Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis

Integrating Augmented Reality Data Into a Mobile Simulation Framework

Bondar Bogdan

Course of Study:	INFOTECH
Examiner:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Supervisor:	Dipl.-Inf. Christoph Dibak
Commenced:	19. April 2017
Completed:	19. October 2017
CR-Classification:	I.6.6

Abstract

Augmented reality devices can be used by engineers and scientists to perform numerical simulations for different physical systems. Using augmented reality glasses for presenting simulation results, in the form of virtual objects, improves interactivity and quality of user experience (QoE). However, despite availability of efficient approaches for fast approximate computations, such as the reduced basis method (RBM), it is not feasible to preload all reduced models for different simulations and sets of parameters in advance. Therefore, an approach is presented, that allows to detect real-world objects, map specific tags to corresponding simulations and to efficiently use available storage. By analyzing the environmental sensor data and estimating the required quality, determined by user perception, only relevant reduced bases are loaded to the internal storage of mobile device. The proposed system periodically obtains solution for the given simulation in accordance with changed parameters and quality demands, either by loading updated reduced model from server or performing the computation of the full problem directly on the mobile device in case of network failure. Simulation results are visualized by means of hologram, overlaid onto the detected object with exact position and pose. Evaluation results show that solving the full numerical problem on mobile device is feasible for dimensions up to 32x32, whereas for higher quality constraints the latency of requesting the reduced model from server can be improved by up to 10 times by performing preliminary availability check and starting computation in advance.

Contents

1 Introduction 11
- 1.1 Motivation . . . 11
- 1.2 Objective . . . 12
- 1.3 Approach . . . 13
- 1.4 Structure . . . 14

2 Background and Related Work 15
- 2.1 Augmented Reality . . . 15
  - 2.1.1 Microsoft Hololens Capabilities . . . 16
  - 2.1.2 Camera-space to World-space Coordinates Transformation . . . 17
  - 2.1.3 Object Detection . . . 18
  - 2.1.4 Marker-based tracking . . . 19
- 2.2 Numerical Simulation . . . 21
  - 2.2.1 Finite Difference Method . . . 22
  - 2.2.2 Finite Elements Method . . . 22
  - 2.2.3 Finite Volumes Method . . . 23
  - 2.2.4 Advection–Diffusion Equation . . . 23
- 2.3 Reduced Basis Method . . . 24
  - 2.3.1 Subspace construction . . . 25
  - 2.3.2 Parameter-separability . . . 25
  - 2.3.3 Offline/Online Decomposition . . . 26
  - 2.3.4 Error estimation . . . 26
- 2.4 Code Offloading . . . 26
  - 2.4.1 Safe-points . . . 27
  - 2.4.2 Server cache . . . 28
- 2.5 Distributed Reduced Basis Method . . . 28
  - 2.5.1 Basic approach . . . 29
  - 2.5.2 Adaptive approach . . . 29

3 System Model and Problem Statement 31
- 3.1 System Model . . . 31
  - 3.1.1 Mobile device . . . 32
  - 3.1.2 Server . . . 33

3.1.3 Numerical Simulation . . . 33
3.1.4 Middleware . . . 33
3.1.5 User Application . . . 34
3.2 Problem Statement . . . 34

4 Methodology 39
4.1 Approach overview . . . 39
4.2 Simulation tag mapping . . . 41
4.3 Training data collection . . . 44
4.4 Simulation quality selection . . . 45
4.5 Reduced model request . . . 47
4.6 Execution strategy selection . . . 48
4.7 Result visualization . . . 51

5 Implementation 53
5.1 Numerical libraries . . . 53
5.2 Client-side middleware . . . 54
5.3 User application . . . 57
5.4 Virtual object construction . . . 61

6 Evaluation 63
6.1 Evaluation Setup . . . 63
6.2 Numerical libraries comparison . . . 65
6.3 Object recognition . . . 68
6.4 Basis request latency . . . 70
6.5 Execution strategies comparison . . . 72
6.6 Data visualization . . . 73

7 Summary and Outlook 77
7.1 Summary . . . 77
7.2 Future work . . . 79

Bibliography 81

List of Figures

2.1 Transformation from world reference frame to camera reference frame adopted from [CFA+10]. 18
2.2 Marker detection, identifications and pose estimation flow adopted from [AM04]. 20
2.3 RB-approximation adopted from [Haa16]. 25
2.4 Code offloading architecture adopted from [CBC+10]. 27

3.1 System components and their interconnections. 31
3.2 Interaction with different systems through mobility. 32
3.3 High level overview of the problem. 35
3.4 Reduced model selection and limitations. 36

4.1 Overview of the individual steps of the approach. 40
4.2 Mismatch between the identified and real-world position and rotation. . 42
4.3 Full problem discretization parameter selection depending on the distance to marker. 45
4.4 Request flow for basis availability verification and preloading for server cache miss scenario. 47
4.5 Flowchart of solver selection procedure. 49
4.6 Full problem solution flow. 51

5.1 UML class diagram for numerical libraries, used for solving the full problem. 54
5.2 UML class diagram for client-side middleware. 55
5.3 UML class diagram for user application. 58
5.4 Heatmap holograms, using custom shader, for different discretization parameter D. 61

6.1 Augmented reality glasses, fiducial markers and programmable sensor, used for building the prototype. 64
6.2 Fiducial markers, used for object tracking. 65
6.3 Numerical libraries memory(RAM) consumption comparison for solving full numerical problem. 65
6.4 Numerical libraries execution time comparison for solving full numerical problem. 66

6.5 ARToolkit marker recognition and alignment correction. 68
6.6 Vuforia target image recognition and alignment. 69
6.7 Latency comparison when the basis is not available on the server (cache miss) for different training sets. 70
6.8 Latency comparison when the basis is available on the server (cache hit) for different training sets. 71
6.9 Simulation solution calculation time for different scenarios. 73
6.10 Performance comparison for heatmap hologram rendering, using different construction strategies. 74
6.11 Heatmap holograms, constructed using dynamic texture generation and pose estimation. 75
6.12 Parallel execution of two distinct simulation instances, using advection-diffusion numerical problem. 75

List of Algorithms

4.1 Marker detection and simulation mapping 43
4.2 Training set construction 44
4.3 Simualtion quality determination 46
4.4 Requesting basis from server 48
4.5 Reduced Basis Method Strategy 50
4.6 Visualization of simulation solution 52

1 Introduction

1.1 Motivation

Nowadays more and more processes in industry and science need some kind of automation and support from the field of computer science. Knowing the physical or technological background of some phenomenon is not always sufficient for detailed analysis of all the processes involved. In order to solve some specific problem, additional data should be available from external sources to monitor the state in real time. Moreover, this additional data should be processed and presented to the specialists in the most accurate and perceptible form, so they could react and adapt the conditions accordingly. This approach facilitates dynamic analysis and evaluation of the task by letting the experts pay more attention to design rather than time-consuming manual parameter verification. With the help of algorithms that collect, process and output the data related to some physical process, scientists and engineers can better understand the problem to be solved and perform real-time parameter adaptation. The later assists in finding the optimized initial parameter set and getting the expected system output.

In the era of Internet of Things and Ubiquitous computing millions of sensors and smart autonomous devices distribute and broadcast all possible useful data about surrounding environment. The main challenge is to recognize, aggregate and fetch only relevant information for specific application. Moreover, after collecting all the data needed, it should be integrated into some physical model that should be meaningful in the current context. Such a model can be based on the numerical simulations of different processes in some dynamic physical system, that is often described by means of partial differential equations. With the numerical simulations it is feasible to analyze all the aggregated information from the huge amount of data sources and make the prediction of the future state of the corresponding physical system. This knowledge gives the opportunity to optimize some processes or reveal some unknown influence, that could be critical in many applications. The use cases could be: monitoring the heat dissipation in hardware manufacturing, predicting the climate change in some specific region, choosing and adapting the treatment in accordance with patients response, etc. [DK14].

Strictly speaking, there are two main approaches for performing the numerical simulations of any system under consideration: mobile computing, when the simulation is

performed on the mobile device directly in touch with the expert and physical process; cloud computing, when required physical parameters are sent to the cloud infrastructure that executes the simulation and returns the meaningful result to the specialist. Each of these approaches has its advantages and drawbacks. Performing the entire simulation on the mobile device results in high power consumption and increased interaction response due to limited resources and low processing power. However, collecting and preprocessing the data and presenting the simulation results to the user is definitely the task of the mobile device. On the other hand, cloud computing can provide high performing, specialized hardware and software components to solve large dynamical numerical simulations in the most optimal manner. The trade-off is achieved by adapting the quality of mobile simulation with degraded results for mobile computing in case of low bandwidth or offloading the calculations to external server infrastructure in case of stable connection for high quality constraints [DDR15].

Recent raise in computer power, researches in computer vision and visualization techniques revealed new kind of augmented reality mobile devices. This type of devices can simultaneously fetch the surrounding sensor values, recognize objects and patterns from real world and additionally construct virtual holograms or provide auxiliary information to the user. All these characteristics make augmented reality devices, such as Microsoft Hololens, the perfect choice for performing numerical simulations, either on their own, or by means of communication with the server and presenting results visually to decision makers in the field. With the real-time and predicted data available, specialist can reason about some physical processes, make changes and observe their influence on the system. But as stated in previous paragraph, the approach should be developed to perform simulation effectively on resource-constrained mobile device. One such solution is to distribute the load between mobile and server in most efficient way, by offloading complex calculations to the server and receiving the precomputed simulation model in a form such as to minimize the data flow and latency for presenting result to the user [DDR17; DSD+17]. Additionally, in dynamic environment, when the input data flow changes rapidly, results should be updated abruptly in order to keep quality constraints. Often simulations should be adapted or changed entirely depending on the location and context of the system. Apparently, it is not feasible for the mobile device to precompute or request all the models for different simulations and its parameters in advance, otherwise, it will lead to indefinite setup times and memory usage.

1.2 Objective

The objective of this thesis is to develop a new approach that enables mobile devices to dynamically detect changes in system context and react accordingly. Specifically,

augmented reality device should recognize specific objects, patterns and markers of the surrounding and differentiate between them in order to identify the corresponding simulation and request appropriate model from server. This approach should also allow to preprocess the data, perform specific simulation and present result to the user simultaneously for multiple detected objects.

One of the problems to be solved is to decide on the quality of the requested simulation model. On one hand, degrading the quality results in lower communication overhead-/latency and gives the possibility to execute entire simulation on mobile device, on the other hand, the quality should meet the corresponding constraints for adequate user perception. Thereby, specific method should be developed to decide on the execution strategy, depending on the input parameters: distance to the object of interest, availability of simulation results on the server, etc.

Another objective is to aggregate physical values of available sensors and integrate them into the simulation model, either as initial parameter set, mapping raw sensor data to simulation parameters, or as an updated set for dynamic model adaptation during run-time. Finally, the simulation results should be presented to the user in the form of virtual data (e.g. heat-map hologram), that visualizes physical processes (e.g. heat dissipation, water/wind flow) under consideration.

In order to accomplish these goals, dynamic system should be developed, that detects the simulation to be solved, collects and monitors sensor data changes, adapts simulation quality and constructs the augmented virtual objects for presenting results to the specialist in the most useful manner.

1.3 Approach

The method, presented in this thesis, serves for real-world objects recognition, in the form of predefined tags, and their mapping to corresponding simulations. Together with precalculated simulation parameters set, based on environmental sensor data, the detected tag is sent to server in order to load the reduced model of the identified simulation to mobile device. This reduced basis is then used for performing fast calculation of approximate solution on the mobile device, using the Reduced Basis Method (RBM) [DSD+17; Haa16]. When the solution of the numerical simulation is computed, the 2D hologram, based on this results, is constructed and overlaid on the found object with exact position and rotation.

Sensor values are periodically read to update the simulation results and present current state of the system. Additionally, the distance to the identified tag is constantly measured to request the updated reduced model with increased dimension from server, when the

user approaches the object. In case of network failure, developed approach allows to solve the full numerical problem with degraded quality directly on the mobile device for continuous system behavior tracking. Updated simulation results with enhanced quality are visualized by the hologram of higher resolution.

The developed prototype recognizes and differentiates between two kinds of real-world markers, that are mapped to advection-diffusion numerical simulation [DSD+17; KT17]. The simulation results for different set of parameters are updated and visualized for each recognized marker simultaneously and independently. So it can be easily extended for the vast majority of markers or other object recognition techniques in order to track and visualize different numerical simulations according to the physical model and gathered sensor values.

Visualizing simulation results with varying quality improves the overall interactivity with the user. Whereas dynamic tag recognition and parameters identification allows continuous monitoring of the current system state without the need to load all possible reduced models to mobile device in advance.

1.4 Structure

The structure of this Master Thesis is divided into seven chapters. In chapter 2 the background knowledge is described, which is needed to understand the rest of the paper, as well as related work, that tries to solve similar problems. Chapter 3 deals with the system model design and states the problem to be solved. The methodology is described in chapter 4. Chapter 5 presents implementation details of the developed design. In chapter 6 proposed approach is evaluated and corresponding test results are shown with the help of the developed prototype. Finally, chapter 8 closes the paper with summary and overview of future work.

2 Background and Related Work

In this chapter the background knowledge is described, that is important for understanding the rest of the thesis.

First of all the concept of augmented reality will be presented that will highlight its applications and limitations. The method of going from camera-space to world-space coordinate system will be shown. Then, the capabilities of the immersive augmented reality headset will be investigated, taking Microsoft Hololens as the example. Following by the object detection and pose estimation algorithms based on marker image recognition. Finally, the technique of data visualization will be examined for constructing the heat-map hologram.

In order to solve physical problems, that are mainly described by the partial differential equations (PDE), the numerical methods should be considered for constructing the algebraic systems, that can be processed on mobile device, or on the server infrastructure. As the example, the advection-diffusion equation is considered, that describes the heat distribution in some material.

However, it is not always feasible and often not required to get the exact solution of the numerical simulation. In most cases approximate result is sufficient. For that purpose the Reduced Bases Method (RBM) is studied. With the pure approach most of the complex calculations are performed on the server, utilizing the code offloading technique, that is reviewed in the following subsection.

At the end of background chapter the overview of computations distribution method between client (mobile device) and server is given, describing the approach of Distributed Reduced Basis Method (DRBM).

2.1 Augmented Reality

Augmented Reality (AR) allows to enhance the real-world environment with virtual object that can be constructed with computer on the bases of processed data. This technique, while simultaneously combining real and virtual objects, gives the opportunity for better interaction with the augmented information. Additional characteristics of the

real-world object or process under consideration are attached in the form of holograms directly to the object itself [CFA+10]. Despite the broad definition of augmenting not only sight, but potentially all senses, AR is most usable when it is applied to head-mounted displays (HMD).

2.1.1 Microsoft Hololens Capabilities

In Chapter 1 it was already argued that with the recent advances in computing capabilities and technology maturity it is now possible to utilize mobile devices for numerical simulations. One of such promises devices is the optical see-through HMD - Microsoft Hololens. In order to understand its benefits and limitations, there is a need to firstly inspect its hardware [EMP+17]:

- **Logical Processor:** 4xIntel Atom x5-Z8100 1.04 GHz;
- **Video Rendering:** HPU/GPU Holographic Processing Unit;
- **Memory:** 64 GB Flash, 2GB RAM;
- **Sensors:** Inertial Measurement Unit (IMU), 4 microphones, ambient light sensor;
- **Communication:** Bluetooth Low Energy (BLE) 4.1, WiFi 802.11ac;
- **Cameras:** 4 environment-processing cameras, RGB camera and 1 depth camera for mapping surrounding;
- **Additional Capabilities:** gaze tracking, gesture input, spatial sound and voice support;
- **Standby Time:** 2-3 hours.

Additionally, Microsoft Hololens runs on Windows 10 operation system, thus the applications should be developed with Universal Windows Platform (UWP) technology, that, among other things, includes extended API for network communication, supports multiple numerical libraries integration and collecting data from external sensors through BLE communication. For building user interface (UI) and data visualization the cross-platform framework - Unity3D is officially recommended by Microsoft [EMP+17].

2.1.2 Camera-space to World-space Coordinates Transformation

The AR devices, such as Microsoft Hololens, also provides the opportunity to utilize simple computer vision methods. Utilizing the video tracking technique special features of some predefined object or image can be detected. Additionally, the pose of the detected image can be obtained when the correct projection from camera coordinates to image coordinates is done. The pose is defined through location, that is described with translation coordinates (x, y, z), and orientation, described by rotation angles (α, β, γ) [Sil12].

Let the point have coordinates $(x, y, z)^T$ in the camera coordinate frame, then its projection on the image plane is $(x/z, y/z, 1)^T$. Thus, for the set of non-collinear reference points $p_i(x_i, y_i, z_i)^T$, where $i = 1, ..., n; n \geq 3$ in the world-space coordinates and corresponding camera-space coordinates $q_i(x'_i, y'_i, z'_i)^T$, the relation between p_i and q_i is described by the following equation:

$$q_i = Rp_i + T \tag{2.1}$$

$$\text{where } R = \begin{pmatrix} r_1^T \\ r_2^T \\ r_3^T \end{pmatrix} \text{ and } T = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{2.2}$$

are rotation matrix and translation vector, respectively [CFA+10].

Strictly speaking, the rotation matrix R consists of three fractions:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \text{ and } R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \text{ and } R_z = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Suchwise arbitrary rotation, depending on the rotation order, can be expressed, for example as $R = R_z R_y R_x$ [Sil12]:

$$R = \begin{bmatrix} \cos\beta\cos\gamma & \cos\gamma\sin\alpha\sin\beta - \cos\gamma\sin\alpha & \cos\gamma\sin\beta\cos\alpha + \sin\alpha\sin\gamma \\ \cos\beta\sin\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\gamma\sin\beta\cos\alpha - \cos\gamma\sin\alpha \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix}$$

The transformation from world reference frame to camera reference frame based on the equations above is illustrated in Figure 2.1. Camera is assumed to be calibrated and the optical distortion is not taken into account. The detailed overview regarding camera calibration and detection error is given by Siltanen in [Sil12].

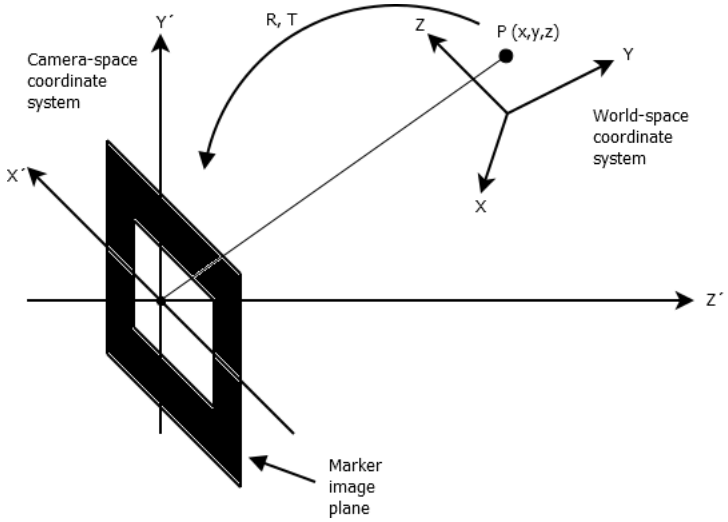


Figure 2.1: Transformation from world reference frame to camera reference frame adopted from [CFA+10].

Distance to Marker

After applying the transformation, the distance from the camera reference point to the detected object can be easily found using Euclidean distance between points p and q in the same coordinate system [Wik17a]:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \quad (2.3)$$

This will be later used for distance estimation between the user, wearing augmented reality device, and the detected marker in order to dynamically adapt the quality of the visualized data.

