that the used local image features are to some extent invariant to the scale of the image they have been detected in, the feature is more likely to be detected in both of two different images if their scale is similar. The scene image would be probably scaled anyway to improve the speed of the computations. For this reason it would be convenient to scale the image so the building in the scene image has a similar resolution as the resolution of the image in the database. This can be achieved by guessing the scale factor from the distance between the location of the phone and the building based on the output of the phone positioning sensors.

---

## Conclusion

We continued the work started by Jan Šefčík. We improved the image recognition processing and moved the localisation from within image object localisation into 3D space and real world localisation. First, we have researched the used technologies, the *Dowry Towns of Bohemian Queens* project and the theses within the project connected to our. After that we analysed the researched technologies, theses and technologies used in the project. There we described the concepts of current designs and realisations already created in the project and then explained the changes we want to make in the design. Based on this, we realised a prototype of mobile device localisation based on image recognition. We wanted to show the performance of the prototype on tests created with the intention of getting as close as possible to the future use of the localisation module. These tests have shown that the device localisation based on image recognition is, on average, more accurate than phone sensors. Also we have observed that by having better quality of reference images the accuracy and reliability will get even higher by recognisable margin.

During the solution, we considered making not only the prototype but also the localisation module for the DTBQ application. Unfortunately this was not possible because the application itself was prepared only one month before the thesis submission deadline. We wanted to finish a fully functional localisation solution, so we spent the rest of the time completing the prototype.

In addition, we outlined the future development of the localisation application module. We have gained a lot of knowledge about this issue and we would like to help develop the *Dowry Towns of Bohemian Queens* project by finishing the localisation module for the mobile application. Therefore, we plan to continue working on it also after submitting the bachelor thesis.



---

## Bibliography

1. ŠEFČÍK, Jan. *Rozpoznávání a editace urbanistické scény*. Praha, 2020. Bachelor thesis. Czech Technical University in Prague, Faculty of Information Technology.
2. ŽÁRA, Jiří; BENEŠ, Bedřich; SOCHOR, Jiří; FELKEL, Petr. *Moderní počítačová grafika*. 2nd ed. Praha: Computer Press, 2005. ISBN 80-251-0454-0.
3. MCHUGH, Sean. *Understanding gamma correction* [online]. c2005–2020 [visited on 2020-11-28]. Available from: <https://www.cambridgeincolour.com/tutorials/gamma-correction.htm>.
4. POYNTON, Charles. *Color FAQ - Frequently Asked Questions Color* [online] [visited on 2020-11-27]. Available from: [http://poynton.ca/notes/colour\\_and\\_gamma/ColorFAQ.html](http://poynton.ca/notes/colour_and_gamma/ColorFAQ.html).
5. *How to Convert an RGB Image to Grayscale* [online] [visited on 2020-11-26]. Available from: [e2eml.school/convert\\_rgb\\_to\\_grayscale.html](http://e2eml.school/convert_rgb_to_grayscale.html).
6. *What is Histogram Equalization and how it works?* [Online] [visited on 2020-11-28]. Available from: <https://www.mygreatlearning.com/blog/histogram-equalization-explained/>.
7. MIKOLAJCZYK, Krystian; TUYTELAARS, Tinne. Local Image Features. In: *Encyclopedia of Biometrics*. Ed. by LI, Stan Z.; JAIN, Anil K. Boston, MA: Springer US, 2015, pp. 1100–1105. ISBN 978-1-4899-7488-4. Available from DOI: 10.1007/978-1-4899-7488-4\_224.
8. GUO, Z.; ZHANG, L.; ZHANG, D. A Completed Modeling of Local Binary Pattern Operator for Texture Classification. *IEEE Transactions on Image Processing*. 2010, vol. 19, no. 6, pp. 1657–1663. ISSN 1941-0042. Available from DOI: 10.1109/TIP.2010.2044957.