2.1.3 Object Detection

As highlighted above, Hololens is equipped with the depth camera that enables mapping the surfaces and save it in the form of spatial meshes. Unfortunately, as Garon et al. conducted in their research [GBD+16], the high resolution depth data is not available to the developers, thus making it impossible to use for object recognition. One of the solutions proposed in their paper is to use external precise depth camera and transfer the data in real-time to the Hololens via WiFi. However, this approach inevitably influences bandwidth, requires additional hardware, that decreases mobility and requires additional power source. Because if this limitation alternative techniques should be examined.

Generally speaking, there exist two types of visual tracking techniques for object detection: feature-based tracking and model-based tracking [CFA+10]. Model-based tracking requires some knowledge beforehand, that could be a 3D model reconstructed from the object to be detected, or some predefined representation of its distinguishable parts or elements. The main disadvantage of this technique is increased model size and large mismatch between the real object and its digital representation due to difference in amount of details, textures, etc. This makes it almost impossible to detect the object with high confidence, the only feasible approach is to perform the lines detection and matching or utilize the hybrid tracking with the help of sensors data [Sil12].

As an alternative, the feature-based ad-hoc tracking method could be used, that extracts and recognizes feature set from marker-based image. Furthermore, image detection can be used with the help of the RGB locatable camera directly available within the Hololens. In the reminder of this section background of marker recognition and differentiation is described in details.

2.1.4 Marker-based tracking

Markers can be used either for presence detection or carry additional information. The later are hard to recognize from large distances and the encoded information is not relevant in the context of presented approach, that concentrates on simple marker shapes that can be distinguished. Two common shapes used as a marker tag are square and circle. In case of a single circular marker tag, it is hard to argue about the pose, because it always leads to ambiguity, however multiple circular tags increase the pose accuracy and offer better location as described by Köhler et al. [KPS10]. In contrast, square-shaped tags have sufficient amount of four non-linear points and thus yield adequate pose estimation results even for a single marker.

Primitive shapes detection can result in a large amount of false-positives, thus in order to correctly identify the image to be a marker, special features should be recognized, that appear in the form of arbitrary images, digital codes or topological regions. Identification of these features can be based on correlation between the marker image points or regions and stored reference data. Another approach is reading the binary digits presented in the marker image as code bins, fourier encoded code words or TriCodes [KPS10].

Most markers come in the form of square fiducial images with contrasting, usually black, border and unique image in center. The border allows locating the marker in environment and image pattern serves as a unique identifier. The ideal fiducial should have the following properties: unambiguous position and orientation detection, orientation invariance, faultless pattern identification and fast recognition [OXM02].

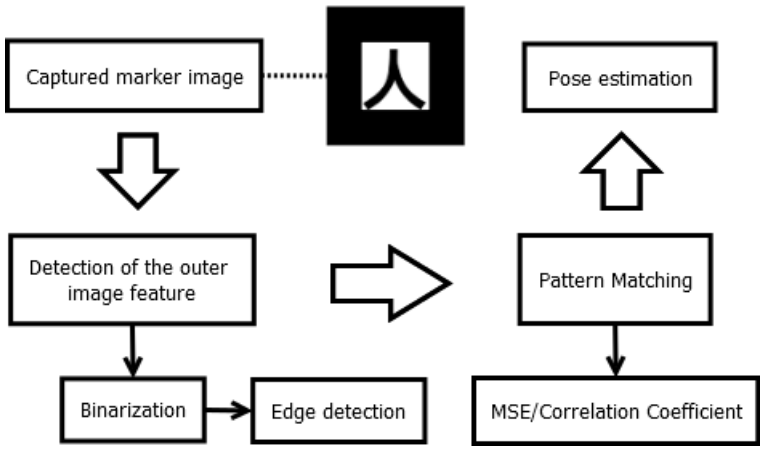


Figure 2.2: Marker detection, identifications and pose estimation flow adopted from [AM04].

The procedure of marker recognition is shown in Figure 2.2 and described in details below.

In order to identify the corners of the border, line fitting to the edges can be applied with the set of linear functions $f_i(x, y) = a_i x + b_i y$, which achieve the minimum values at the searched vertices. After corner identification the marker image should be normalized by finding the the homography matrix H that defines the correspondence between real scene $(x'_i, y'_i), i = 1, 2, 3, 4$, and the identified corners $(x_i, y_i), i = 1, 2, 3, 4$ [GWB16]:

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = H \begin{pmatrix} x'_i \\ y'_i \\ 1 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix}$$

Afterwards, the normalized image within the located region can be correlated with a marker images, stored in the database. The basic approach to compare the candidate image with the predefined pattern is to use mean squared error (MSE):

$$c(I, P) = \left(\sum_x \sum_y \Big(I(x, y) - P(x, y) \Big)^2 \right)^{1/2}$$

where $I(x, y)$ - candidate image, $P(x, y)$ - pattern image, $c(I, P)$ - dissimilarity(small values indicate similarity);
However, MSE is luminance invariant, so more advanced technique for image difference detection is determining the correlation coefficient [AM04; OXM02]:

$$\rho = \frac{\sum_x \sum_y (I(x, y) - \mu_I)(P(x, y) - \mu_P)}{\sigma_I \sigma_P}$$

where the mean of the candidate and pattern image is given by:

$$\mu_I = \frac{1}{xy} \sum_x \sum_y I(x, y)$$

$$\mu_P = \frac{1}{xy} \sum_x \sum_y P(x, y)$$

and standard deviations as:

$$\sigma_I = \left(\sum_x \sum_y (I(x, y) - \mu_I)^2 \right)^{1/2}$$

$$\sigma_P = \left(\sum_x \sum_y (P(x, y) - \mu_P)^2 \right)^{1/2}$$

Finally, the pose is estimated by finding the rotation matrix and translation vector following the slightly modified procedure described in Section 2.1.2. In order to transform from world coordinate system to marker image coordinate system the Equation (2.1) should be multiplied by camera intrinsic matrix K [AM04]:

$$K = \begin{bmatrix} \alpha_x f & 0 & u_0 \\ 0 & \alpha_y f & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where f - focal length of the camera, α_x - horizontal pixel size, α_y - vertical pixel size, (u_0, v_0) - projection of camera center (principal point);

2.2 Numerical Simulation

A great variety of physical, biological and chemical processes are described by means of partial differential equations (PDEs) and application of recent advances in computer resources and computational power facilitate more thorough analysis and evaluation of mathematical models used to describe different real-world systems. However, analytical form of PDEs can't be processed directly because of its continuous nature, it should be first discretized and appear as a linear system of algebraic equations easily processed by computer. In order to solve this problem a large amount of numerical methods exist, couple of which will be described in this section.

2.2.1 Finite Difference Method

Finite Difference Method is based on replacing the derivatives in PDE by difference approximations. As an example, the scaled boundary-value problem is taken, as proposed by Langtangen in [Lan03]:

$$-u''(x) = f(x), \quad 0 < x < 1, \tag{2.4}$$

$$u(0) = 0, \tag{2.5}$$

$$u'(1) = 1. \tag{2.6}$$

In order to apply the finite differences method, the domain $(0, 1)$ needs to be partitioned into $n-1$ intervals $[x_i, x_{i+1}], i = 1, \ldots, n-1$, with $x_1 = 0, x_n = 1$. Then, for each point in the grid (discretized domain) the approximate solution u_i is sought.The second order derivatives $u''(x_i)$ are substituted by central difference approximations:

$$u''(x_i) \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = \frac{u(x_i + h) - 2u(x_i) + u(x_i - h)}{h^2}, \tag{2.7}$$

where $h = x_{i+1} - x_i$ is the length of the interval and is assumed to be constant - $h = 1/(n-1)$, the error of this approach to the exact derivative is of order h^2. The $u_i'(x_i)$, in its turn, is approximated by the centered difference approximation:

$$u'(x_i) \approx \frac{u_{i+1} - u_{i-1}}{2h} = \frac{u(x_i + h) - u(x_i - h)}{2h}. \tag{2.8}$$

Another requirement is that the differential equation must be fulfilled for each grid point. Letting $u_1 = 0$ and $u_n' = 1$ be the boundary conditions, above transformations result in the following n algebraic equations for n unknowns $u_1, \ldots, u_n$ [Lan03]:

$$u_1 = 0, \tag{2.9}$$

$$u_{i+1} - 2u_i + u_{i-1} = -h^2 f(x_i), i = 2, \ldots, n-1, \tag{2.10}$$

$$2u_{n-1} - 2u_n = -2h - h^2 f(x_n). \tag{2.11}$$

This set of equations form a linear system that can be rewritten in matrix form $Au = b$, where A is the $n \times n$ matrix of coefficients and b is the right-hand side. After getting to this point, the system could easily be solved by numerous iterative methods(Jacobi, Gauss-Seidel, SOR, etc.) [Saa03] or simply by Gaussian elimination.

2.2.2 Finite Elements Method

The main idea of the finite elements method is to find an approximation to the unknown function $u(x)$ in the following form [Lan03]:

$$\hat{u} = \sum_{j=1}^{M} u_j N_j(x)$$

where $N_j(x)$ - basis function; u_j - unknown coefficients.

In this method the particular set of coefficients u_j is sought that produces the minimal norm of the error $||u - \hat{u}||$, that is called residual.

2.2.3 Finite Volumes Method

The finite volumes method as opposed to previous methods uses the integral form of the equation, that is specifically suitable for conservation laws [LeV02]:

$$\frac{d}{dt}\int_{x_1}^{x_2} q(x,t)dx = f(q(x_1,t)) - f(q(x_2,t))$$

where q defines density and f - the flux.
The above equation shows that the change of some conserved quantity between any two points x_1 and x_2 is regulated by the flow at the endpoints.

Finite volumes method solves the problem of discontinuities, where the differential equations break by dividing the domain into grid cells and calculating the averaged approximate integral over each of them. In each iteration the values are adapted according to the flow through the edges of the grid cell that is possible by finding the right numerical flux function to correctly approximate the flow [LeV02].

2.2.4 Advection–Diffusion Equation

As an example of partial differential equation for numerical simulation, the advection-diffusion equation is used, that describes the transport of some material or energy with the constant velocity (e.g heat distribution on the surface with the constant airflow). The general form of the equation for 1D case is given by:

$$\frac{du}{dt} = \beta\frac{d^2u}{dx^2} - a_x\frac{du}{dx} + f$$

where u - the sought quantity, β - diffusion coefficient, a_x - averaged velocity and f is the function describing the flux of the matter u.

The equation above belongs to the general class of parabolic equations. However, assuming the steady state of the system $\frac{du}{dt} = 0$ the advection-diffusion equation is transformed into the stationary form, that belongs to elliptic class of equations [KT17]:

$$-\beta\frac{d^2u}{dx^2} + a_x\frac{du}{dx} = f, \quad 0 < x < 1, \tag{2.12}$$

$$u(0) = 0, \tag{2.13}$$

$$u(1) = 1. \tag{2.14}$$

One can definitely see the correspondence with Equation (2.4). So applying the same reasoning as in Section 2.2.1, by substituting the $\frac{d^2u}{dx^2}$ with the central difference approximation from Equation (2.7) and $\frac{du}{dx}$ with the centered difference approximation from Equation (2.8) in Equation (2.12) one can get:

$$-\beta\frac{u_{i+1}-2u_i+u_{i-1}}{h^2}+a_x\frac{u_{i+1}-u_{i-1}}{2h}=f, \quad on\,[0,1], \tag{2.15}$$

$$u_0=0, \tag{2.16}$$

$$u_{n+1}=0. \tag{2.17}$$

that can be transformed to:

$$au_{i+1}+bu_i+cu_{i-1}=f(x_i) \tag{2.18}$$

where $a=\frac{a_x}{2h}-\frac{\beta}{h^2}$, $b=\frac{2\beta}{h^2}$, $c=-\frac{a_x}{2h}-\frac{\beta}{h^2}$.

The equation Equation (2.18) represents the linear system of the from $Au=f$, where $u=[u_1,u_2,\ldots,u_n]^T$ is the vector of the unknown values, $f=[f(x_1),f(x_2),\ldots,f(x_n)]^T$ is the right-hand side vector and A is the matrix of coefficients, that combines the a,b,c fractions defined above.

This reasoning can be easily extended to 2D case as proposed by [DSD+17] for the equation in the form:

$$-\beta\nabla^2u+a_x\frac{\partial u}{\partial x}+a_y\frac{\partial u}{\partial y}=f,$$

where a_x and a_y are the advection coefficients in x and y directions accordingly.

2.3 Reduced Basis Method

Discretization techniques described in Section 2.2 lead to the numerical models with high dimensions, that imposes high hardware demands and results in high computation times. Often the exact solution is not required and the result should be available as soon as possible, that is the case e.g. in real-time applications or when multiple results are requested depending on the input parameters change, the solution should be computed on the device with constrained resource, etc. Thus the method should exist that constructs the low-dimensional subspace from high-dimensional full numerical problem, such that $X_N \subset X$. One of such methods is the Reduced Basis Method (RBM), that reduces the model to low-dimensional approximate solution $u(\mu)$, where μ is the parameter vector, allowing fast computation even with the limited set of resources [Haa16].

2.3.1 Subspace construction

The subspace x_N is usually constructed by means of snapshots, that is the precalculated solutions for some defined parameters $\mu_i, i = 1, \ldots, N$. Those parameters act as a training set, that's why it is important for low approximation errors to choose suitable parameter set, depending on the operational conditions. When the subspace is available, the approximate solution $u_N(\mu) \in X_N$ is obtained from the reduced model, constructed e.g. by Galerkin projection. The real solution can then be obtained as: $u(\mu) \approx V u_N(\mu)$, where V is the snapshot projection matrix [Haa16]. The above description is represented visually in Figure 2.3.

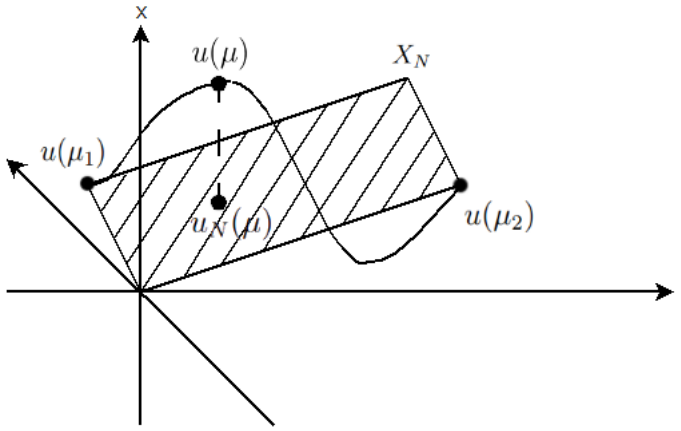


Figure 2.3: RB-approximation adopted from [Haa16].

2.3.2 Parameter-separability

Another important property of the RBM is the affine parameter dependance, or parameter-separability [Haa16]:

$$A(\mu) = \sum_{q=1}^{Q_A} \theta_q^A(\mu) A_q, \quad Q_A \in N,$$

where θ_q^A - is the coefficient function and A, A_q - matrices of the same shape. It gives the possibility to solve the linear systems of the form $Ax = b$ by precomputing the separable matrices in advance, changing only the coefficient function $\theta(\mu)$ for required parameter set.

2.3.3 Offline/Online Decomposition

The whole computation is then subdivided into two distinct parts: the offline phase and online phase. Offline phase is used for generating the reduced model with the required number of snapshots and usually requires high processing power, because of its intensive computation procedure. During the online phase the approximate solution for arbitrary parameter μ can be easily acquired, making it suitable for fast query answering on resource constrained hardware. Moreover, the computation complexity for the online phase depends only on the dimension of the reduced subspace X_N for any full problem space X [HO11].

2.3.4 Error estimation

It is also important to estimate the error of the approximate solution to the exact one: $e(\mu) = u(\mu) - u_N(\mu)$. In RBM the quality metric is given by the residual, that for the system of linear equations $A(\mu)u(\mu) = f(\mu)$, using the approximate solution $u_N(\mu)$ defined above, is given by: $r = A(\mu)Vu_N(\mu) - f(\mu)$. The norm of the residual again consists of separable matrices, that can be precomputed for fast error estimation [DSD+17; VPRP03].

2.4 Code Offloading

Despite the growth in the computing power of mobile devices, most of them still lack the ability to perform CPU-intensive operations efficiently, because of rapid battery drain, energy consumption and heat dissipation. So the most complex operations should be offloaded to the server and executed on the remote infrastructure either partially or fully. There are mainly two distinct wireless technologies for client-server communication, namely IEEE 802.11 (WiFi) and 3/4G cellular networks. However, as discovered by the Cuervo et al. in [CBC+10] 3/4G technologies suffer large round trip time (RTT), that appears to be the significant factor in energy consumption, so the proposed solution is to utilize the nearby server infrastructure in the same network as the mobile device, thus minimizing RTT and saving battery life.

The run-time component presented in [CBC+10] mainly consists of solver, that is responsible for deciding if method should be executed remotely or locally based on the data received from profiler, that constantly monitors the network connectivity, power consumption, program characteristics and instruments the execution accordingly; and finally the client and server proxies, that regulate control flow and transfer of the data

according to solvers decision. These components are briefly shown in Figure 2.4. The decision on which operations should be offloaded to the server is made by the application programmer by means of annotating the code parts with *remote* meta-attributes, utilizing the reflection API.

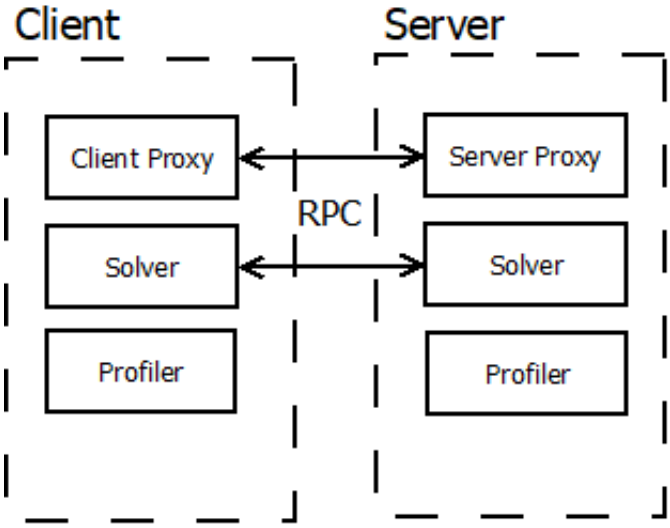


Figure 2.4: Code offloading architecture adopted from [CBC+10].

2.4.1 Safe-points

Important part in code offloading is also the data serialization and incremental data transfer between client and server for minimizing the bandwidth and exchange of only the relevant information instead of complete state of the system. This is particularly useful in case of network failures, when the previous parts of the state are stored on the client, and the communication channel is able to recover after link restore using the preemptive code offloading as proposed in [BDR14]. In their paper Berg et al. use the so called safe-points, that represent the intermediate states or snapshots of the system, sent dynamically with adaptive scheduling algorithm. The execution time of the code blocks are predicted using the history-based approach [CBC+10], that maps input parameters to the previously stored execution times. And the connectivity is predicted using the time-continuous Markov chain, that is built mainly from the latency and bandwidth parameters.

Two approaches described above have a serious drawback, that is the need to share and synchronize the state between client and server, because the code is splitted between two parties. Often, it is not feasible to make such split, whereas mobile client and server should process the independent set of operations. Thus, presented work makes a clear

separation of concerns, completely offloading process intensive tasks to the server and communicating back the reduced model, that can be efficiently utilized by the mobile device independently. It saves the bandwidth overhead of continuous state transfer and code synchronization by making fewer requests to the server that allows the mobile device to be more autonomous.

2.4.2 Server cache

Another approach to minimize the communication latency for code-offloading technique is to use the remote-side hash as proposed in [BDR15]. By utilizing the local cache on the server the data access time is drastically improved and in the case of cache hit the mobile device can acquire the result almost immediately, where the bandwidth and latency of the network is the only limiting factor. The cache is stored remotely in the form of huge hash map that maps the incoming unique keys from the request with the available data on the server. On the other hand, it takes substantially larger time to first execute the code remotely and then send it to the client in case of cache miss. Often, it is better to execute the code locally on the mobile device in case of unstable connection, low network quality or when the local execution would be faster compared to offloading processing to the server without precomputed cache. The above technique is utilized in presented method to retrieve the precomputed reduced models from server cache for fast data transfer, thus minimizing the communication latency.

2.5 Distributed Reduced Basis Method

As described in previous section, despite offloading compute intensive tasks to the server the state of the whole application should be constantly monitored and synchronized, that imposes bandwidth overhead, requires storing additional safe-points and limits the autonomous usage of the mobile device.

As a solution Dibak et al. in their paper [DSD+17] propose to use the RBM (see Section 2.3) approach to precompute the reduced model for the set of given parameters, so called training set, and quality constraints, such as discretization factor D and maximal residual r_{max} on the server in order to later perform fast approximate computations for given simulation parameter set μ on mobile device. Even on the powerful server infrastructure computation of the reduced basis takes reasonable amount of time, typically exceeding the pure offloading of the full problem solution to the server, however, when the reduced model is formed and transferred to the mobile device it is capable of

providing the acceptable solution for the range of input parameters with low latency for better user experience.

2.5.1 Basic approach

The above paper presents couple of approaches for utilizing RBM in efficient manner. The most straight forward is the basic approach, when the mobile device is mainly responsible for efficiently answering multiple user queries for the changed simulation parameters, coming from the sensor readings, using the reduced basis, requested from the server. However, it has couple of serious drawbacks, namely the acceptable error of the approximate solution can only be guaranteed for the parameters that fit into the range of reduced model's training set; the training set is not adapted depending on the operational conditions; it makes no estimation on the communication link quality and time needed to load the reduced basis from server to mobile device.

2.5.2 Adaptive approach

One of the problems is solved by the Adaptive approach, when along with answering the user queries the error of the approximate solution is estimated and if it exceeds the r_{max} the new exact solution for parameter μ is requested from the server. The server then updates the reduced model with the new snapshot and newly calculated separable matrices and sends the result back to client, thus eliminating the error for the specific parameter μ. The problem with this approach arises when the physical system under consideration rapidly changes the range of parameters. In this case, the upper bound of approximate solution error, identified by r_{max}, can't be guaranteed for all the parameters in the updated range.

The system, developed in this work, will utilize the approaches presented in [DSD+17], while solving the above mentioned problems and limitations, and extending their applicability to the set of numerical simulations, identified from the real-world environment and operational conditions of the system.

3 System Model and Problem Statement

In this chapter, the system model for numerical simulations on augmented reality devices will be described, followed by problems arising within the design of such system.

3.1 System Model

Figure 3.1 gives the overview of the system components, representing two physical components: mobile device and server; as well as three software components: numerical simulation, middleware and user application.

The simulation middleware runs on both mobile device and server, orchestrating their communication. The server is responsible for complex numerical computations of the numerical problem and mobile application uses the simulation results received from the server to present additional data to the user. As the communication link between the server and mobile device IEEE 802.11 (WiFi) technology is used. Below each component will be described in details.

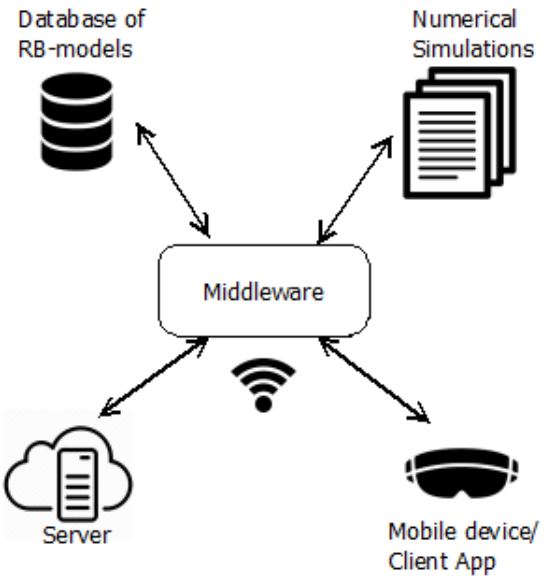


Figure 3.1: System components and their interconnections.

3.1.1 Mobile device

The mobile device appears in the form of augmented reality glasses, that collects visual and sensor data of the surrounding, forming the raw sensor data list $p = \{p_i\}, i \in N$. Real-world objects are represented by markers with unique feature images, defined as tags. While the user of the system changes location, mobile device performs the dynamic mapping between the tags $M = \{m_1, \ldots, m_n\}$ and corresponding numerical simulations, see Figure 3.2. Mobile device is connected with the middleware through the client app, sending all the acquired data for further processing, previously mapping the sensor values p to simulation parameters $\mu = \{\mu_j\}, j \in N$ through the corresponding mapping function: $g(p) : p \mapsto \mu$. Additionally, after the middleware has verified the connection status, mobile device can decide on the future execution strategy in case of network failure. When the reduced solution for the numerical simulation under consideration is received from the middleware, mobile device first stores the reduced bases R_{mob} in internal storage, identified by the available space s_{avail}, for fast computation of approximate simulation result. Finally, the identified real-world object is augmented with the solution of numerical problem in the form of virtual data.

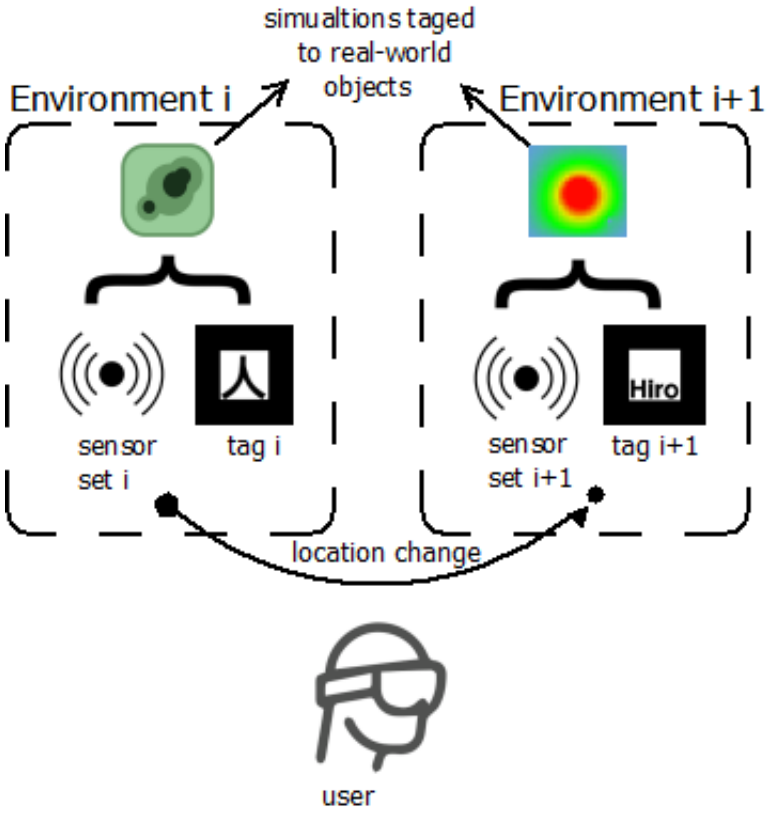


Figure 3.2: Interaction with different systems through mobility.

3.1.2 Server

The server is often situated on the remote data center or cloud infrastructure. It allows the use of the most efficient hardware, specialized for fast numerical computations or the processing resources can be scaled-up horizontally by adding additional computing nodes. From the above reasoning, it is especially suitable for solving numerical problems. Almost unrestricted storage and processing capabilities make it possible to store and process multiple simulations for the large range of input parameters set. The particular simulation can then be chosen depending on the request context. Server is then responsible for solving the numerical problem for exact input parameters received from the middleware and storing the result in cache for further requests. If S is the set of all simulations and $\forall S : \exists\{R_s\}$, where $R_s = \{r_m\}, m \in N$ - the set of all possible reduced bases for the particular simulation, then all precalculated reduced models $R_{pre} \in R_s$, available on server, are stored in RB-models database.

3.1.3 Numerical Simulation

Numerical simulations can span the wide range of physical models and are generally implemented by the simulation expert. Despite the ability to solve the problem with some generic algorithm, the expert usually selects the most suitable solver method for specific simulation, for example using the RBM as discussed in Section 2.3. In either case, the quality specifications of the system should be defined. They can be regulated by some quality metrics, as the discretization parameter of the full problem D, maximum error of the simulation results in the form of residual r_{max} or the training set $T = \{\mu_1, \ldots, \mu_n\}, n \in N$ that determines the number of snapshots and the applicability range of the reduced model. Through the middleware, each simulation is mapped to the corresponding real-world tag and the reduced model database (RB-database), that stores the reduced bases for the set of input parameters and quality constraints defined above.

3.1.4 Middleware

The middleware acts as the distributed controller, connecting the server and mobile device, while executing the code on both components. It is responsible for preprocessing the sensor data, received from the user application, mapping the recognized real-world tags to the corresponding numerical simulations, monitoring the status of communication link and sending the result back to the user application for further interpretation.

3.1.5 User Application

User application encodes all of the environmental data as well as the simulation tag and quality requirements and sends the query to the midlleware. It sends the queries periodically in correspondence with the sensor values p and changed quality demands, identified by the visual perception parameters v. It also decides on the execution strategy (should the simulation be performed directly on the mobile device, by solving the full numerical problem as discussed in Section 2.2.4, or the reduced model should be requested from server by finding the required reduced basis from RB-models database), depending on the connection status, received from the middleware. User application either solves numerical problem directly on mobile device or answers user queries locally with low latency, after receiving the reduced basis from the middleware. Regardless of the selected strategy, simulation results are dynamically presented to the user. For the identified problem and current set of parameters only corresponding reduced model is requested, thus saving available storage s_{avail}, energy and increasing responsiveness, that can be identified as the time t_s from sending the query to presenting the simulation results to the user.

3.2 Problem Statement

The results of numerical simulations should be available ubiquitously for the user demand in place (e.g. in physical lab, production lines, manufacturing, etc.). The main challenge is to identify the particular simulation and have the reduced model available on the mobile device, without the need to preload it in advance. Mobility also comes into play, whereas specific objects and physical places should be dynamically mapped to the corresponding model while the user changes its location and examines particular system. The foregoing problems are also complemented by the limitations of the mobile device: limited storage, low computational resources, constraints on power consumption. In addition, the specialist in the field should be able to interactively change the input data set and perceive the results of the simulation in adequate time. As a consequence the low latency should be achieved for calculating the simulation results. Figure 3.3 depicts the general overview of the problem.

Optimization Problem

The problem is to find the optimal way of solving the numerical problem, either by dynamically selecting the reduced simulation model available on the mobile device, requesting the reduced solution from server or performing the calculation of the full problem directly on the mobile device in order to find the best simulation time t_s for the given set of parameters. The selection of specific solution depends on the following

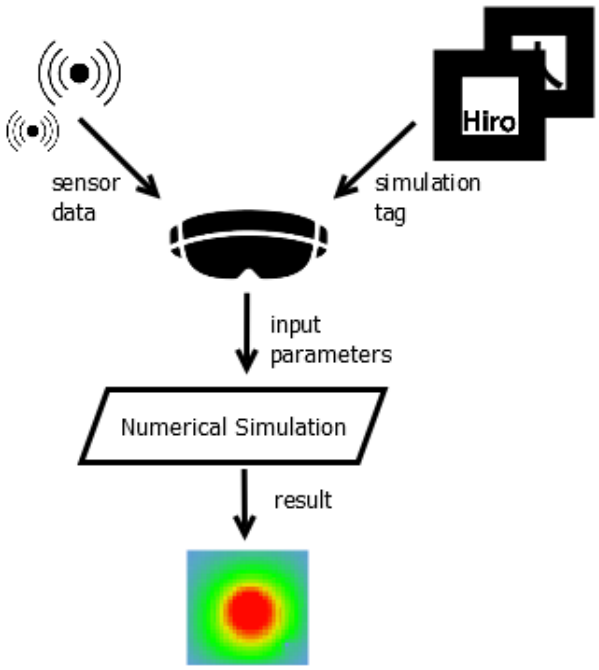


Figure 3.3: High level overview of the problem.

constraints: simulation quality, available storage on the mobile device, bandwidth of communication link and quality of user experience (QoE) - see Figure 3.4.

Simulation quality

The quality of numerical simulation result can be regulated by means of the discretization parameter D of the full problem, maximum error of the approximate solution given by residual r_{max} and the training set $T = \{\mu\}$, where μ is the set of particular simulation parameters. The problem now is to dynamically map those simulation parameters to the quality metrics of user experience (QoE), that come from the sensor readings p and visual perception parameters v. Then for the approximate solution $u_N(\mu)$, the problem of selecting the training set for reduced model is define as:

$$\forall p : \exists T\{\mu\}, \text{ s.t. } error(u_N(\mu)) \leq r_{max}$$

In such a way, the optimal function should be found for precalculating the training set, depending on the given parameters, in order to minimize the approximation error with the upper bound of maximal residual.

Additionally, there should exist the mapping function between user perception parameters v and full problem discretization D: $map(v) : v \mapsto D$, such that the quality of simulation result is adapted in accordance with user experience demands.

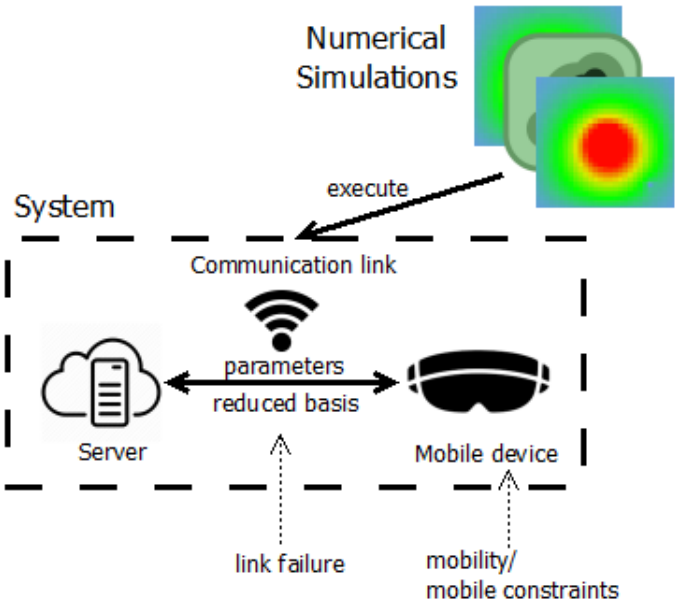


Figure 3.4: Reduced model selection and limitations.

Storage

It is not possible to preload all possible reduced bases for all simulations and parameter sets to mobile device. That is, only the relevant set of reduced simulation models need to be loaded to the internal storage of mobile device, keeping their total size below the available space s_{avail}:

$$\sum_{r \in R_{mob}} s(r) < s_{avail},$$

where $s(r)$ - the space taken by one reduced model, R_{mob} - set of all the reduced bases, available on the mobile device.

Bandwidth

The user expects to perceive the results of the simulation with minimum latency and adaptive quality. In the case of connection loss the time for simulation calculation t_s should be finite, that is $t_s < \infty$, otherwise it would contradict the quality constraints. So there should exist a method to identify the network failure and still allow to dynamically follow the parameter change and present valuable data to the expert, possibly with the coarse result. Let the function $a(t) \to \{0, 1\}$ represent the network availability at time t. Then the problem of permanent communication link failure can be formulated as follows:

$$a(t) = 0 \implies t_s \mapsto m(t) \text{ s.t. } t_s < \infty$$

That is, if the network is not available at time t, when the simulation update is requested, the aim is to make the simulation calculation time t_s finite by switching to the alternative computation method $m(t)$.

Server cache

For the particular set of simulation quality parameters, namely training set T, discretization factor D and maximal residual r_{max}, it should be identified if the corresponding reduced basis $r(D, T, r_{max})$ is already precalculated on server by checking if it is stored in RB-models database: $r \in R_{pre}$, where R_{pre} represents the server cache as defined previously. Solving this problem will allow to optimize the time of reduced basis delivery in case of server cache miss scenario.

4 Methodology

This chapter describes the approach of selecting the simulation depending on the recognized real-world objects, loading only relevant reduced model for the RBM, depending on simulation quality parameters to overcome mobile storage constraints, and providing the user with additional data, in the form of simulation results, to better understand and interact with the system of interest.

As the input data for the approach, the identifier for the real-object is expected, that is defined as a tag. It is assumed, that for every tag there exists a mapping to particular simulation, that will determine the simulation parameter set and quality metrics. Another input is generated from the environmental sensor values that determine the range of simulation parameters and allow dynamic observation of the process.

4.1 Approach overview

All the steps required to present the simulation results to the user are illustrated in Figure 4.1, and described below:

Simulation tag mapping
: While the user changes its location the application constantly monitors the environment, analyzing the video stream captured by the integrated camera of mobile device, in order to find predefined image patterns in the form of fiducial markers (see Section 2.1.4). When the marker is recognized, its pose and world-space coordinates are estimated as described in Section 2.1.2. Then the particular tag is identified and mapped to the corresponding simulation, adapting its quality depending on the distance to the user.