9. LOWE, David G. *Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image*. Google Patents, 2004. US Patent 6,711,293.
10. LOWE, David G et al. Object recognition from local scale-invariant features. In: *iccv*. 1999, vol. 99, pp. 1150–1157. No. 2.
11. AMERINI, I.; BALLAN, L.; CALDELLI, R.; DEL BIMBO, A.; SERRA, G. A SIFT-Based Forensic Method for Copy–Move Attack Detection and Transformation Recovery. *IEEE Transactions on Information Forensics and Security*. 2011, vol. 6, no. 3, pp. 1099–1110. ISSN 1556-6021. Available from DOI: 10.1109/TIFS.2011.2129512.
12. MIKOLAJCZYK, K.; SCHMID, C. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2005, vol. 27, no. 10, pp. 1615–1630. ISSN 1939-3539. Available from DOI: 10.1109/TPAMI.2005.188.
13. ARANDJELOVIĆ, Relja; ZISSERMAN, Andrew. Three things everyone should know to improve object retrieval. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 2911–2918.
14. SUÁREZ, Iago; SFEIR, Ghesn; BUENAPOSADA, José M.; BAUMELA, Luis. BEBLID: Boosted Efficient Binary Local Image Descriptor. *Pattern Recognition Letters*. 2020. ISSN 0167-8655. Available from DOI: <https://doi.org/10.1016/j.patrec.2020.04.005>.
15. SZELISKI, Richard. *Computer vision: Algorithms and applications*. 2011th ed. Guildford, England: Springer, 2010.
16. YAN KE; SUKTHANKAR, R. PCA-SIFT: a more distinctive representation for local image descriptors. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004*. 2004, vol. 2, pp. II–II. ISSN 1063-6919. Available from DOI: 10.1109/CVPR.2004.1315206.
17. MORDVINTSEV, Alexander; K., Abid Rahman. *Feature Matching* [online]. c2013 [visited on 2020-12-16]. Available from: [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_feature2d/py\\_matcher/py\\_matcher.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html).
18. MUJA, M.; LOWE, D. G. Fast Matching of Binary Features. In: *2012 Ninth Conference on Computer and Robot Vision*. 2012, pp. 404–410. Available from DOI: 10.1109/CRV.2012.60.
19. LOWE, David G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*. 2004, vol. 60, no. 2, pp. 91–110.



20. FISCHLER, Martin A.; BOLLES, Robert C. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*. 1981, vol. 24, no. 6, pp. 381–395. ISSN 0001-0782. Available from DOI: 10.1145/358669.358692.
21. RIBA, Edgar. *Real Time pose estimation of a textured object* [online]. OpenCV team, 2021 [visited on 2021-05-06]. Available from: [https://docs.opencv.org/master/dc/d2c/tutorial\\_real\\_time\\_pose.html](https://docs.opencv.org/master/dc/d2c/tutorial_real_time_pose.html).
22. *O projektu: Věnná města českých královen* [online]. Univerzita Hradec Králové [visited on 2020-11-25]. Available from: <https://www.kralovskavennamesta.cz/about.html>.
23. VANČURA, Daniel. *Věnná města českých královen - jádro*. Praha, 2020. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
24. ŠTĚPÁN, Jaroslav. *Věnná města českých královen - Modul rozpoznání obrazu*. Praha, 2019. Master Thesis. Czech Technical University in Prague, Faculty of Information Technology.
25. *About OpenCV* [online]. OpenCV team, 2021 [visited on 2021-04-13]. Available from: <https://opencv.org/about/>.
26. *OpenCV License* [online]. OpenCV team, 2021 [visited on 2019-12-12]. Available from: <https://opencv.org/license/>.
27. *Apache License, Version 2.0* [online]. Wilmington, DE 19801 U.S.A.: The Apache Software Foundation, 2004-01 [visited on 2021-04-13]. Available from: <https://www.apache.org/licenses/LICENSE-2.0>.
28. *The 3-Clause BSD License* [online]. West Hollywood, CA 90069-4109 United States: Open Source Initiative [visited on 2021-04-13]. Available from: <https://opensource.org/licenses/BSD-3-Clause>.
29. HUAMÁN, Ana. *Features2D + Homography to find a known object* [online]. OpenCV team, 2021 [visited on 2021-04-15]. Available from: [https://docs.opencv.org/4.5.2/d7/df/tutorial\\_feature\\_homography.html](https://docs.opencv.org/4.5.2/d7/df/tutorial_feature_homography.html).
30. *ML Kit* [online]. Google LLC [visited on 2021-04-15]. Available from: <https://developers.google.com/ml-kit/>.
31. *ARCore* [online]. Google LLC [visited on 2021-04-15]. Available from: <https://developers.google.com/ar>.
32. SUÁREZ, Iago. *BEBLID: Boosted Efficient Binary Local Image Descriptor* [online]. Madrid: GitHub, Inc., 2020 [visited on 2021-04-14]. Available from: <https://github.com/iago-suarez/BEBLID>.
33. POPOVA, Tatiana. *Urban scene recognition and editing II*. Praha, 2021. Bachelor Thesis. Czech Technical University in Prague, Faculty of Information Technology.