Training data collection
: The environmental sensor values are periodically read to understand the context of the system. Typical values of sensor readings are collected in order to form the training data set T for reduced model of the simulation. It is also adapted for dynamic change of the operational range, such that the approximate solution error is minimized. The current sensor readings are mapped to the simulation

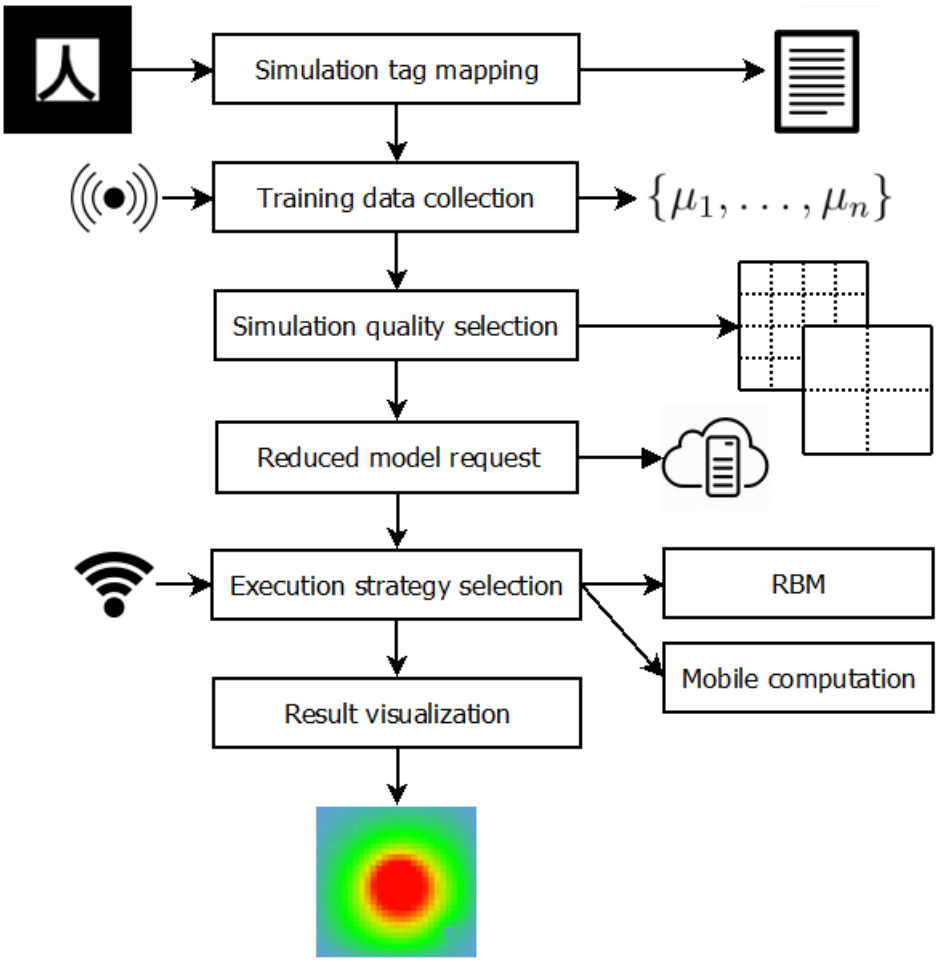


Figure 4.1: Overview of the individual steps of the approach.

parameters μ and act as the input for user queries to continuously track the system behavior.

Simulation quality selection

Depending on the user experience parameter, or the visual perception parameter v, that is defined as the distance d to the marker, the discretization parameter D of the identified simulation is adapted to provide the expert with the adequate amount of information that can be perceived. Here, the maximum allowed error of the approximate solution is also defined, determined by maximal residual r_{max} (see Section 2.3.4).

Reduced model request

After identifying the simulation quality parameters, consisting of the training set, discretization factor and maximal residual defined above, the corresponding reduced basis is requested from server. But depending on the server cache, there

exist two distinct cases: the requested reduced model is already precalculated on the server or it should be constructed from scratch by solving the set of full numerical problems for the given training set. So the request to identify the basis availability is sent in advance to minimize the latency for future quality updates.

Execution strategy selection

When the simulation itself and all of its parameters are known, the connection quality is verified in order to decide on the strategy of choosing the computation method. When the communication link suffers permanent or partial failure, the approach first checks if the reduced basis for the current set of parameters is already available on the mobile device, so it can be reused, utilizing the RBM technique. If it is not the case, system switches to computation of the full problem on the mobile device, lowering the simulation quality if needed. For the normal operational conditions, when connection is not broken, approach uses the reduced basis with maximal available quality, while waiting for the updated reduced model from server.

Result visualization

After solving the numerical simulation problem, depending on the strategy defined above, the results are presented to the user in the form of the hologram overlaid on the real-world object, using the pose and distance estimation, performed in the first step, with the resolution that corresponds to the previously identified discretization parameter D.

4.2 Simulation tag mapping

In this step, mappping between real-world objects and corresponding simulations is done by using tags, identified by the predefined image patterns (e.g. the interior image of the fiducial marker). Surely, the selection of target object is not limited to the specific choice of marker patterns, generally, any technique of object recognition and image detection can be used, that can be recognized by the computer vision algorithms. This decision is based on the reasoning, described in Section 2.1.3.

Pose estimation and correction

Utilizing the continuous video stream from the RGB integrated camera, available on the mobile device, each frame is analyzed with the help of computer vision algorithm to recognize the exterior shape of the marker and get its rotation matrix R, presented in the form of quaternion, and translation vector T. The identified position and rotation usually deviate from the real-world values due to improper camera calibration (see Figure 4.2). In order to eliminate the produced offset, the affine transformation is

applied, as proposed by Qian et al. in [QAKN17], where this method is identified to produce the minimum reprojection error. So if the original normalized coordinates of the marker points identified by the tracking system are given by $q = \{q_1, \ldots, q_n\}$, then new coordinates after applying the transformation $p = \{p_1, \ldots, p_n\}$ can be obtained as:

$$p_i = T_A \times q_i, T_A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where the parameters a_{ij} are identified manually, achieving the best perceptible alignment.

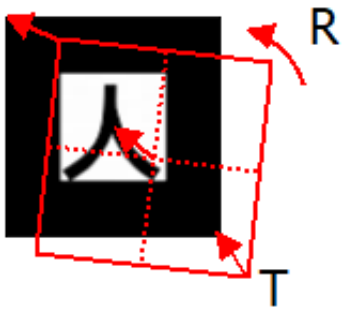


Figure 4.2: Mismatch between the identified and real-world position and rotation.

Mapping simulation to real-world tag

After the object is identified to be a marker, the specific interior image is correlated with the predefined patterns in order to uniquely identify the real-world tag (see the details in Section 2.1.3). So, for the current location, the set of parameters $(location, tag)$ is passed to server for mapping the identified tag with the corresponding simulation by the following procedure: let the list of all the predefined simulations be a vector $S = \{s_1, \ldots, s_n\}$, where $n \in N$ - the amount of unique simulations, and $M = \{m_1, \ldots, m_n\}$ - the list of the tags for mapping the real-world objects to corresponding simulation model, identified by the marker feature. Then for a given tag m_i and current user location l the mapping function for the corresponding simulation s_i is generally defined as:

$$m_i \mapsto s_i(l), 1 < i < n, n \in N.$$

In presented approach, each marker tag has the unique identifier, that acts as a key for one-to-one mapping to the particular simulation through the hashmap data structure, though mapping can as well be done through the database engine or other techniques. The specific marker object, that is constructed for the identified marker tag, then stores the corresponding simulation as its property for future user queries.

Algorithm 4.1 Marker detection and simulation mapping

```
1: procedure MARKERTRACKING(preTags, detectedMarkers, C)
in: preTags represents the set of predefined marker tags,
    detectedMarkers represents the set of all recognized markers,
    C represents the camera world-coordinates vector
2:     loop
3:         feature ← analyse video frame and extract feature image
4:         for all preTag ∈ preTags do
5:             if correlation(feature, preTag) → 1 then
6:                 detected ← True
7:             end if
8:         end for
9:         if detected then
10:            id ← tag identifier from preTags
11:            (R, T) ← identified rotation matrix and translation vector
12:            correct(R, T)          // apply the affine transformation for offset correction
13:            d ← distance(T, C)          // determine the distance to the user
14:            if id ∈ detectedMarkers then
15:                detectedMarkers(id) ← (R, T, d)
16:            else
17:                simulation ← sendTagToServer(id)      // get simulation for identified tag
18:                marker ← (simulation, R, T, d)          // construct the marker object
19:                add marker to detectedMarkers
20:            end if
21:        end if
22:    end loop
23: end procedure
```

Distance estimation

Another value of interest is the distance d to the marker. This value can be obtained after identification of markers world-space coordinates $w_m = (x', y', z')$, assuming that camera world-coordinates $w_c = (x, y, z)$ are known, by the Euclidean distance between two point in 3D space as mentioned in Section 2.1.2:

$$d(w_c, w_m) = \sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2}.$$

The described steps are executed continuously, allowing to dynamically track the pose of the marker and the distance to the user. It also allows to either simulate different processes for multiple markers simultaneously or switch between the simulations, observing one tag at a time. The general procedure for markers detection and simulation tag mapping is outlined in Algorithm 4.1.

4.3 Training data collection

Mobile device constantly monitors sensors, broadcasting environmental data for the current location. The raw sensor values can act as the input data for the simulation directly, or they can be previously mapped to the specific simulation parameters $\mu = \{\mu_1, \ldots, \mu_n\}$ (e.g. for the advection-diffusion problem this parameters are $(\mu_{diff}, \mu_{advx}, \mu_{advy})$, namely the diffusion coefficient, advection in x and y direction). In order to simulate the system behaviour for different operational conditions, the distribution of the simulation parameters is constructed, using random set of values in the predefined range $[\mu_{min}, \mu_{max}]$.

Algorithm 4.2 Training set construction

1: **function** TRAININGSET($\mu_1^{range}, \ldots, \mu_n^{range}$)
in: μ^{range} represents the range $[\mu_{min}, \mu_{max}]$ of distinct simulation parameter
out: $trainingSet$ represents all possible combinations of given parameters

2: $\mu_1 \leftarrow \min(\mu_1^{range})$
3: **while** $\mu_1 \leq \max(\mu_1^{range})$ **do**
4: $\mu_2 \leftarrow \min(\mu_2^{range})$
5: **while** $\mu_2 \leq \max(\mu_2^{range})$ **do**
6: ...
7: $\mu_n \leftarrow \min(\mu_n^{range})$
8: **while** $\mu_n \leq \max(\mu_n^{range})$ **do**
9: add $(\mu_1, \ldots, \mu_n)$ to $trainingSet$
10: $\mu_n \leftarrow \mu_n + \text{step}(\mu_n^{range})$
11: **end while**
12: ...
13: $\mu_2 \leftarrow \mu_2 + \text{step}(\mu_2^{range})$
14: **end while**
15: $\mu_1 \leftarrow \mu_1 + \text{step}(\mu_1^{range})$
16: **end while**
17: **return** $trainingSet$
18: **end function**

Training set construction

Based on the collected sensor data, the training set $T = \{\mu_n\}, 1 < n < M$ is constructed, where M is the amount of distinct parameters, that defines the applicability range of simulation result approximation, not exceeding the error threshold of maximum residual r_{max}. The training set is generated on basis of statistical parameter values distribution. However, for current approach it is sufficient to simulate this distribution

by the predefined interval of parameters in the range described in previous paragraph, changing the step width for more coarse or fine-grained set. This approach gives us the possibility to model the behavior of the system in case of rapid operation conditions change and dynamically react by adapting the training set interval accordingly. The construction of the training set is shown in Algorithm 4.2.

4.4 Simulation quality selection

The quality of the simulation is determined by the discretization parameter D, that defines the reduced space dimension. Evidently, the quantity of the information and simulation result perception quality grow with respect to this parameter, but computation intensity and the size of the reduced model grow as well for increased problem dimensions. Thus the approach is developed to optimize the user experience, while at the same time minimizing the latency of presenting the result to the user.

Discretization parameter selection

First, the discretization of the full problem should be found, such that result appears in adequate form for user perception. As the user perception metric the distance d to the identified marker is selected. Then the function is constructed, such as $m(d) : d \mapsto D$. It is defined as the mapping function between the discrete distance intervals $di = (di_1, \ldots, di_n)$ and the corresponding simulation sizes $D(D_1, \ldots, D_n)$, where n is the amount of discrete parameter values (see Figure 4.3):

$$\forall d \in \mathbb{R} | d > 0 : \exists D_j \text{ s.t. } d \in di_j \implies m(d) \mapsto D_j.$$

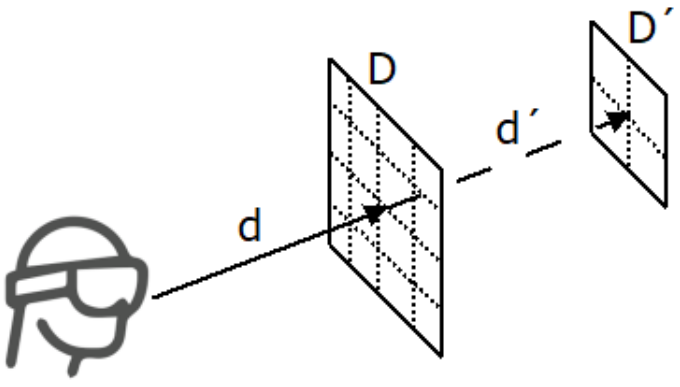


Figure 4.3: Full problem discretization parameter selection depending on the distance to marker.

Simulation quality change

The maximal error of the numerical solution approximation is defined by the maximum residual r_{max} to present the result that closely follows the exact system behavior and is set by the domain expert. Next, it is being checked if the new discretization parameter D_j changes the simulation quality requirements. In such a case, system tries to find the corresponding reduced model $r(D_j, T, r_{max})$, determining the training set T for the current parameters range $[\mu^j_{min}, \mu^j_{max}]$, as defined previously in Section 4.3.

At first, the search is conducted among all preloaded bases in the cache of the mobile device R_{mob}. Additionally, the connection to the server is monitored in order to detect network failure, verifying whether the response time from server t_r exceeds the maximum allowed timeout t^r_{max}.

The discretization value for the current simulation is updated either if the basis is available on the mobile device or the connection with the server is broken. It is then the work of the execution controller to decide on the solution strategy, that will be described in following section.

All of the steps described above are depicted in Algorithm 4.3. It is also important to notice, that this procedure is applied for all the detected markers simultaneously and independently, so multiple simulations can be observed in parallel with different set of quality parameters.

Algorithm 4.3 Simualtion quality determination

1: **procedure** QUALITYCONTROLLER(D_c, d_m, r_{max})
in: D_c represents current simulation discretization parameter value,
d_m represents distance to the current marker
r_{max} represents maximal residual

2: $D_j \leftarrow$ m(d_m) // map the distance to the simulation size
3: $T \leftarrow$ TrainingSet($[\mu^j_{min}, \mu^j_{max}]$)
4: $connection \leftarrow \begin{cases} \textbf{True}, & \text{network is available: } t^r < t^r_{max} \\ \textbf{False}, & \text{network is not available: } t^r > t^r_{max} \end{cases}$
5: **if** $D_c \neq D_j$ **then**
6: **if** $(r(D_j, T, r_{max}) \in R_{mob}) \vee (\text{not } connection)$ **then**
7: $D_c \leftarrow D_j$
8: **else**
9: RequestBasis(D_j, T, r_{max})
10: **end if**
11: **end if**
12: **end procedure**

4.5 Reduced model request

When all of the simulation quality parameters are known, the updated basis needs to be requested from server if it is not available in the cache of the mobile device R_{mob}. All of the parameters are firstly serialized in order to efficiently utilize the bandwidth and sent within the basis request. Depending on the server cache, the request time can vary significantly as discussed in Section 2.4.2. For the initial request with low discretization parameter D_1 the time to calculate and load the reduced model is small, but for more fine-grained results, i.e. higher values of D, calculation of the full problem on the server takes substantial amount of time. Presented approach allows to identify basis availability and preload the reduced models in case of server cache miss scenario $r \notin R_{pre}$.

Basis availability request

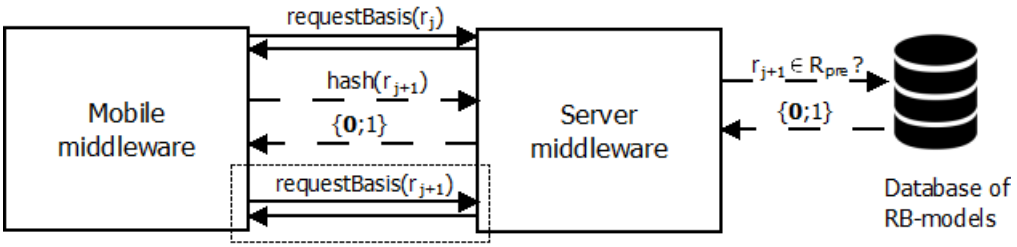


Figure 4.4: Request flow for basis availability verification and preloading for server cache miss scenario.

While requesting the basis r_j for the updated simulation size D_j, server availability of the precalculated reduced model for the next discretization parameter D_{j+1} is also verified. The function $h : (r) \mapsto hash$ is defined to calculatet the effective hash, uniquely identifying the reduced basis r. This hash is then sent to server, that tries to find the corresponding entry in RB-models database and returns one of two states: $\{0, 1\}$, either the basis is available on server or not (see Figure 4.4). For the reduced model r_{j+1}, that corresponds to the discretization parameter D_{j+1} the availability is defined as:

$$\begin{cases} 1, & \forall r_s \in R_{pre} : \exists r_s \text{ s.t. } h(r_s) = h(r_{j+1}) \\ 0, & \text{otherwise} \end{cases}$$

where R_{pre} is the set of all the precalculated basis stored in server cache.

If the basis is available on the server (cache hit), it is immediately returned and the latency is only determined by the communication link. On the other hand, if the following basis is not precalculated on the server, the reduced model should be first computed for the increased discretization factor D_{j+1}, that can increase the time for

presenting result to the user significantly. In order to minimize the latency for future user queries, while requesting basis for current simulation size D_j, another request is made to preload the basis for D_{j+1} in advance. Thus for server cache miss scenario the time for future simulation quality updates is decreased and for cache hit scenario, the internal storage space of mobile device is saved by loading the updated basis only when required. The code flow for requesting the updated basis from server is shown in Algorithm 4.4.

Algorithm 4.4 Requesting basis from server

```
1: procedure REQUESTBASIS(D_j, T, r_max)
in: D_j represents updated simulation discretization parameter value,
    T represents the calculated training set
    r_max represents maximal residual

2:     p_j^ser ← serialize(D_j, T, r_max)          // serialize simulation quality parameters
3:     r_j ← sendRequestToServer(p_j^ser)
4:     add r_j to R_mob                            // save received basis to mobile device cache
5:     hash_{j+1} ← h(D_{j+1}, T, r_max)           // compute hash for next simulation size D_{j+1}
6:     basisAvailable ← requestAvailableOnServer(hash_{j+1})
7:     if (not basisAvailable) then
8:         p_{j+1}^ser ← serialize(D_{j+1}, T, r_max)
9:         r_{j+1} ← sendRequestToServer(p_{j+1}^ser)
10:        add r_{j+1} to R_mob
11:    end if
12: end procedure
```

4.6 Execution strategy selection

After defining the quality parameters of the simulation, system needs to select between two approaches: either perform the approximate computation using the reduced basis, loaded from server utilizing the RBM, or compute the full problem solution directly on the mobile device. By choosing the appropriate approach, the system is able to dynamically answer user queries for the given set of input simulation parameters μ with minimum error and latency.

In order to make such a decision, the availability of required reduced basis on the mobile device and the status of the communication link between mobile device and server need to be known. The differentiation is made only between two states, namely: connection is available or connection is unavailable. Presented approach doesn't use the network quality prediction techniques or bandwidth monitoring as described in [DDR15], but

this methods could be easily integrated to switch between two approaches with respect to communication quality variations.

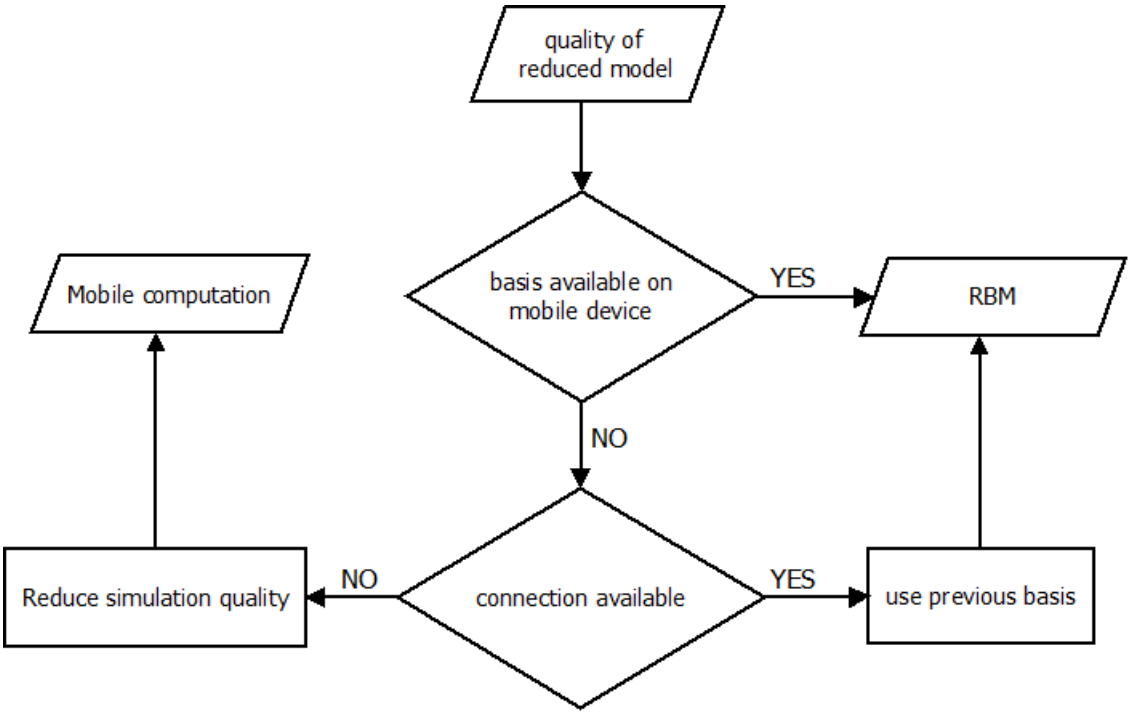


Figure 4.5: Flowchart of solver selection procedure.

The flow of the decision making is illustrated in Figure 4.5. Initial check identifies if the reduced basis for the given simulation quality is already loaded on the mobile device and in such a case it is being reused, utilizing the RBM technique. If the reduced basis is not available in the cache of mobile device and there is no connection with the server, the simulation quality is lowered, by trimming the discretization parameter D to the maximum value D_{max}^{mob} for mobile calculation in order to keep the time of presenting result to the user in adequate bounds. Then the full problem for the given simulation parameter set μ is solved directly on mobile device, thus still allowing to periodically answer user queries and track parameter change. In case of stable connection with the server, the absence of updated reduced model on mobile device means that it is still being requested from the server, so the suitable reduced basis is searched among the loaded bases on mobile device R_{mob} for previous discretization parameter D_{prev}, training set T_{prev} and maximal residual r_{max}^{prev}:

$$R_{mob} \neq \emptyset \implies \exists r_i(D_{prev}, T_{prev}, r_{max}^{prev}), r_i \in R_{mob}, 1 < i < N,$$

where N is the amount of available bases on mobile device. While waiting for the server response, found basis is used to answer user queries with RBM technique.

RBM

The basic approach for solving the numerical simulation, using the reduced basis method (RBM), follows the technique presented in [DSD+17] with slight modifications. The general flow is presented in Algorithm 4.5 and discussed below.

Algorithm 4.5 Reduced Basis Method Strategy

1: **function** REDUCEDBASISSTRATEGY(r_{curr}, r_j,$\mu_1 \dots \mu_n$)
in: r_{curr} represents currently used reduced basis
r_j represents the reduced basis, received from execution strategy controller
$\mu_1 \dots \mu_n$ represents given simulation parameters set for current query
out: u represents the result of numerical simulation

2: **if** $(r_{curr} = \emptyset) \vee (r_{curr} \neq r_j)$ **then**
3: $r_{curr} \leftarrow r_j$ // updated basis is loaded for answering current query
4: **end if**
5: $u_N \leftarrow$ solveReduced($r_{curr}, \mu_1 \dots \mu_n$) // reduced solution for given set of parameters
6: **if** residual(u_N) > r_{max} **then**
7: $T \leftarrow$ TrainingSet([min($\mu_1 \dots \mu_n$),max($\mu_1 \dots \mu_n$)]) // update training set
8: request updated basis for new T
9: **end if**
10: $V \leftarrow$ snapshots(r_curr) // extract snapshots matrix from reduced model
11: $u \leftarrow$ mul(V, u_N) // get the real solution
12: **return** u
13: **end function**

Initially, the reduced basis, received from the execution strategy controller is compared to the previously used one, in order to obtain if the updated reduced model has been loaded. If it is the case, the new reduced basis is read from the cache of mobile device and used for the following steps, otherwise, previous basis is reused. First, the reduced solution u_N is calculated, in order to estimate the approximation error. If it exceeds the maximal residual r_{max}, the new training set T is calculated for the updated parameters range and new basis is requested from the server. In contrast to adaptive approach, proposed by Dibak et al. and discussed in Section 2.5.2, instead of computing additional snapshot for the given simulation parameters, the basis is completely updated for new parameters range. Despite increased computational effort, this method guarantees the upper bound of approximation error to not exceed maximal residual r_{max} for the given range of parameter values. This approach is thus suitable for the systems, that operate mostly in predefined operational ranges, that are changed periodically to examine the system under specific conditions. Finally, the real solution u of the numerical simulation is obtained by multiplying the reduced solution u_N with the snapshot matrix V: $u = Vu_N$, as discussed in Section 2.3.1.

Mobile computation

When the execution strategy controller decides to perform the calculation of the full problem directly on the mobile device, user application needs to construct the linear system $Ax = b$ for the specific partial differential equation (PDE), corresponding to the detected simulation. The input is presented by the given parameters of simulation $\{\mu_i \dots \mu_n\}$, coming from sensor readings, and the full problem dimension, received from execution strategy controller, limited to the maximum value D_{max}^{mob}. In order to build the coefficient matrix A and the right-hand side vector b, the finite difference discretization method is applied to the PDE of detected simulation, as described in Section 2.2.1. After that, the LU decomposition method [Wik17b] is applied to solve the linear system and get the simulation solution (see Figure 4.6).

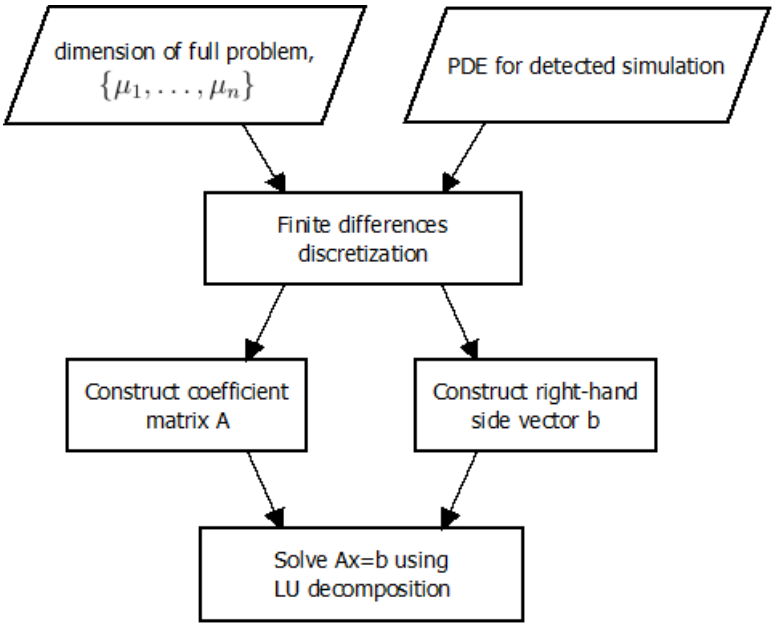


Figure 4.6: Full problem solution flow.

4.7 Result visualization

After the result of the simulation is known, it appears in the form of the solution vector $(u_1, \dots, u_N)$, where N is the count of result values, determined by the discretization parameter D: $N = D^2$. In order to construct the 2D virtual plane for visualizing the solution, this vector should be transformed to matrix form by mapping flat values to the set of rows, assuming the square plane: $(u_1, \dots, u_D), (u_{D+1}, \dots, u_{2D}) \dots, (u_{D^2-D+1}, \dots, u_{D^2})$. After the matrix representation M is constructed, individual cells should be mapped to

the corresponding texture, that corresponds to result value and determines the value of interest(e.g. intensity, temperature, pressure, etc.). Before the texture mapping is done, the result values $u_i, i \in N$ should be first normalized to $[0; 1]$ interval:

$$u_i^{norm} = \frac{u_i - min(u_1, \ldots, u_N)}{max(u_1, \ldots, u_N) - min(u_1, \ldots, u_N)}.$$

Then the normalized values are mapped to given texture $t(u_i^{norm}) : u^{norm} \mapsto C$, where $C = (c_1, \ldots, c_M)$ are the texture color values with M distinct colors. Finally, when the plane is ready, the rotation matrix R and translation vector T are applied to align the hologram with the detected real-object, determined from the step in Section 4.2. Described procedure is shown in Algorithm 4.6.

Algorithm 4.6 Visualization of simulation solution

1: **procedure** VISUALIZERESULT(N, u, R, T)
in: N represents the totall amount of result values
u represents the solution vector
R represents the rotation matrix
T represents the translation vector
2: $rows, cols \leftarrow \sqrt{N}$
3: $M \leftarrow$ matrix($rows, cols$) // define empty matrix
4: **for** row in [0, $rows$) **do**
5: **for** col in [0, $cols$) **do**
6: $i \leftarrow$ mul($row, rows$)+col
7: $u_i^{norm} \leftarrow \frac{u_i - min(u)}{max(u) - min(u)}$
8: $c_i \leftarrow$ texture(u_i^{norm}) // map normalized value to texture color
9: $M_{row,col} \leftarrow c_i$
10: **end for**
11: **end for**
12: apply (R, T) to M // define position of virtual plane
13: **end procedure**

5 Implementation

For system evaluation, all the required components were implemented, in order to solve the following tasks: predefined real-world tags recognition and their mapping to corresponding simulations; efficient computation of numerical simulation, using reduced basis method (RBM); solving the full problem on mobile device in case of network failure; presenting results to the user in the form of virtual augmented object.

The above mentioned components consist of: the client-side middleware, used for managing simulation computation and communication with server; and user application, used for predefined real-world tag detection, identification of simulation quality parameters and data visualization. In this section, the implementation of the above mentioned components will be described in details.

5.1 Numerical libraries

For the presented approach, the solver for computing the full problem solution directly on the mobile device should be found. In order to make such a decision, three different numerical libraries were selected to evaluate their performance, namely the MathNet.Numerics, Accord.Math managed code libraries and Eigen C++ native library. For evaluation purposes, the advection-diffusion numerical problem was selected with the constant right-hand side vector. All of the libraries inherit from the common parent class `MobileSolver`, that provides the `solve` method, taking the diffusion coefficient, advection in x and y directions as parameters. Also the dimension of the full problem is defined by `fullProblemDim` property as shown in Figure 5.1.

NMathSolver

The MathNet.Numerics provides the `DenseMatrix, SparseMatrix` and `DenseVector, SparseVector` classes from `MathNet.Numerics.LinearAlgebra` namespace to represent the dense and sparse matrices/vectors accordingly. In order to solve the linear system of equations, `fullMatrix` and `rhsVector` methods were used to construct the matrix if coefficients and the right-hand side vector, evaluating both sparse and dense matrices, as presented in evaluation.

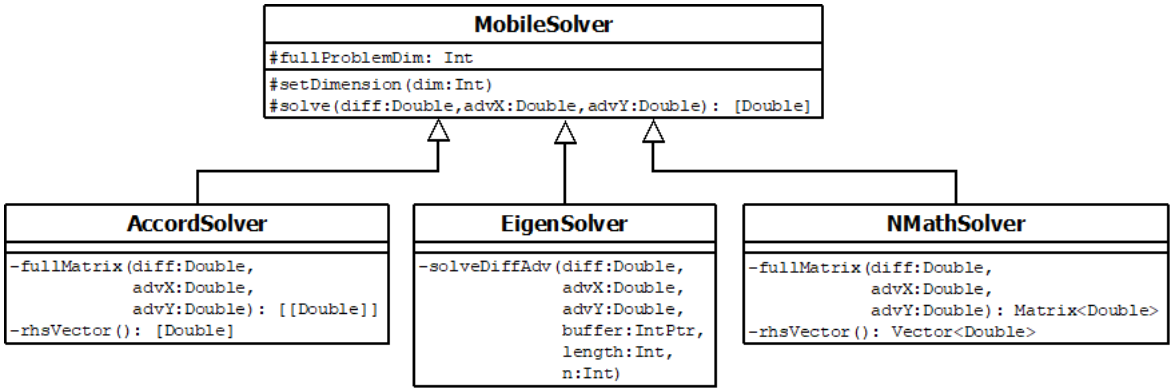


Figure 5.1: UML class diagram for numerical libraries, used for solving the full problem.

AccordSolver

The `AccordSolver` implements the same methods for constructing the coefficient matrix and right-hand side as `NMathSolver`,however, Accord.Math library extends the standart one-dimensional and two-dimensional arrays with additional linear algebra methods, thus making it suitable for extending legacy code, or porting the implementation from other languages.

EigenSolver

The native nature of Eigen C++ library introduces some interoperability issues between the managed and unmanaged (native) code. Thus, it is required to implement the wrapper method `solveDiffAdv`, that calls the native method, only after allocating the needed amount of managed memory and passing the pointer to this buffer, together with its size and the dimension of the full problem. After receiving the result from native method, that is stored in previously created buffer, it should be copied to the local result array, finally garbage-collecting the temporary memory buffer. From the developers perspective this approach appears to be very limiting, time-consuming and error-prone, however, the execution times benefits, presented in evaluation, while using the `EigenSolver`, made it a perfect choice for solving the full problem on the resource constrained mobile device.

5.2 Client-side middleware

The client-side middleware is implemented as a Universal Windows Platform (UWP) library, that utilizes .Net Core framework, written in C# language. The middleware component is used, mostly, for implementing the reduced basis method (RBM) by loading the reduced models from server to the storage of mobile device and computing

the approximate solution of numerical simulation. The implementation part of the basic RBM approach was adopted from [DDR17; DSD+17], ported to UWP platform and extended with additional functionality for presented approach. The UML diagram, representing the relevant class structure of the client-side middleware is shown in Figure 5.2 and described below.

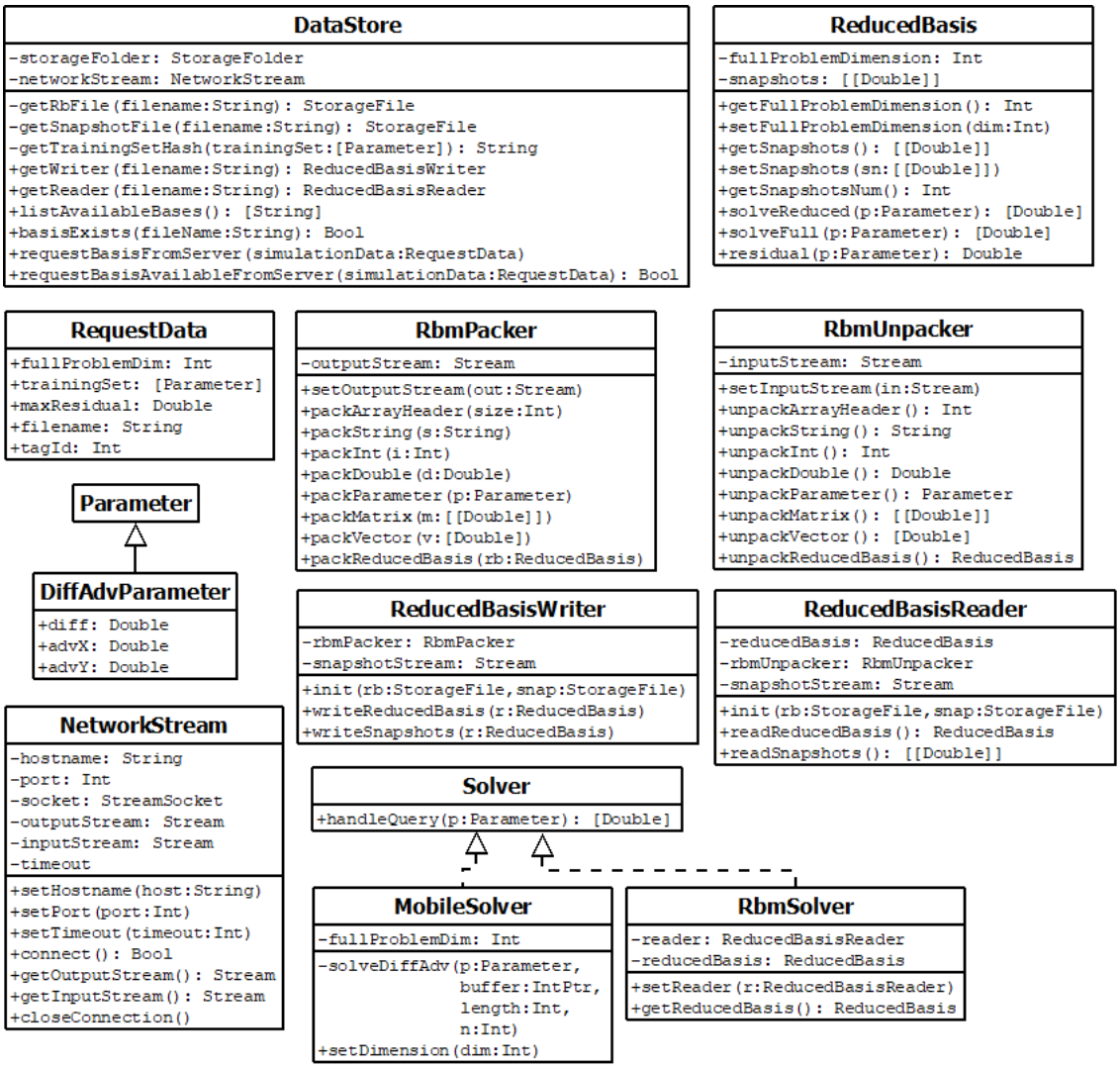


Figure 5.2: UML class diagram for client-side middleware.

NetworkStream

The `NetworkStream` class is used mainly for communicating with the server, defined by the given `hostname` and `port` properties. It opens the TCP socket connection, using the `Windows.Networking.Sockets` namespace and gets the input and output streams for transferring the data between the mobile device and server. `connect` method can

return `False` in case of network failure, or when the waiting time for the response is exceeded.

The maximum waiting time is defined by the `maxTimeout` property, specified im miliseconds. It is passed to the `CancellationTokenSource` object from the `System.Threading` namespace, that effectively closes the connection after specified timeout.

DiffAdvParameter

The `DiffAdvParameter` class is used to represent the parameters for the advection-diffusion numerical problem, namely the diffusion coefficient `diff`, advection in x direction `advX` and advection in y direction `advY`. This class inherits from the general abstract `Parameter` class, used for simulation parameter values. It gives the possibility to extend the approach for different simulation parameters, not limited to advection-diffusion case.

RequestData

In order to request the particular reduced basis from server, it should be uniquely identified by some set of parameters. The `RequestData` class is used to describe the reduced basis by: full problem dimension parameter, that determines the reduced space size; training set, holding the array of `Parameter` to define the parameter range of the reduced model; maximal residual, that defines the maximum approximation error; and real-world tag identifier, in order to request the reduced basis for corresponding simulation, mapped to detected tag. String representations of these parameters are appended to form the unique `filename` property, using the MD5 hash value for the training set, insted of plane string.

RbmPacker/RbmUnpacker

For serializing and deserializing the parameters from `RequestData` class, `MsgPack.Cli` library was used. `RbmPacker` and `RbmUnpacker` classes are used as wrappers for the native library calls, providing convenient methods, such as, for example, `packReducedBasis/unpackReducedBasis` to efficiently serialize reduced model data. Depending on the given input/output stream, these classes can be used for data serialization/deserialization in data-files or network transmission.