## BIBLIOGRAPHY

---

34. *Releases* [online]. OpenCV team, 2021 [visited on 2021-05-05]. Available from: <https://opencv.org/releases/>.
35. GÁBOR, Bernát. *Installation in Windows* [online]. OpenCV team, 2021 [visited on 2021-05-05]. Available from: [https://docs.opencv.org/master/d3/d52/tutorial\\_windows\\_install.html](https://docs.opencv.org/master/d3/d52/tutorial_windows_install.html).
36. *Getting Started on Windows* [online]. 2021 [visited on 2021-05-05]. Available from: [https://www.boost.org/doc/libs/1\\_76\\_0/more/getting\\_started/windows.html](https://www.boost.org/doc/libs/1_76_0/more/getting_started/windows.html).
37. VOJTÍŠKOVÁ, Jana. Případová studie k zaniklým objektům královského věnného města Hradce Králové. In: *Královéhradecko: historický sborník pro poučenou veřejnost*. 10, 2019. Hradec Králové: Státní okresní archiv, 2019, pp. 355–363. ISBN 978-80-87686-25-6. ISSN 1214-5211.
38. YU, Guoshen; MOREL, Jean-Michel. ASIFT: An algorithm for fully affine invariant comparison. *Image Processing On Line*. 2011, vol. 1, pp. 11–38.

---

## List of used abbreviations

- AHE** Adaptive histogram equalisation
- API** Application programming interface
- BEBLID** Boosted efficient binary local image descriptor
- BELID** Boosted efficient local image descriptor
- CIE** International Commission on Illumination
- CLAHE** Contrast limited adaptive histogram equalization
- CRUD** Create, read, update, and delete
- DTBQ** Dowry Towns of Bohemian Queens
- FLANN** Fast Library for Approximate Nearest Neighbors
- GPS** Global Positioning System
- IDE** Integrated development environment
- JNI** Java native interface
- LBP** Local binary patterns
- ORB** Oriented FAST and Rotated BRIEF
- PnP problem** Perspective-n-Point problem
- RANSAC** Random Sample Consensus
- SIFT** Scale-invariant feature transform



---

## Contents of the attached data storage

readme.txt.....	brief description of the data storage contents
exe.....	directory with executable form of implementation
├─ image_database.....	image database
├─ config.....	directory with configuration
src	
├─ impl.....	implementation source codes
├─ thesis.....	source form of work in L <sup>A</sup> T <sub>E</sub> X format
text.....	thesis text
├─ thesis.pdf.....	thesis text in PDF format



---

## Test results

In this appendix are all the results from testing. They are in the form of two pictures and a caption. The left picture is an aerial map, where the detected building's facade is marked with a yellow line. The global location calculated by our prototype is then located at the tip of the green pin and the location detected by the phone sensor is at the tip of the cyan pin. The right image represents the recognised volume of the building (the place where the building is located). It should be taken into account that the aerial map may not accurately correspond to the real location of the buildings. All the input scene photos were taken by us or from the image database by Tatiana Popova [33].

The caption has the following format: **Our:** global coordinates; **phone:** global coordinates; **difference:** distance; **note:**(optionally) text. Where **Our** represents a field with the global coordinates calculated by our prototype and **phone** then those which have been detected by the phone positioning sensor. **Difference** is then the distance between **Our** and **phone**. The distance value is preceded by a plus if the result was evaluated as more accurate for **Our** or a minus if the **phone** was more accurate. The question mark is then used for undecidable cases and  $-\infty$  where our prototype failed (then there is also no green pin on the map).

## C. TEST RESULTS

---

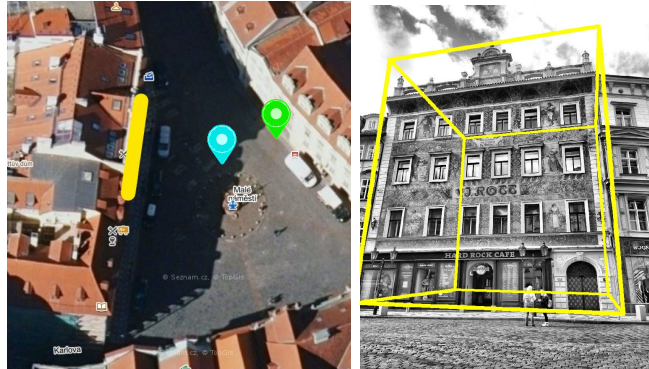


Figure C.1: Our: 50.0867079, 14.4196045; phone: 50.0866778, 14.4194886; difference: +8.92 m. Source of the aerial map: Mapy.cz

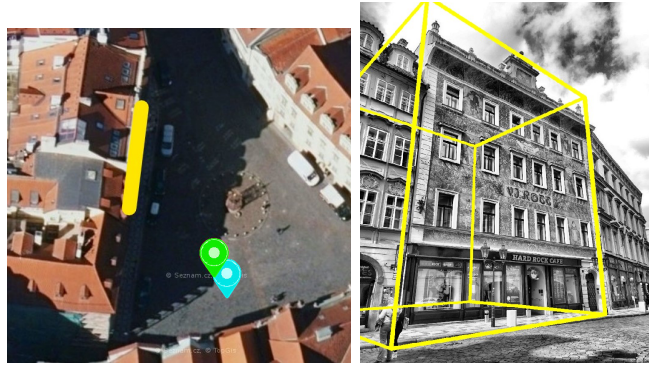


Figure C.2: Our: 50.0865094, 14.4194608; phone: 50.0864750, 14.4194681; difference: ?3.86 m. Source of the aerial map: Mapy.cz