Reduced Basis

`Reduced Basis` class is the main class, implementing RBM. It stores the reduced coefficient matrix and reduced right hand side in the form of `SeparableMatrix` (not shown in UML diagram), to calculate the reduced solution of numerical problem for the given `Parameter`, executing the `solveReduced` method. The residual is computed by the `residual` method and `solveFull` is responsible for returning the real solution, multiplying the `snapshots` matrix by reduced solution.

ReducedBasisReader/ReducedBasisWriter

These classes are used for reading/writing the reduced basis from/to internal storage of mobile device. The `init` method extracts the input/output stream from the reduced basis (`rb`) and snapshot (`snap`) files, passed as parameters, creating the `RbmPacker/RbmUnpacker` objects to read/write serialized data, using the corresponding methods.

DataStore

`DataStore` is the main class for providing the interface for the user application to utilize the RBM and communicate with the server. The most important methods are `requestBasisFromServer` and `requestBasisAvailableFromServer`, that both use `NetworkStream` object and `RbmPacker/RbmUnpacker` to transmit the `RequestData` parameters to server and receive the awaited response. `requestBasisFromServer` method awaits the `ReducedBasis`, that can be saved, using the `ReducedBasisWriter`. Whereas `requestBasisAvailableFromServer` receives `True/False` values, that define the availability of precomputed reduced basis on server, by comparing the MD5 hash value of requested one with hash values of all precomputed reduced models. The later is used by the user application to determine if the basis with enhanced quality should be requested in advance.

In order to estimate whether reduced model is available in the cache of the mobile device, `basisExists` method is used. It tries to find the reduced model, with the given `filename` in Local folder, given by the `storageFolder` property, using the `TryGetItemAsync` method from `Windows.Storage` namespace. The result of this method is used by the user application to decide on the execution strategy.

Solver

The `Solver` interface provides the method to handle parameter update queries, coming from the user application. It is realized by the `RbmSolver` and `MobileSolver`, that both are used for computing simulation solution, either by reading the loaded reduced basis and getting the approximate solution, or solving the full numerical problem in the form of linear system accordingly.

5.3 User application

User application is implemented, using Unity3D engine, and including the client-side middleware as an UWP library managed plugin. Marker-based real-world tag recognition implementation is based on the Unity wrapper of ARToolkit computer vision library, found in [Qia17]. User application is used for mapping the detected real-world tag to

corresponding simulation, determining the quality requirements, reading environmental sensor values and deciding on the execution strategy to periodically update simulation results with low latency, minimizing the storage usage by loading only relevant reduced bases. The specific of Unity3D engine is that it represents virtual objects by, so called, `GameObjects` in the scene, with different execution scripts attached. Thus, user application mainly consists of specific scripts, that are executed each frame to monitor the state of the system and update the corresponding `GameObjects` as required. The UML diagram, representing the relevant class structure of the user application is shown in Figure 5.3 and described below.

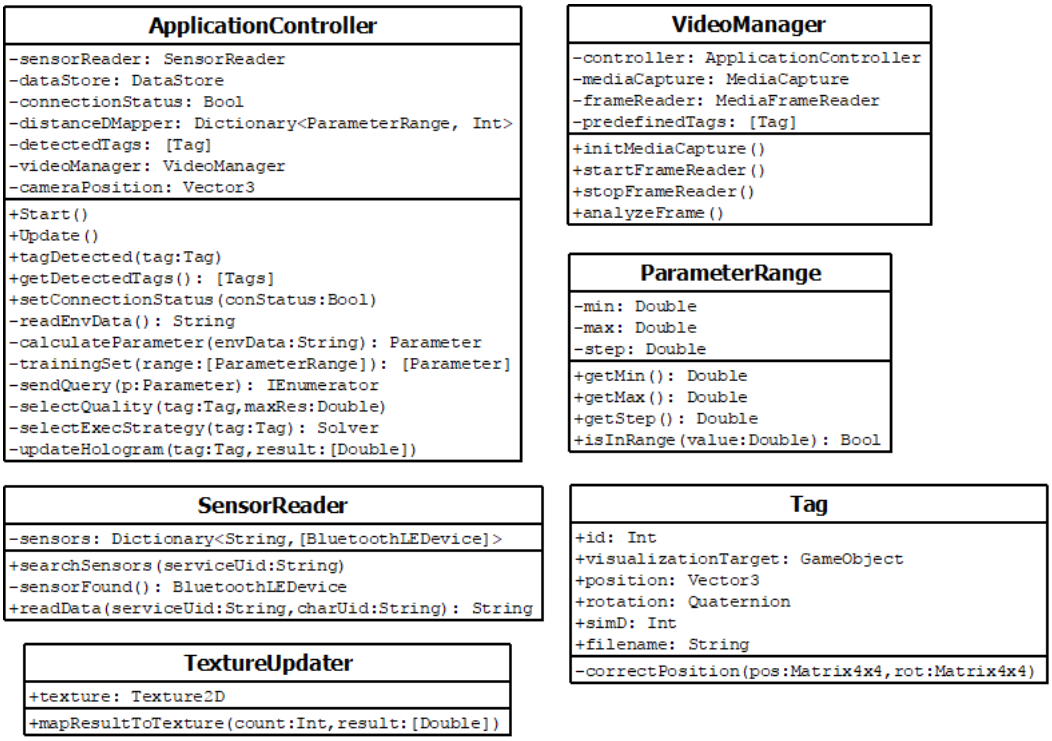


Figure 5.3: UML class diagram for user application.

Tag

Class `Tag` is used to represent the real-world tag, in the form of fiducial marker, and its properties. The `id` property is used to map this tag to the corresponding simulation, sending it within the `RequestData` parameter, while requesting the reduced basis from the server. After receiving the corresponding tag identifier, the server-side middleware will find the corresponding simulation and later basis requests will utilize this knowledge to find the suitable reduced model from RB-models database. `Tag` object also saves its world `position` and `rotation` to follow the position and pose of the real-world marker. This values are periodically updated by `VideoManager`, described below.

To every Tag object the corresponding visualizationTarget GameObject is attached, presented in the form of Quad with specific material and texture. It is used to construct the virtual plane of simulation results overlaid on the detected real-world marker, using position and rotation properties. In order to eliminate the alignment offset, correctPosition method is applied, using affine transformation.

The last property of interest is the discretization parameter of the mapped simulation simDim, that defines the quality requirement of simulation results and comes from the selectQuality method of ApplicationController.

SensorReader

The SensorReader class is responsible for searching Bluetooth Low Energy (BLE) devices for the passed serviceUid through searchSensors method. Internally it calls the BluetoothLEAdvertisementWatcher object from Devices.Bluetooth.Advertisement namespace in order to listen to the broadcasted data of environmental sensors. When device with corresponding service Uid is found, the sensorFound callback is called, adding the BluetoothLEDevice object to the values array for the given serviceUid key in sensors hashmap data-structure.

The readData method is periodically called by the ApplicationController to calculate the updated simulation parameters. It loops through all the found sensors for the given serviceUid and reads the sensor values, using GattCharacteristic object, retrieved for the passed charUid.

Interestingly, that Hololens augmented reality glasses, used as the mobile device, run the Anniversary Update version of Windows 10 and, by the time of writing, the Creators Update version is still not available. The latest update brings improved BLE API, that doesn't require to first pair with BLE device, in order to read services and characteristics. However, with the current BLE API on Hololens, devices of interest should be paired to mobile device in advance, that seriously limits its capabilities to collect all available environmental sensor data in arbitrary location.

VideoManager

VideoManager is used to analyze each frame, coming from integrated video camera of the mobile device, search specific features and correlate them with the predefined set of known marker patterns, stored in predefinedTags property. When tag is found analyzeFrame method first checks if the tag was already recognized by searching it in detectedTags array of ApplicationController. If it is not the case, it calls the tagDetected callback function to add it to the list of detected markers. Finally, it updates the position vector and rotation matrix of the found Tag, by changing corresponding properties.

ParameterRange

The `ParameterRange` is a simple class to define the range and step-width of simulation parameters for constructing the training set and map the simulation discretization parameter to the distance from real-world tag to the user.

ApplicationController

The core class of the user application is presented by the `ApplicationController`. In the `Start` method, that is called by Unity3D engine at the beginning of program execution, the `videoManager` instance is initialized by calling the `startFrameReader` method. `Update` method is called each frame and loops through all the `detectedTags`, passing them to `selectQuality` method, together with the previously defined maximal residual value.

The quality is determined by measuring the distance from the user (known camera world coordinates) to detected tag and mapping it to the simulation discretization parameter, received from `distanceDMapper` hash-map. If the discretization parameter is updated, that is, it is not equal to the `simD` property of the `Tag`, `qualitySelector` first tries to find the reduced basis for the updated quality in the cache of mobile device. If it is found, or connection with server is broken, the `simD` property of corresponding tag is updated. Otherwise, reduced basis for the updated discretization parameter is requested from the server, by calling `requestBasisFromServer` method from client-side middleware `DataStore` class, following by `requestBasisAvailableFromServer` call for the next discretization parameter from `distanceDMapper`. If the reduced basis for future quality update is not available on server, another `requestBasisFromServer` method call is performed to load it in advance.

In order to periodically update the simulation results, depending on environmental sensor readings, the `sendQuery Coroutine` is called each second. It first calls the `readEnvData` method and gets the simulation parameters by passing the environmental sensor data to `calculateParameter` method. For current implementation, we model the parameter values by random distribution in specific range. Then the `selectExecutionStrategy` method is called to decide on the `RbmSolver` or `MobileSolver`, depending on the availability of reduced model on the mobile device, obtained by means of `basisExists` method of `DataStore` and connection status. If the basis is not available and connection is lost, full numerical simulation is computed by `MobileSolver`, limited to the maximal discretization parameter for low latency. Otherwise, approximate solution is calculated with the help of `RbmSolver`, using updated, or previously loaded basis.

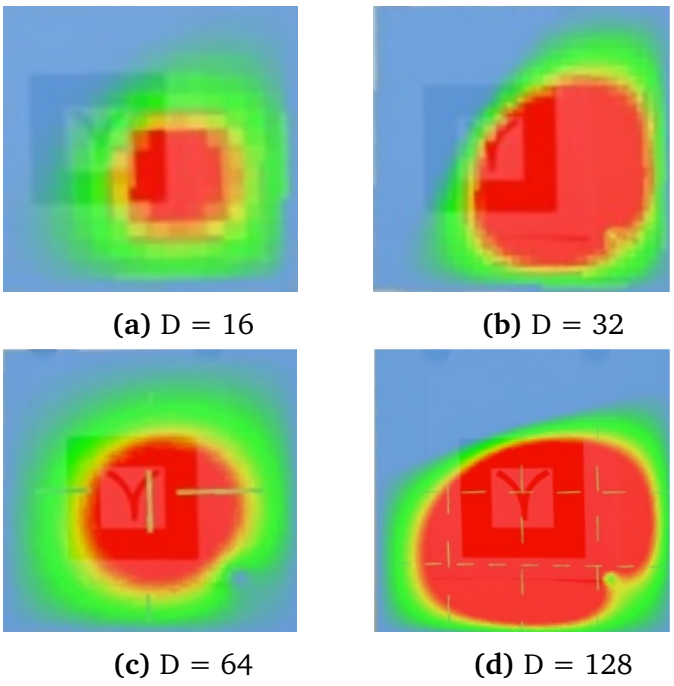
After solution is available, `updateHologram` method is called to update the `visualizationTarget` for the given tag, that transfers the execution to the `TextureUpdater` script.

TextureUpdater

`TextureUpdater` acts as a script, attached to the `visualizationTarget` `GameObject` property of the detected tag. Its main function is to update the texture, by mapping the simulation results to corresponding texture color, thus updating the visual representation of virtual plane, using the `mapResultToTexture` method.

5.4 Virtual object construction

In order to construct virtual plane, two approaches were developed: passing uniform array of simulation results to custom fragment and vertex shader; and dynamically constructing the 2D texture, applied to the visualization target material.



(a) D = 16 **(b)** D = 32

(c) D = 64 **(d)** D = 128

Figure 5.4: Heatmap holograms, using custom shader, for different discretization parameter D.

The maximum size of uniform array, that can be passed to custom shader is limited by 1023 distinct values by the graphical unit of Microsoft Hololens. Due to this limitation, there is a need to construct multiple quads for discretization parameter values greater than 32. This approach needs to first initialize required amount of Quad `GameObjects`, followed by their scaling and calculating the submatrix of simulation solution to produce corresponding matrices with 32x32 dimensions. Another task is their correct alignment

with each other. In order to solve this problem, all the constructed quads were placed within the common root `GameObject` and were positioned and scaled relative to the root. However, as seen from Figure 5.4, even after relative alignment, artifacts, in the form of dashed lines, appear on the borders of individual quads. Moreover, adding additional virtual objects to the scene decreases rendering performance as shown during evaluation.

In order to overcome above mentioned limitations, another approach of dynamic texture generation was found. From the predefined texture, passed to `TextureUpdater`, initially all available color values are read and stored in `colors` array property. When the call to `mapResultToTexture` method is performed from the `ApplicationController`, new `Texture2D` object is constructed, with dimension of simulation result. The solution vector is mapped to the corresponding color values, read from `colors` array and the generated texture is applied to the material of `visualizationTarget` for the given tag.

6 Evaluation

In this chapter the evaluation of the system with respect to latency, energy consumption and visualization quality is presented. In the evaluation different quality metrics are considered. First, the efficiency of different libraries for solving the numerical simulation directly on mobile device is evaluated. Then, the latency of requesting the reduced basis from server is examined for different input parameters, taking into account two distinct cases: when the basis is already precalculated on the server and when the server should first solve the full numeric problem before sending the result back to client. For all of the cases the power usage of the mobile device is measured.

6.1 Evaluation Setup

Before presenting the results of the study, the hardware and software setup of the evaluation is given. All the measurements are performed on middle-range laptop and augmented reality glasses, that communicate with the server via wireless communication and fetch sensor data from the programmable sensor device (see Figure 6.1).

All run-time evaluations are performed on two distinct devices, Lenovo Thinkpad E460 (20EUS00000) and Microsoft Hololens v1.0 (Development Edition). Despite representing the different classes of devices, they both run on Windows 10 operation system, thus giving the possibility to develop the cross-platform Universal Windows Presentation (UWP) application.

For the communication link the IEEE 802.11 (WiFi) technology was used. The latency between the client device and the server had the minimum value of 2.9ms, maximum value of 20.8ms and the average of 8.3ms as given by the ping command. The bandwidth in its turn was between 16 MBits/s and 10 MBits/s. For simulation of sensor data broadcast the Bosch XDK toolkit was programmed to gather the temperature data from the surrounding and transmit it as the Bluetooth Low Energy (BLE) user-defined characteristic with Alpwise exchange service.

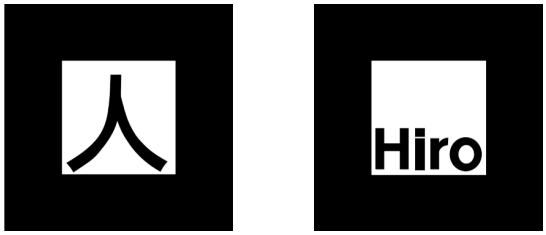
The numerical simulation was based on the stationary advection-diffusion equation as described in Section 2.2.4. This type of equation can potentially be used for steady

Figure 6.1: Augmented reality glasses, fiducial markers and programmable sensor, used for building the prototype.

state of different systems, where both advection and diffusion processes are happening, for example the heat distribution on some surface, or fluid mixing in the chemical reactions. The main input parameters are the diffusion coefficient μ_{diff}, the advection in horizontal μ_{advx} and vertical μ_{advy} directions. For the efficient approximate solution on the mobile device the distributed RBM was used as described in Section 2.5, that uses additional parameters, namely the simulation discretization parameter D, the training set $T = \mu_{diff}, \mu_{advx}, \mu_{advy}$ for the reduced basis request and maximum residual value r_{max} that defines the maximum allowed error of the simulation result.

The implementation was based on UWP platform (version 5.4.0) using the C# programming language for the client application combined with the Unity3D Engine (version 2017.1) for object recognition and hologram construction. For real-world tag detection two libraries were used: Vuforia (version 6.2.10) and ARToolkit (version 5.3.2) wrapper for UWP platform together with the target images in the form of two distinct fiducial markers, namely Kanji and Hiro patterns (see Figure 6.2).

In order to solve linear systems on mobile device three different numerical libraries were used: MathNet.Numerics (version 3.20.0), Accord.Math (version 3.0.2) and Eigen C++ native library (version 3.3.3). Eigen C++ native library was chosen for computing the full problem on the mobile device, based on the evaluation results presented below. For optimizing the data transfer between client and server the MsgPack.Cli (version 0.9.0) serialization library was used. For visualization of the simulation results in the form of heat-map hologram two approaches were used: custom vertex and fragment shader, using the Cg/HLSL language, and dynamic texture construction, using Unity3D engine.



(a) Kanji **(b)** Hiro

Figure 6.2: Fiducial markers, used for object tracking.

6.2 Numerical libraries comparison

Three numerical libraries for solving the full simulation problem were evaluated, namely MathNet.Numerics, Accord.Math and Eigen C++ native library. The times for solving the full problem for different discretization values on the laptop and mobile device are depicted in Figure 6.4 and RAM consumption is shown in Figure 6.3. Naturaly, the execution times are lower on the laptop, due to more powerful CPU. However, the relative difference between the libraries is the same and if not stated otherwise, the following results discussion will correspond to the Microsoft Hololens mobile device.

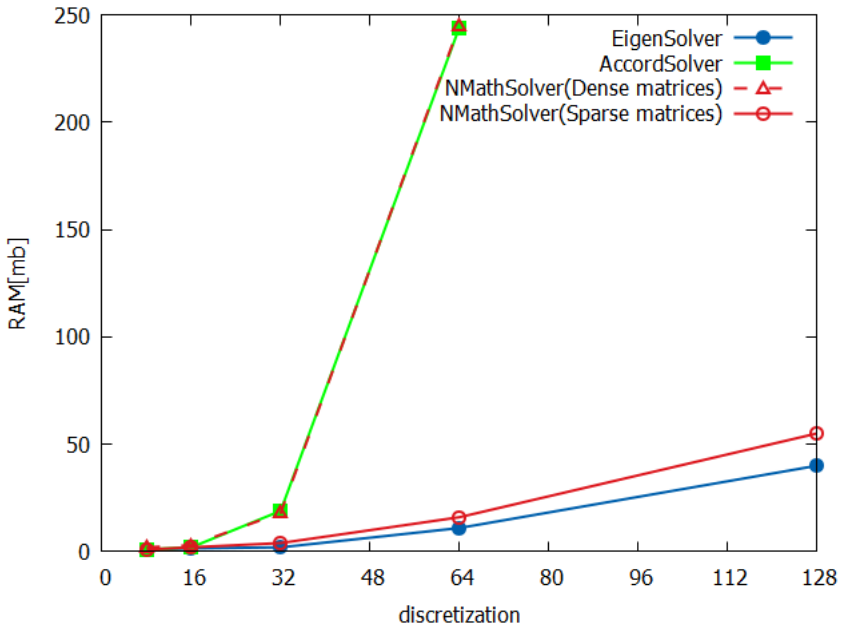
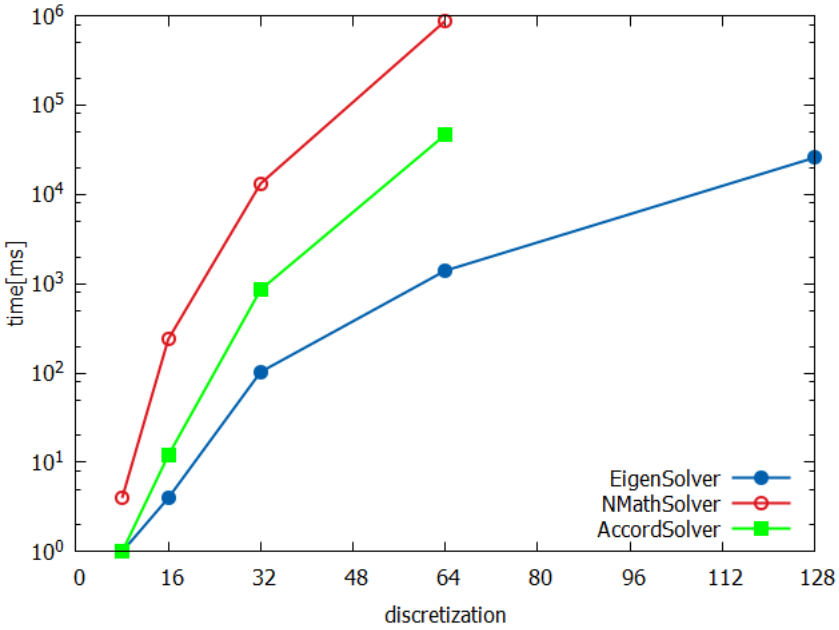
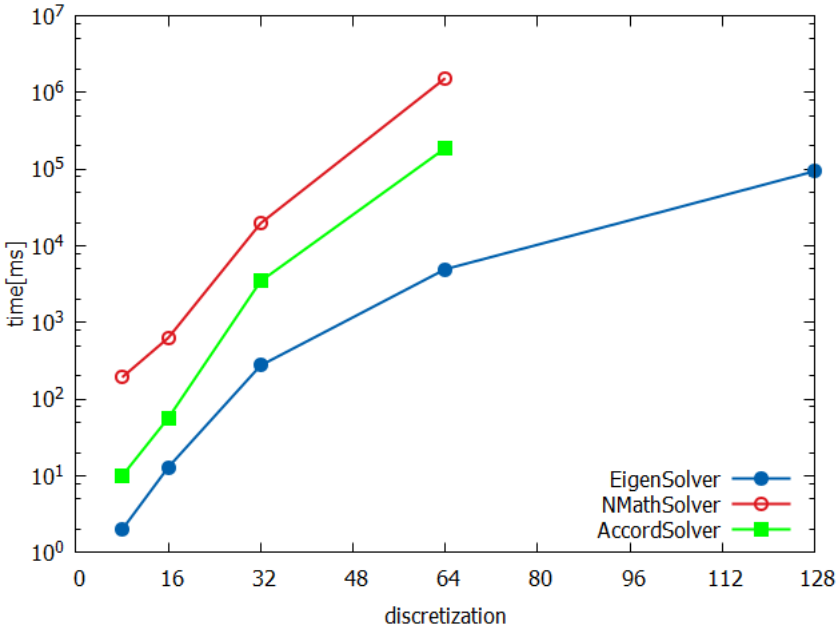


Figure 6.3: Numerical libraries memory(RAM) consumption comparison for solving full numerical problem.



(a) performed on the laptop



(b) performed on the mobile device

Figure 6.4: Numerical libraries execution time comparison for solving full numerical problem.

MathNet The worst results showed the MathNet.Numerics being from 20 to 600 times slower than others, not making it feasible even to evaluate 128x128 discretization level. It achieves good memory consumption, using the sparse matrix for construct-

ing the matrix of coefficients A,increasing the RAM only by 15mb, going from 8x8 to 64x64 dimension. When the dense matrix is used instead, performance is increased by approximately 3 times, but the memory consumption growth rapidly, achieving 245mb leap for 64x64 problem dimension.

Accord Accord.Math showed middle-range execution times, being from 3 to 30 times slower compared to Eigen native library. Moreover, it produces high memory consumption, utilizing 244mb RAM for 64x64 problem dimension as in the case of dense matrix usage in MathNet.Numerics and throws the OutOfMemory exception for 128x128 problem size. However, for low dimensions such as 32x32 the execution time never goes above 1s, that makes it suitable for solving the low-dimensional linear systems, utilizing simple data structures as double and float arrays with extended functionality.

Eigen Eigen library due to its native code nature easily outperforms its competitors in both execution time and memory consumption. For the 32x32 dimension, the execution time doesn't exceed half a second, consuming only 2 mb of memory. However, for 64x64 problem size the solution appears only after approximately 5s. It efficiently utilizes the available RAM, causing transient memory burst of 40 mb even for the 128x128 problem dimension. Thus, it is the perfect choice for solving the full numerical simulation problem directly on the mobile device, but the problem dimension shouldn't go above 32x32, otherwise QoE will be considerably degraded. Additionally, because of native with managed code interoperability issues for different simulations the UWP library wrapper should be constantly adapted, that substantially limits its generic usage.

For all three libraries, input parameters for solving the advection-diffusion numerical problem were constant and had the following values: $(\mu_{diff} = 1.0, \mu_{advx} = 1.0, \mu_{advy} = 1.0)$ for the diffusion coefficient, the advection in horizontal and vertical directions accordingly. During the execution of `Solve()` method, the processor with four virtual cores was loaded only by 25%, that means no multi-threading optimization is used by either of them.

Interestingly, that both managed libraries Accord.Math and MathNet.Numerics spend 99% of execution time for solving the linear system and native library Eigen spends 98% of time for constructing the coefficient matrix A. Evidently, it can be explained by the different memory organization models used by the managed and unmanaged code, as well as the usage of sparse matrices with further compression by Eigen library.

6.3 Object recognition

For mapping simulations to the corresponding real-world objects, specific tags of the surrounding were recognized, represented by the predefined image targets. For tag identification two computer vision libraries were examined, namely Vuforia and ARToolkit. Vuforia is the well-known and highly supported commercial solution for object detection, that provides free license for development and evaluation purposes, whereas ARToolkit, while still allowing to perform marker-based recognition, is the open-source library giving higher flexibility, free usage and ability to adapt it, according to developers needs.

ARToolkit Squared fiducial markers with black border were used as the recognition targets for ARToolkit. Particular interior image patterns of such markers allow to uniquely identify the real-world tag. The maximum distance of marker recognition is approximately 8m for the Kanji pattern with 14x14cm outer border size. ARToolkit allows the dynamic pose estimation, that makes it suitable for tracking the moving objects or the objects, constantly changing their orientation.

However, initially ARToolkit can't extract the exact position and rotation of a given marker, introducing the offset, that depends on the camera calibration parameters. In order to correctly align the virtual object with the detected tag, the offset elimination techniques are applied, such as affine transformation as described in [QAKN17]. The hologram overlay before and after applying the position correction procedure is shown in Figure 6.5.

Depending on the lighting conditions and surrounding objects, false-positive marker detection can appear for single fiducial markers due to only four feature points of outer border, though false detections can be minimized by combining multiple markers in the greed or choosing unique exterior shape [KPS10].

(a) initial pose estimation with offset

(b) offset elimination with affine transformation applied

Figure 6.5: ARToolkit marker recognition and alignment correction.

Vuforia Vuforia was used to detect the image target with high amount of feature points, previously extracted and stored in a database, that is loaded together with the application on mobile device. It results in better position detection and stability of the overlaid virtual object, however for the image of 20x14cm size the maximum detection distance was limited only to 2m. Additionally, despite providing the extended tracking capabilities (ability to dynamically follow the object), position information was updated roughly each second, thus limiting its application to mostly static real-world objects.

But if dynamic tracking is not needed and the main requirement is accurate alignment of real and virtual objects at a short distance from the user, Vuforia can be a suitable choice, without the need of additional camera calibration and further correction as illustrated in Figure 6.6.

Figure 6.6: Vuforia target image recognition and alignment.

In terms of processing power consumption both of the libraries are comparable: the overall memory consumption doesn't exceed 100mb and the CPU usage varies between 30% to 40% in the tracking mode and drops to 5-10% in standby mode, with Vuforia being slightly more efficient, due to its optimization for Unity engine. The usage of ARToolkit, on the other hand, requires additional wrapper and manual methods extraction to run under UWP platform, thus the optimization task is the responsibility of the application developer.

Thereby, both libraries can be used for simulation tag recognition and provide acceptable object position and pose estimation for accurately overlaying the hologram to provide the user with additional data. However, despite more stable behavior of Vuforia library, the commercial license seriously limits its usage in non-commercial and research projects. On the other hand, due to its extension capabilities and open-source nature, the robustness of the ARToolkit library can be enhanced by designing specific markers as described in [OXM02] to decrease the amount of false-positive detections and better extract the world-space coordinates.

6.4 Basis request latency

In this section the latency for requesting the reduced basis from server is evaluated with respect to different discretization of the numerical problem. The results are examined for two scenarios: when the basis is available on server (cache hit) and when it should be first computed (cache miss). In order to evaluate the influence of the passed training set, three different sets are constructed with parameter vectors $(\mu_{diff}, \mu_{advx}, \mu_{advy})$: A, B and C, such that $A \in B$, $A \in C$ and $B \notin C$, $C \notin B$. The values for the advection-diffusion parameters and their count for each training set are shown in Table 6.1, where $[a, b]$ is the interval with the minimum a and maximum b parameter value discretized with step width 1.0. The maximum residual value was the same for all training sets $r_{max} = 0.01$.

Table 6.1: Training set parameters

	μ_{diff}	μ_{advx}	μ_{advy}	# of parameters
Training Set A	[1;5]	[1;5]	[1;5]	125
Training Set B	[1;10]	[1;5]	[1;5]	250
Training Set C	[1;5]	[1;10]	[1;10]	500

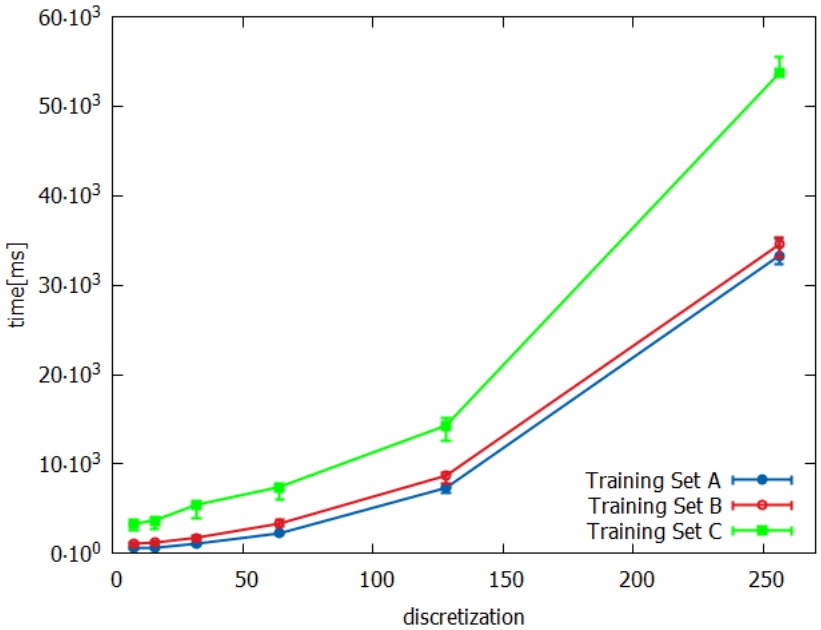


Figure 6.7: Latency comparison when the basis is not available on the server (cache miss) for different training sets.

The basis availability request for querying if the bases is already precomputed on the server and available in the server cache took, for all the training sets and discretization

levels, on average 20 ms, influenced only by the connection quality. It is caused by the fact, that the hash for the training set is calculated on the client, thus sending and receiving the same payload regardless of the parameters.

Figure 6.7 compares times needed to request and load the reduced basis from server to mobile device depending on the discretization parameter for different training sets in case it is not available on the server. In this case, the time for calculating the reduced solution for the full numerical problem on the server should be taken into account, because it becomes the major factor in overall request-response cycle. It can be clearly seen, that increasing the amount of training set parameters, thus extending the range of reduced model's applicability, takes more server time, non-linearly degrading the latency. Such wise, going from 125 to 250 parameters only implies the delay of about 15%, but the training set with 500 entries increases the latency already by 60%, taking about 14s for the 128x128 problem dimension. So, it is better to request the reduced bases, not available in server cache, with the small training sets, that match environmental conditions and update them in case of dynamic sensor values change.

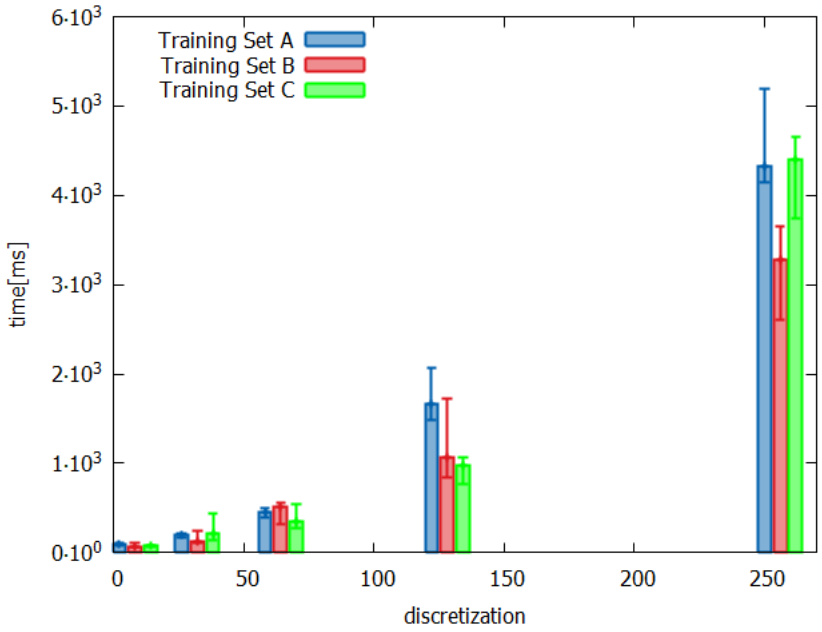


Figure 6.8: Latency comparison when the basis is available on the server (cache hit) for different training sets.

Another scenario is having the reduced bases precomputed on the server, so called cache hit. The results are presented in Figure 6.8. It can be seen, that latency depends mainly on the discretization parameter, on the other hand, the training set plays a minor role and, taking the confidence interval into account, barely influences the time of data

delivery. This behavior can be explained by the algorithm of snapshot generation on the server. For the presented training sets the number of generated snapshots for D=256 are: 15 for A, 14 for B and 23 for C; so the generated reduced bases files have nearly the same size and the latency is defined mainly by the communication link quality and the discretization parameter. Of course, substantially increasing the amount of parameters in the training set leads to large snapshots count and increased data size. But as stated in previous paragraph for cache miss scenario, large training sets take inadequate server computation times, so this case should be generally avoided.

Obviously, the latency is drastically improved for cache hit, compared to the cache miss scenario by up to 5 times for the training set A with 125 parameters. It is also clear, based on the previous reasoning, that this difference in latency will only grow for increased number of training set parameters, e.g. for training set C with 500 parameters, the latency for requesting the basis that should be first computed on server is increased by up to 10 times compared to loading the precomputed reduced model. This results confirm, that estimation of basis availability, presented in developed approach, improves the overall latency by starting to compute required reduced model on server before the user actually requests it for increased quality demands.

6.5 Execution strategies comparison

In this section the evaluation of different execution strategies is described, namely: latency to solve the full numerical problem directly on the mobile device; latency to calculate approximate solution, using the reduced basis stored in the cache of mobile device; latency to load the reduced model from server and solve reduced numerical problem in case of server cache hit and server cache miss scenarios. The results, presented in Figure 6.9, were measured for real-world scenario, when the system constantly tracks the environmental tags, collects the sensor readings and estimates simulation quality demands, thus absolute values of time, needed for full problem solution on mobile device with Eigen native library, are increased, on average, by 2 times, compared to results depicted in Section 6.2.

It can be easily seen, that for the discretization parameter values up to D=16, the mobile device is able to compute the full numerical solution within the same time frame as getting the approximate result, using the RBM approach. Moreover, if the reduced basis is not available in the cache of the mobile device, it is reasonable to execute simulation on mobile device directly, while waiting for the response from the server, even in case of server hit scenario for D=32. This value of discretization parameter, however, determines the maximum quality of simulation results for mobile computing in case of network failure in order to keep the latency of presenting results to the user in

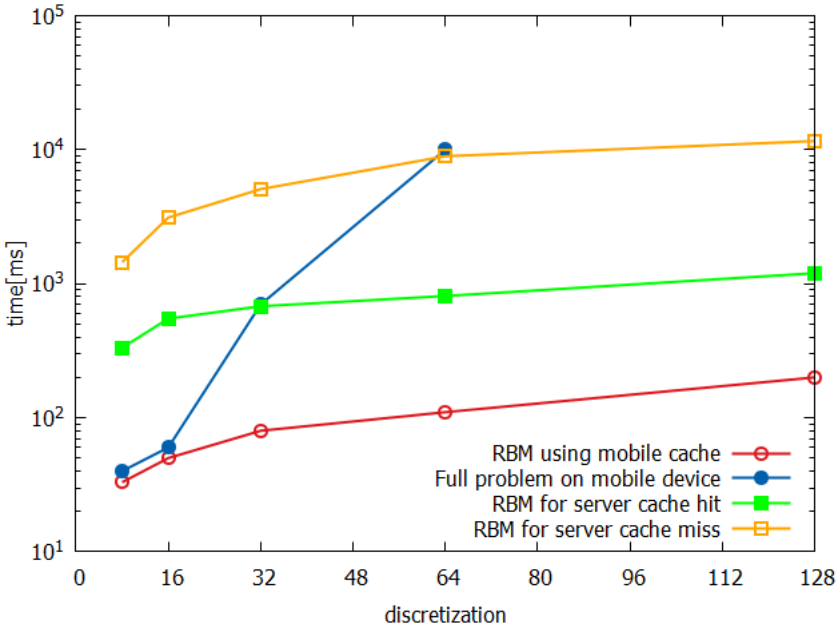


Figure 6.9: Simulation solution calculation time for different scenarios.

adequate bounds. On the other hand, when the required reduced basis is available on server, the quality can be greatly improved while still providing the low latency, following user expectations, such-wise even for D=128 the time for obtaining the approximate result takes on average 1s. Here we can again observe the need to request the updated basis in advance for the reduced model not precomputed on server, that, depending on quality update request time, can improve the latency by up to 10 times for D=128. Otherwise it is not possible to dynamically update the simulation quality constraints without seriously influencing user experience.

6.6 Data visualization

For visualizing the simulation results, corresponding heat-map hologram is constructed, with the adaptive resolution, depending on the simulation quality, determined by the discretization parameter D. In order to construct such a heat-map, two techniques were used: passing the solution vector of simulation to the custom shader as a uniform array; and dynamically constructing the heat-map texture from simulation results and applying it to the standard diffuse shader.