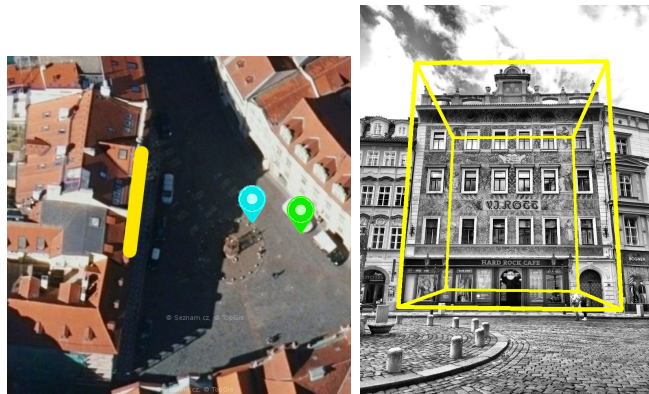


Figure C.3: Our: 50.086637, 14.4196669; phone: 50.0866556, 14.4195519; difference: +8.46 m. Source of the aerial map: Mapy.cz



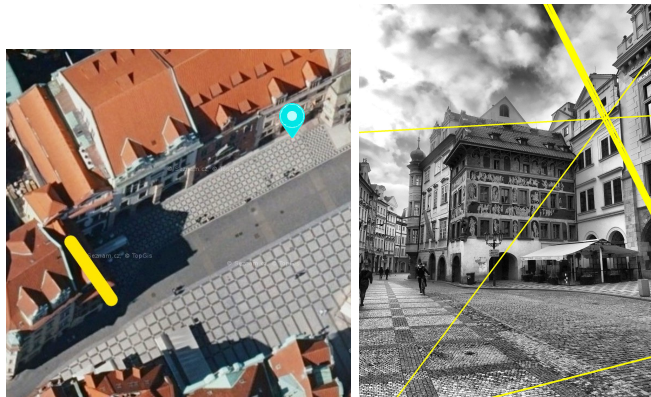


Figure C.4: Our: 50.0872907, 14.4213531; phone: 50.0869444, 14.4205556;  
difference:  $-\infty$  m. Source of the aerial map: Mapy.cz

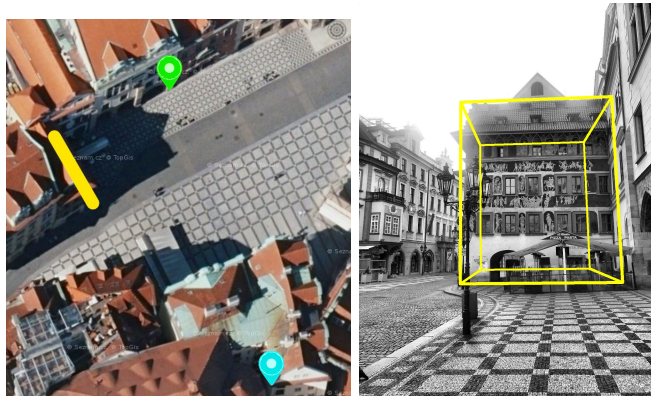


Figure C.5: Our: 50.0868708, 14.420301; phone: 50.0863889, 14.4205556;  
difference: +56.58 m. Source of the aerial map: Mapy.cz

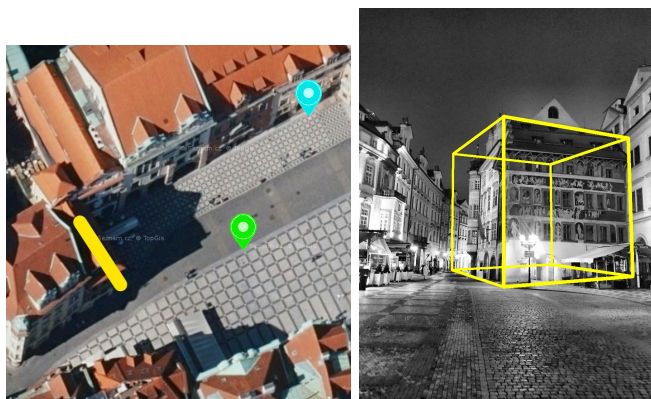


Figure C.6: Our: 50.0867419, 14.4204009; phone: 50.0869444, 14.4205556;  
difference: +25.08 m. Source of the aerial map: Mapy.cz

## C. TEST RESULTS

---

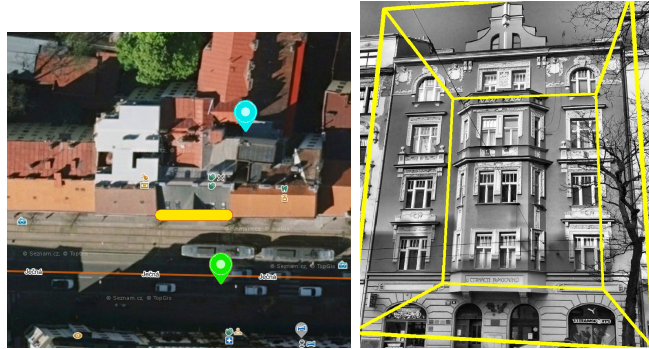


Figure C.7: Our: 50.0755673, 14.4229881; phone: 50.0758333, 14.4230556;  
difference: +29.97 m. Source of the aerial map: Mapy.cz

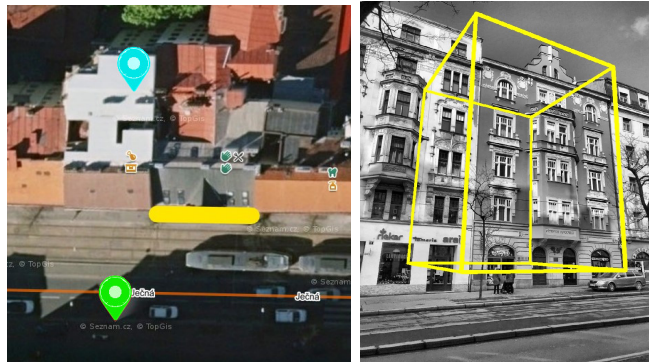


Figure C.8: Our: 50.0755462, 14.4227431; phone: 50.0758333, 14.4227778;  
difference: +32.02 m. Source of the aerial map: Mapy.cz

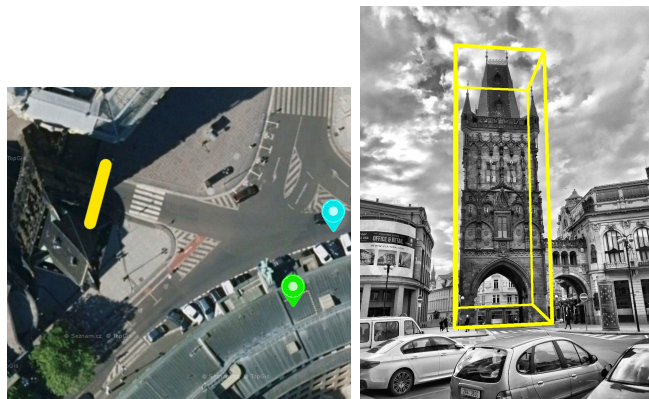


Figure C.9: Our: 50.0870019, 14.4284049; phone: 50.0871414, 14.4285222;  
difference: ?17.63 m. Source of the aerial map: Mapy.cz

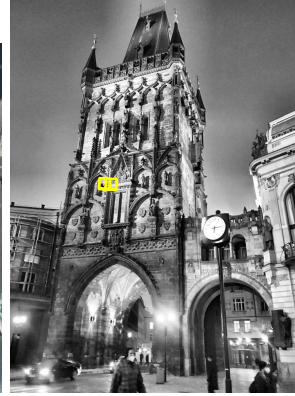
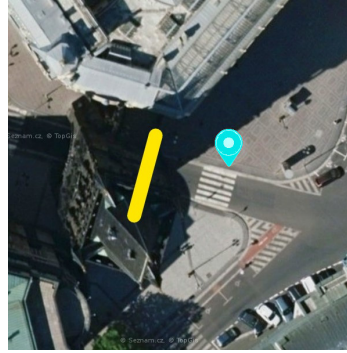


Figure C.10: Our: 50.0857078, 14.4191975; phone: 50.0872222, 14.4280556; difference:  $-\infty$  m. Source of the aerial map: Mapy.cz

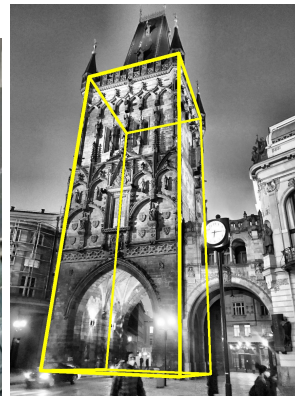
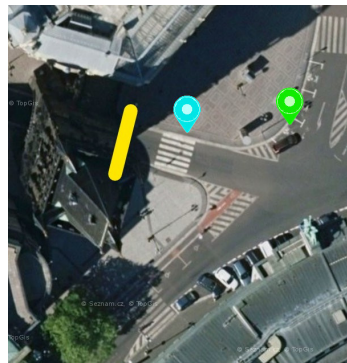


Figure C.11: Our: 50.0872281, 14.4282983; phone: 50.0872222, 14.4280556; difference: +17.33 m; note: In comparison to the test C.10 this test was done after the night reference building image was added to the database. Source of the aerial map: Mapy.cz



## C. TEST RESULTS

---

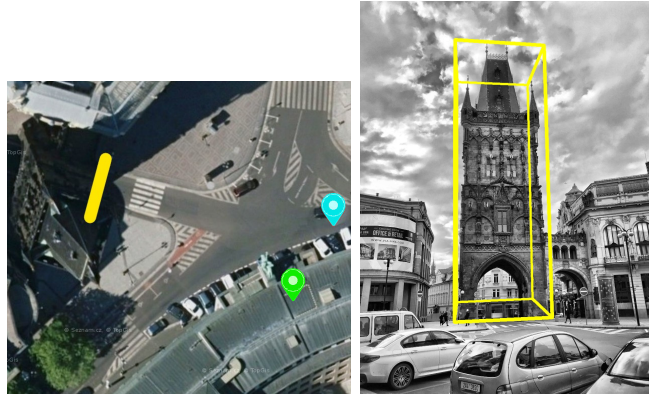


Figure C.12: Our: 50.0870014, 14.4284044; phone: 50.0871414, 14.4285222; difference: ?17.69 m; note: In comparison to test C.9 this test was done after the night reference building image was added to the database. Small difference in output was created, because the Lowe's ratio threshold was changed from 0.7 to 0.75. Source of the aerial map: Mapy.cz

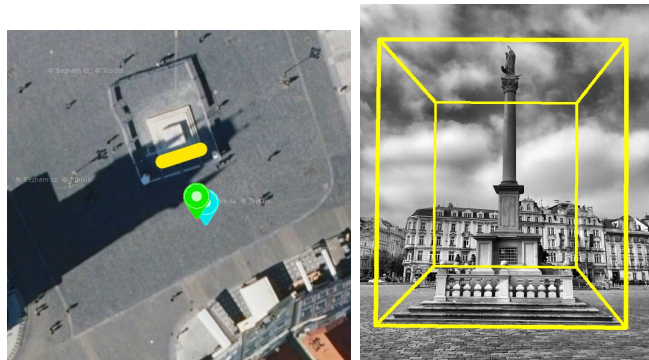


Figure C.13: Our: 50.087235, 14.4213759; phone: 50.0872222, 14.4213889; difference: ?1.70 m. Source of the aerial map: Mapy.cz

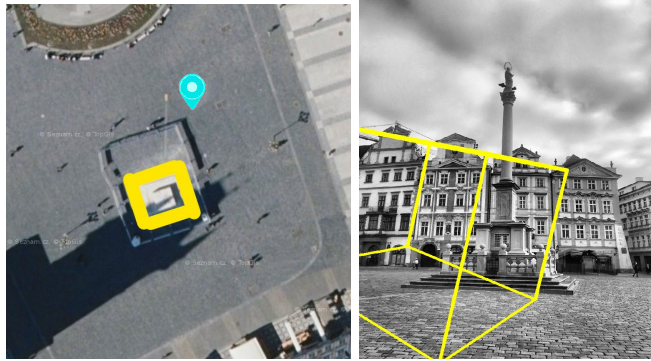


Figure C.14: Our: 50.0669414, 14.4487256; phone: 50.0875000, 14.4213889; difference:  $-\infty$  m; note: The reference image of the column from that side was missing. Source of the aerial map: Mapy.cz

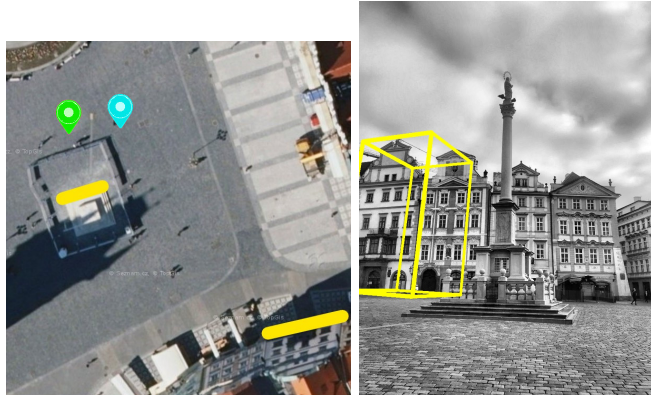


Figure C.15: Our: 50.0874903, 14.4212677; phone: 50.0875000, 14.4213889; difference: +8.71 m; note: This test was made after the test C.14 and the difference was that the the reference image of the detected building in the background was improved. Source of the aerial map: Mapy.cz

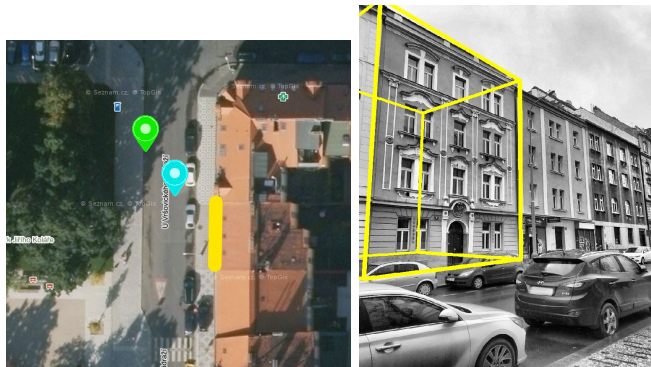


Figure C.16: Our: 50.0668904, 14.4486542; phone: 50.0668333, 14.4487250; difference: +8.11 m. Source of the aerial map: Mapy.cz

## C. TEST RESULTS

---

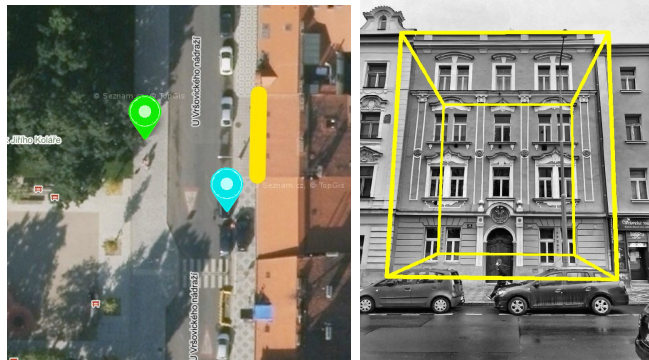


Figure C.17: Our: 50.0667679, 14.4486013; phone: 50.0666778, 14.4487639; difference: +15.33 m. Source of the aerial map: Mapy.cz

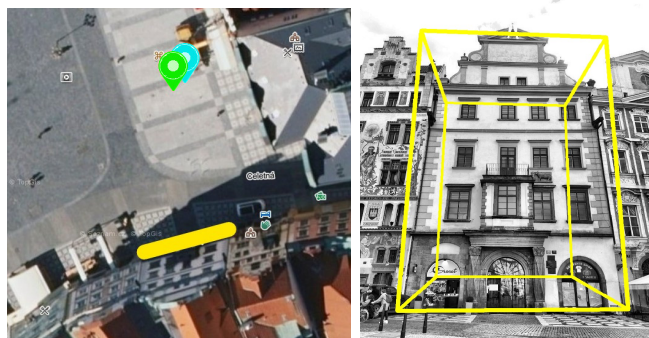


Figure C.18: Our: 50.0873903, 14.421842; phone: 50.0874025, 14.4218625; difference: +1.99 m. Source of the aerial map: Mapy.cz

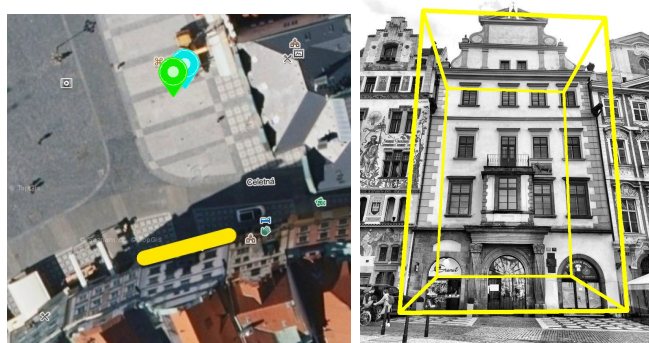


Figure C.19: Our: 50.0873903, 14.4218407; phone: 50.0874025, 14.4218625; difference: +2.06 m; note: This test was done after adding improved reference image of the building (difference to test C.18), but the old reference image was still left in the database (difference to test C.20). The old reference image was detected instead of the improved one. Source of the aerial map: Mapy.cz



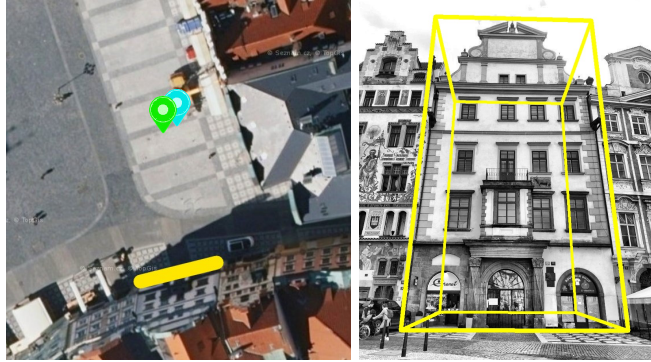


Figure C.20: Our: 50.0873926, 14.4218324; phone: 50.0874025, 14.4218625; difference: +2.41 m; note: The test was done after improving the reference image of the building (difference to test C.18). Source of the aerial map: Mapy.cz

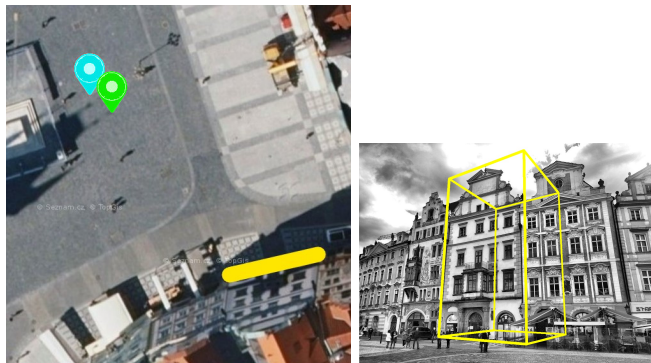


Figure C.21: Our: 50.0873907, 14.4215317; phone: 50.0874142, 14.4214806; difference: +4.49 m. Source of the aerial map: Mapy.cz

## C. TEST RESULTS

---

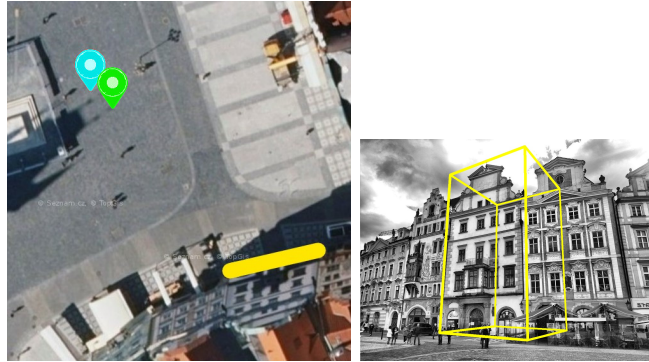


Figure C.22: Our: 50.0873907, 14.4215317; phone: 50.0874142, 14.4214806; difference: +4.49 m; note: In comparison with the test C.24) was this test made after adding to the database the house to the left of the detected house. The test was made with the intention to find out which building will be detected. Source of the aerial map: Mapy.cz

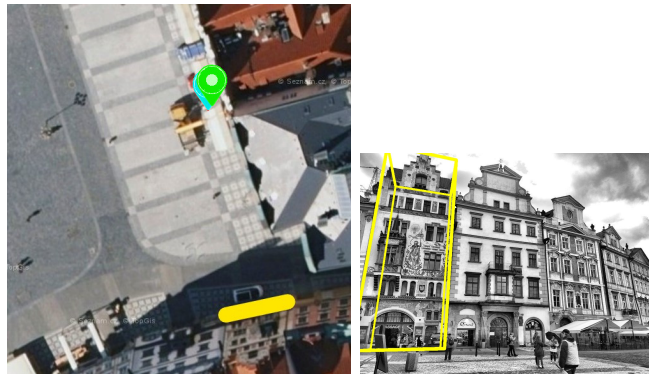


Figure C.23: Our: 50.0874767, 14.4219311; phone: 50.0874733, 14.4219272; difference: ?0.47 m; note: This test was also made with the intention to observe the behavior of the detection when, there are two reference images in the scene. Source of the aerial map: Mapy.cz



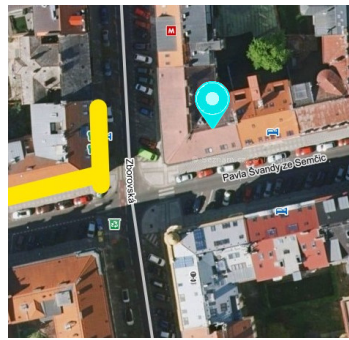


Figure C.24: Our: 50.0819741, 14.4506596; phone: 50.0772222, 14.4075000; difference:  $-\infty$  m. Source of the aerial map: Mapy.cz

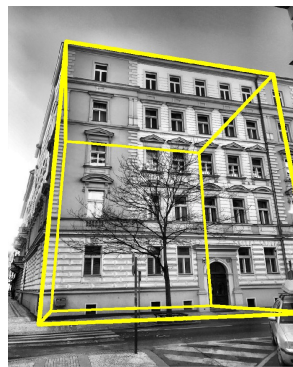


Figure C.25: Our: 50.0771431, 14.4073852; phone: 50.0769444, 14.4072222; difference: +24.97 m. Source of the aerial map: Mapy.cz