Due to custom shader approach limitations, described in Section 5.4, increasing the amount of virtual objects, presented by multiple quads, substantially drops the overall rendering performance, making it almost impossible to interact with the system of

high quality. However, the approach for dynamically generating the heat-map texture barely influences the performance, producing the drop of 5 frames per second (FPS) for the highest quality with 128x128 resolution (see Figure 6.10). The average time of heat-map construction for both approaches varies from 20ms to 40ms, depending on the resolution, thus the overall latency of presenting result to the user is mainly determined by the time needed to solve the numerical problem, as presented above.

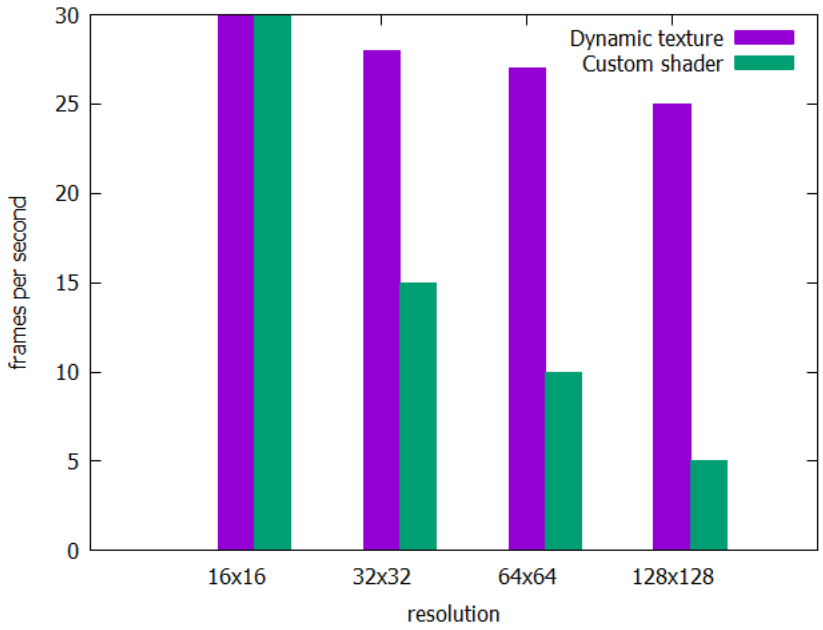
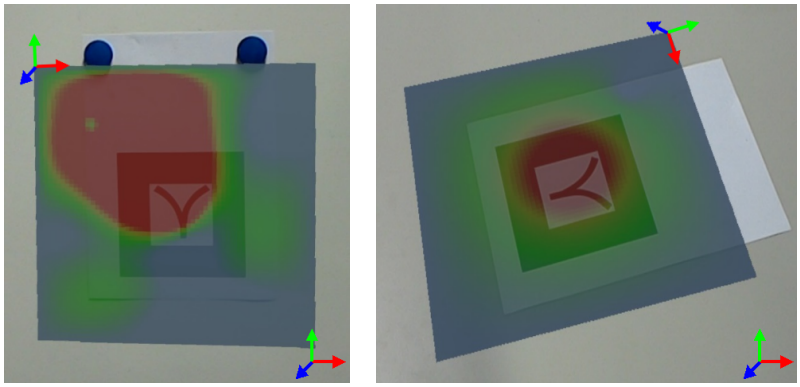


Figure 6.10: Performance comparison for heatmap hologram rendering, using different construction strategies.

The example of the constructed heat-map, using the dynamic texture generation approach is shown in Figure 6.11 for the resolution of 64x64 points. The pose estimation for different marker orientation is illustrated, as well as the difference of the produced heat distribution for the case, when the error of approximate simulation result is greater than maximal residual r_{max} (see Figure 6.11a) and when the error is less than r_{max} (see Figure 6.11b), after the training set for the reduced basis is adapted in accordance with the given range of input parameters. It can be seen, that for approximation error value below the r_{max}, the heat-map follows the normal Gaussian distribution, that is determined by the form for the right-hand side vector b of linear system $Ax = b$ for algebraical form of stationary advection-diffusion equation, whereas exceeding the r_{max} results in random heat distribution and can influence the correctness of system analysis.

All the results, described above are applicable for running multiple simulations for different real-world tags simultaneously, degrading the overall performance only by



(a) simulation error is greater than maximal residual r_{max}

(b) simulation error is less than maximal residual r_{max}

Figure 6.11: Heatmap holograms, constructed using dynamic texture generation and pose estimation.

10% in case of two parallel executions. By mapping the simulation results to different textures, it is possible to construct augmented data in most suitable representation for the given physical process. In such a way, for the Hiro pattern, another instance of advection-diffusion simulation was chosen, using the semi-transparent texture, that can be used, for example, to observe the flow of some physical or chemical particles on a surface (see Figure 6.12).

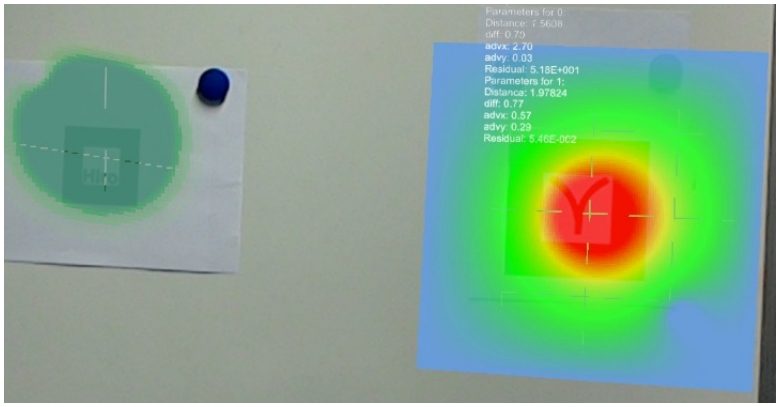


Figure 6.12: Parallel execution of two distinct simulation instances, using advection-diffusion numerical problem.

7 Summary and Outlook

7.1 Summary

This master thesis describes an approach to map the recognized real-world objects to corresponding numerical simulations in order to model and observe different physical processes. For solving the identified numerical simulation, reduced basis method (RBM) is used, loading only required reduced models to internal storage of mobile device. Results of the simulation appear in the form of augmented virtual data overlaid on top of detected object.

While RBM allows to efficiently solve numerical problem by means of reduced model, it is not possible to load all reduced bases for different simulations and quality parameters to the memory of mobile device in advance. Additionally, the latency for presenting simulation results to the user should be minimized and finite, in case of network failure. In order to solve described problems, the system was developed, consisting of: the mobile device, that detects real-world objects, tracks and adapts simulation quality parameters; server, used to compute reduced models for the corresponding simulation; and middleware component, that acts as the distributed controller.

Collecting environmental sensor data and reacting to user perception quality demands allows to load only relevant reduced model of corresponding full numerical problem, thus optimizing the usage of the bandwidth and internal storage of the mobile device. Specific real-world tags recognition and their mapping to particular simulations allows to dynamically switch between different systems of interest, available in specific location, increasing the mobility of the user. Using augmented reality for simulation results visualization improves user interaction with the observed system, by allowing to adapt the input parameters and monitor the reaction of the system simultaneously.

The related work papers in this field don't take limited storage constraints into account and concentrate on solving one particular simulation. In addition, most of them don't analyze the sensor data, available in the current location, often using only sensors available on mobile device, and mainly focus on power consumption optimization, not taking the quality of user experience and current state of the system into account.

The presented work uses the augmented reality glasses, equipped with RGB video camera, to constantly analyze the surrounding, detect predefined real-world tags and read environmental sensor values. All the collected data is then sent to server in order to identify the required simulation and construct the reduced model, depending on the received operational range and quality of user experience (QoE). After the corresponding reduced model is loaded to the mobile device, it is used for efficient computation of numerical problem and presenting the results to the expert with low latency in the form of hologram, that is periodically updated in accordance with sensor readings.

Developed system concentrates not only on specific tag detection, but also on estimation of its exact location and pose in order to superpose the real and virtual object and track dynamic systems. For alignment error elimination, introduced by camera-space to world-space coordinates transformation, the manual correction procedure is applied to adapt camera calibration parameters. Additionally, the distance to the identified object is constantly estimated that is used for determining the user quality demands and adapting the quality of simulation results, that are mainly determined by the discretization factor, accordingly.

For minimizing the latency of presenting the simulation results to the user, while requesting the updated reduced model, additionally, the availability of the reduced basis for increased quality constraints on server is obtained, utilizing the hash function. In such a way,computation of the reduced model for future request is performed in advance, thus substantially decreasing the waiting time for enhanced quality requirements.

In case of network failure, approach, presented in this work, adapts the strategy of computing simulation solution, either by using preloaded bases or by solving the full problem directly on mobile device. Despite degrading the quality of simulation results, this method still allows to track current context of the system without breaking user expectations.

Output of the system is visualized by means of a grid on a 2D virtual plane. This plane appears in the form of a texture map, with individual cells color values coming from simulation result to texture mapping. The resolution of the color map is changed depending on the dimension of problem solution, that is determined by the distance to the user, for more detailed analysis of the system.

In order to investigate the properties of the proposed system, all the required components were practically implemented. They consist of: the UWP library for implementing the middleware component; user application, implemented with Unity3D engine for object recognition and data visualization; and the script for the programmable sensor device, that simulates environmental sensor values broadcasting. As the example of the numerical simulation, 2D stationary advection-diffusion equation was chosen with these input parameters: diffusion coefficient, advection in x and y directions.

Evaluation results show, that the particular real-world tag can be definitely recognized within the distance of 8m, providing the correct pose estimation, after applying the correction procedure. The availability of the reduced model on the server decreases the request time for updated basis by up to 5 times for the simulation training set with 125 parameters, and by up to 10 times for the set with 500 parameters. For the discretization parameter values up to D=32, solving the full numerical problem directly on the mobile device results in acceptable latency, especially when compared with the time to request the reduced model not precomputed on server (server cache miss). However, in case of server cache hit, even for D=128, the latency hardly exceeds 1s. After the basis is loaded to mobile device, the approximate solution for the evaluated set of discretization parameters is obtained almost immediately.

7.2 Future work

This master thesis focused on the basic idea of running different simulations, depending on the recognized environmental context and adapting their quality according to user demands. In order to further improve the overall user experience and better optimize the utilization of the mobile device resources, more advanced techniques could be examined.

Bandwidth quality monitoring

Presented approach detects only permanent network failures, however, for more efficient execution strategy selection, the quality of the connection in terms of bandwidth should be analyzed. One of the approaches is to predict the network quality and the execution time on the mobile device to design a scheduler for dynamic switching between the execution strategies as described in [DDR15]. Utilization of this technique will lead to improved latency for answering user queries by loading the reduced basis during the phase of stable connection and switching to mobile computation in case of low bandwidth, when it outperforms the basic strategy, while waiting for the response from the server.

Advanced object recognition

Because of somewhat limited capabilities of the current augmented reality devices for detailed object and surface recognition, fiducial markers were used for real-world tags detection (see Section 2.1.3). However, more advanced computer vision techniques exist to recognize arbitrary objects of interest. They can be based on the local object features detection, such as scale-invariant feature transform (SIFT), speeded up robust features (SURF) approaches or using the neural networks as discussed in [RKS16]. Another interesting approach for 3D object modeling and recognition, namely keypoints based

surface representation (KSR), is presented in [SBB17]. After the offline training phase of 3D models library construction is finished, the KSR for the given scene and predefined model are correlated to identify the real-world object.

IoT infrustructure utilization

Current work can be enhanced by monitoring environmental sensor data from all available sources. Context of the system can be determined through communication with sensors, available in the field, as well as coming from remote devices through the intermediate node (e.g. remote server for receiving and collecting sensor readings). Usage of larger input parameters set will allow to better estimate the suitable training data for the RBM and accurately calculate the up-to-date values of simulation parameters.

Bibliography

[AM04] F.-e. Ababsa, M. Mallem. “Robust Camera Pose Estimation Using 2D Fiducials Tracking for Real-time Augmented Reality Systems.” In: VRCAI ’04 (2004), pp. 431–435. DOI: 10.1145/1044588.1044682. URL: http://doi.acm.org/10.1145/1044588.1044682 (cit. on pp. 20, 21).

[BDR14] F. Berg, F. Dürr, K. Rothermel. “Optimal Predictive Code Offloading.” In: *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. MOBIQUITOUS ’14. London, United Kingdom: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014, pp. 1–10. ISBN: 978-1-63190-039-6. DOI: 10.4108/icst.mobiquitous.2014.258023. URL: http://dx.doi.org/10.4108/icst.mobiquitous.2014.258023 (cit. on p. 27).

[BDR15] F. Berg, F. Dürr, K. Rothermel. “Increasing the efficiency of code offloading through remote-side caching.” In: *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. Oct. 2015, pp. 573–580. DOI: 10.1109/WiMOB.2015.7348013 (cit. on p. 28).

[CBC+10] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl. “MAUI: Making Smartphones Last Longer with Code Offload.” In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’10. San Francisco, California, USA: ACM, 2010, pp. 49–62. ISBN: 978-1-60558-985-5. DOI: 10.1145/1814433.1814441. URL: http://doi.acm.org/10.1145/1814433.1814441 (cit. on pp. 26, 27).

[CFA+10] J. Carmigniani, B. Furht, M. Anisetti, P. Ceravolo, E. Damiani, M. Ivkovic. “Augmented reality technologies, systems and applications.” In: 51 (Dec. 2010), pp. 341–377 (cit. on pp. 16–19).

[DDR15] C. Dibak, F. Dürr, K. Rothermel. “Numerical Analysis of Complex Physical Systems on Networked Mobile Devices.” In: *Proceedings of the 12th IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS)* (Oct. 2015) (cit. on pp. 12, 48, 79).

[DDR17] C. Dibak, F. Dürr, K. Rothermel. “Demo: Server-Assisted Interactive Mobile Simulations for Pervasive Applications.” In: *Proceedings of the 15th IEEE International Conference on Pervasive Computing and Communications (PerCom 2017)* (Mar. 2017) (cit. on pp. 12, 55).

[DK14] C. Dibak, B. Koldehofe. “Towards Quality-aware Simulations on Mobile Devices.” In: *Proceedings of the 44. Jahrestagung der Gesellschaft fur Informatik e.V. (INFORMATIK)* (Sept. 2014) (cit. on p. 11).

[DSD+17] C. Dibak, A. Schmidt, F. Dürr, B. Haasdonk, K. Rothermel. “Server-assisted interactive mobile simulations for pervasive applications.” In: *Proceedings of the 15th IEEE International Conference on Pervasive Computing and Communications (PerCom 2017)* (Mar. 2017) (cit. on pp. 12–14, 24, 26, 28, 29, 50, 55).

[EMP+17] G. Evans, J. Miller, M. I. Pena, A. MacAllister, E. Winer. “Evaluating the Microsoft HoloLens through an augmented reality assembly application.” In: *Proc.SPIE* 10197 (2017), pp. 10197–16. DOI: 10.1117/12.2262626. URL: http://dx.doi.org/10.1117/12.2262626 (cit. on p. 16).

[GBD+16] M. Garon, P. O. Boulet, J. P. Doironz, L. Beaulieu, J. F. Lalonde. “Real-Time High Resolution 3D Data on the HoloLens.” In: (Sept. 2016), pp. 189–191. DOI: 10.1109/ISMAR-Adjunct.2016.0073 (cit. on p. 18).

[GWB16] Y. F. Gao, H. Y. Wang, X. N. Bian. “Marker tracking for video-based augmented reality.” In: 2 (July 2016), pp. 928–932. DOI: 10.1109/ICMLC.2016.7873011 (cit. on p. 20).

[Haa16] B. Haasdonk. “Chapter 2: Reduced Basis Methods for Parametrized PDEs—A Tutorial Introduction for Stationary and Instationary Problems.” In: *Model Reduction and Approximation: Theory and Algorithms*. Mar. 2016, pp. 65–136. ISBN: 978-1-61197-481-2. DOI: 10.1137/1.9781611974829.ch2. eprint: http://locus.siam.org/doi/pdf/10.1137/1.9781611974829.ch2. URL: http://locus.siam.org/doi/abs/10.1137/1.9781611974829.ch2 (cit. on pp. 13, 24, 25).

[HO11] B. Haasdonk, M. Ohlberger. “Efficient reduced models and a posteriori error estimation for parametrized dynamical systems by offline/online decomposition.” In: *Mathematical and Computer Modelling of Dynamical Systems* 17.2 (2011), pp. 145–161. DOI: 10.1080/13873954.2010.514703. eprint: http://dx.doi.org/10.1080/13873954.2010.514703. URL: http://dx.doi.org/10.1080/13873954.2010.514703 (cit. on p. 26).

[KPS10] J. Köhler, A. Pagani, D. Stricker. “Detection and Identification Techniques for Markers Used in Computer Vision.” In: 19 (Jan. 2010), pp. 36–44 (cit. on pp. 19, 68).

[KT17] T. Kajishima, K. Taira. “Finite-Difference Discretization of the Advection-Diffusion Equation.” In: *Computational Fluid Dynamics: Incompressible Turbulent Flows*. Cham: Springer International Publishing, 2017, pp. 23–72. ISBN: 978-3-319-45304-0. DOI: 10.1007/978-3-319-45304-0_2. URL: https://doi.org/10.1007/978-3-319-45304-0_2 (cit. on pp. 14, 23).

[Lan03] H. P. Langtangen. *Computational Partial Differential Equations: Numerical Methods and Diffpack Programming*. 2nd ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003. ISBN: 354043416X (cit. on p. 22).

[LeV02] R. J. LeVeque. *Finite-Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002 (cit. on p. 23).

[OXM02] C. B. Owen, F. Xiao, P. Middlin. “What is the best fiducial?” In: (2002), p. 8. DOI: 10.1109/ART.2002.1107021 (cit. on pp. 19, 20, 69).

[QAKN17] L. Qian, E. Azimi, P. Kazanzides, N. Navab. “Comprehensive Tracker Based Display Calibration for Holographic Optical See-Through Head-Mounted Display.” In: (Mar. 2017) (cit. on pp. 42, 68).

[Qia17] L. Qian. *ARToolKitUWP-Unity*. 2017. URL: https://github.com/qian256/HoloLensARToolKit (cit. on p. 57).

[RKS16] R. Rani, R. Kumar, A. P. Singh. “A Comparative Study of Object Recognition Techniques.” In: *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*. Jan. 2016, pp. 151–156. DOI: 10.1109/ISMS.2016.43 (cit. on p. 79).

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003. ISBN: 0898715342 (cit. on p. 22).

[SBB17] S. A. A. Shah, M. Bennamoun, F. Boussaid. “Keypoints-based surface representation for 3D modeling and 3D object recognition.” In: *Pattern Recognition* 64.Supplement C (2017), pp. 29–38. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2016.10.028. URL: http://www.sciencedirect.com/science/article/pii/S0031320316303429 (cit. on p. 80).

[Sil12] S. Siltanen. *Theory and applications of marker based augmented reality*. Jan. 2012 (cit. on pp. 17, 19).

[VPRP03] K. Veroy, C. Prud’Homme, D. V. Rovas, A. T. Patera. “A Posteriori Error Bounds for Reduced-Basis Approximation of Parametrized Noncoercive and Nonlinear Elliptic Partial Differential Equations.” In: *16th AIAA Computational Fluid Dynamics Conference*. Orlando, United States, June 2003. URL: https://hal.archives-ouvertes.fr/hal-01219051 (cit. on p. 26).

[Wik17a] Wikipedia. *Euclidean distance — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-June-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Euclidean_distance&oldid=801348305 (cit. on p. 18).

[Wik17b] Wikipedia. *LU decomposition — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=LU_decomposition&oldid=798576639 (cit. on p. 51).

All links were last followed on October 17, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature