2020 Annual Report of the Chair of Computer Science 2 (Programming Systems)

1 Staff

Michael Baer, M. Sc., Dipl.-Inf. Thorsten Blaß (until September 30, 2020), Julian Brandner, M. Sc. (since June 15, 2020), Tobias Feigl, M. Sc., Hon.-Prof. Dr.-Ing. Bernd Hindel, Florian Jung, M. Sc. (until August 31, 2020), Marius Kamp, M. Sc., Hon.-Prof. Dr.-Ing. Detlef Kips, Patrick Kreutzer, M. Sc., Florian Mayer, M. Sc., Dipl.-Inf. Daniela Novac, Dr.-Ing. Norbert Oster, Akad. ORat, Prof. Dr. Michael Philippsen (Ordinarius), Prof. em. Dr. Hans Jürgen Schneider (Emeritus), Bastian Söllmann (IT-support, until June 30, 2020), Margit Zenk (Secretary).

Guests and external teaching staff: Dr.-Ing. Josef Adersberger, Veronika Dashuber, M. Sc., Dr.-Ing. Martin Jung, Florian Lautenschlager, M. Sc., Dr.-Ing. Christopher Mutschler, Dr.-Ing. Klaudia Dussa-Zieger.

2 Overview

We develop scientific solutions for software engineers in industry who work on **parallel software** for multicores and for distributed or embedded systems made thereof. We take a **code-centric approach**, construct operational **prototypes**, and **evaluate** them both quantitatively and qualitatively. Corner stones of our field of research:
(a) We work on **programming models** for **heterogeneous** parallelism, from which we then generate portable and efficient code for multicores, GPUs, accelerators, mobile devices, FPGAs, etc.
(b) We help parallelize software for multicores. Our tools analyze code repositories and help developers in **migrating** and **refactoring** projects.
(c) We analyze code. Our **code analysis tools** are fast, interactive, incremental and sometimes work in parallel themselves. They not only detect race conditions, conflicting accesses to resources, etc. The resulting suggestions on how to improve the code also show up in the IDE where they matter.
(d) We **test** parallel code and **diagnose** the root causes of problems. Our tools generate test data, track down erratic runtime behavior, and prevent **authenticity attacks**.

3 Research projects

3.1 AnaCoRe – *Analysis of Code Repositories*

Software developers often modify their projects in a similar or repetitive way. The reasons for these changes include the adoption of a changed interface to a library, the correction of mistakes in functionally similar components, or the parallelization of sequential parts of a program. If developers have to perform the necessary changes on their own, the modifications can easily introduce errors, for example due to a missed change location. Therefore, an automatic technique is desireable that identifies similar changes and uses this knowledge to support developers with further modifications.

Extraction of Code-Changes
In 2017, we developed a new code recommendation tool called ARES (Accurate REcommendation System). It creates more accurate recommendation compared to previous tools as its algorithms take care

of code movements during pattern and recommendation creation. The foundation of ARES lies in the comparison of two versions of the same program. It extracts the changes between the two versions and creates patterns based on the changed methods. ARES uses these patterns to suggest similar changes for the source code of different programs automatically.
The extraction of code changes is based on trees. In 2016 we developed (and visibly published) a new tree-based algorithm (MTDIFF) that improves the accuracy of the change extraction.

Symbolic Execution of Code-Fragments
In 2014 we developed a new symbolic code execution engine called SYFEX to determine the behavioral similarity of two code fragments. In this way we aim to improve the quality of the recommendations. Depending on the number and the generality of the patterns in the database, it is possible that without the new engine SIFE generates some unfitting recommendations. To present only the fitting recommendations to the developers, we compare the summary of the semantics/behavior of the recommendation with summary of the semantics/behavior of the database pattern. If both differ too severely, our tool drops the recommendation from the results. The distinctive features of SYFEX are its applicability to isolated code fragments and its automatic configuration that does not require any human interaction.
In 2015 SYFEX was refined and applied to code fragments from the repositories of different software projects. In 2016 we investigated to which extend SYFEX can be used to gauge the semantic similarity of submissions for a programming contest. In 2017 and 2018 we optimized the implementation of SYFEX. We also began collecting a data set of semantically similar methods from open source repositories. We published this data set in 2019.

Detection of Semantically Similar Code Fragments
SYFEX computes the semantic similarity of two code fragments. Therefore, it allows to identify pairs or groups of semantically similar code fragments (semantic clones). However, the high runtime of SYFEX (and similar tools) limit their applicability to larger software projects. In 2016, we started the development of a technique to accelerate the detection of semantically similar code fragments. The technique is based on so-called base comparators that compare two code fragments using a single criterion (e.g., the number of used control structures or the structure of the control flow graph) and that have a low runtime. These base comparators can be combined to form a hierarchy of comparators. To compute the semantic similarity of two code fragments as accurately as possible, we use genetic programming to search for hierarchies that approximate the similarity values as reported by SYFEX for a number of pairs of code fragments. A prototype implementation confirmed that the method is capable of detecting pairs of semantically similar code fragments.
We further improved the implementation of this approach in 2017 and 2018. Additionally, we focused on evaluating the approach with pairs of methods from software repositories and from programming exercises. Moreover, we created a data set of semantically similar methods from open-source software repositories that we published in 2019.
Techniques for symbolic execution rely on algorithms to detect the satisfiability of logic/mathematic expressions. These are used to detect whether an execution path in a program is feasible. The algorithms often use a large amount of the total computation time. To improve the speed of this satisfiability check, in the years 2019 and 2020 we experimented with a technique to replace complicated expressions with simpler expressions that have the same meaning. These simpler expressions result from the application of program synthesis. In 2020 we augmented the program synthesis with a novel approach to detect beforehand if some operations can form an expression with the same meaning as a more complicated expression.

Semantic Code Search
The functionality that has to be implemented during the development of a software product is often already available as part of program libraries. It is often advisable to reuse such an implementation instead of rewriting it, for example to reduce the effort for developing and testing the code.

To reuse an implementation that fits the purpose, developers have to find it first. To this end developers already use code search engines on a regular basis. State-of-the-art search engines work on a syntactic level, i.e., the user specifies some keywords or names of variables and methods that should be searched for. However, current approaches do not consider the semantics of the code that the user seeks. As a consequence, relevant but syntactically different implementations often remain undetected ("false negatives") or the results include syntactically similar but semantically irrelevant implementations ("false positives"). The search for code fragments on a semantic level is the subject of current research.
In 2017 we began the development of a new method for semantic code search. The user specifies the desired functionality in terms of input/output examples. A function synthesis algorithm from the literature is then used to create a method that implements the specified functionality as accurately as possible. Using our approach to detect similar code fragments, this synthesized method is then compared to the methods of program libraries to find semantically similar implementations. These implementations are then presented as search results to the user. A first evaluation of our prototypical implementation shows the feasibility and practicability of the approach.

Clustering of Similar Code-Changes
To create generalized change patterns, it is necessary that the set of extracted code changes is split into subsets of changes that are similar to each other. In 2015 this detection of similar code changes was improved and resulted in a new tool, called C3. We developed and evaluated different metrics for a pairwise similarity comparison of the extracted code changes. Subsequently, we evaluated different clustering algorithms known from the literature and implemented new heuristics to automatically choose the respective parameters to replace the previous naive approach for the detection of similar code changes. This clearly improved the results compared to the previous approach, i.e., C3's new techniques detect more groups of similar changes that can be processed by SIFE to generate recommendations.
The aim of the second improvement is to automatically refine the resulting groups of similar code changes. For this purpose we evaluated several machine learning algorithms for outlier detection to remove those code changes that have been spuriously assigned to a group.
In 2016 we implemented a new similarity metric for the comparison of two code changes that essentially considers the textual difference between the changes (as generated, for example, by the Unix tool 'diff'). We published both a paper on C3 and the dataset (consisting of groups of similar changes) that we generated for the evaluation of our tool under an open-source license, see https://github.com/FAU-Inf2/cthree. This dataset can be used as a reference or as input data for future research. In addition, we prototypically extended C3 by techniques for an incremental similarity computation and clustering. This allows us to reuse results from previous runs and to only perform the absolutely necessary work whenever new code changes are added to a software archive.

3.2 AuDeRace – *Automatic Detection of Race-Conditions*

Large software projects with hundreds of developers are difficult to review and often contain many bugs. Automatic tests are a well established technique to test sequential and deterministic software. They test the whole project (system test) or each module by itself (unit test). However, recent software contains more and more parallelism. This introduces several new bug patterns, like deadlocks and concurrent memory accesses, that are harder or even impossible to be detected reliably using conventional test methods. Whether the faulty behavior actually shows at runtime depends on the concrete scheduling of the threads which is indeterministic and varies between individual executions depending on the underlaying system. Due to this unpredictable behavior such bugs do not necessarily manifest in an arbitrary test run or may never arise in the testing environment at all. As a result, conventional tests are not well suited for modern, concurrent software.
With the project AuDeRace, we develop methods to efficiently and reliably detect concurrent bugs while

keeping the additional effort for developers as low as possible. In an initial approach we define a testing framework that allows the specification of a scheduling plan to regain deterministic execution. However, a major problem still remains: The developer has to identify and implement well suited test cases that cover the potential fault in the program and execute them in a special deterministic way in order to trigger the failure. Especially in the context of concurrency, it is difficult to visualize the behavior of a program and identify the problematic parts. To overcome this, the critical parts shall automatically be narrowed down before even writing dedicated test cases. Existing approaches and tools for this purpose generate too many false positives or the analysis is very time consuming, making their application to real world code prohibitive. The goal of this project is to generate less false positives and increase the analysis speed by combining existing static and dynamic analysis. This allows for the efficient use in not only small example codes but also large and complex software projects.

In 2016 existing approaches were studied regarding their usability as a starting basis for our project. The most promising method uses model checking and predefined assertions to construct thread schedules that trigger the faulty behavior. However, the approach is currently infeasible for larger projects because only very small codes could be analyzed in reasonable time. Therefore, we focused on automatically detecting and removing statements that are unrelated to the parallelism respectively to the potentially faulty code parts in order to decrease the execution time of the preliminary static analysis.

In 2017 the work on automatically reducing programs to speed up furher analysis was continued. Furthermore, we evaluated whether the concept of mutation testing can be applied to parallel software as well. The results indicate that this extension is indeed possible to rate tests qualitativly. However, to complete the analysis for larger programs in reasonable time, a few heuristics need to be applied during the process.

In 2018 the focus moved to a deterministic execution of test cases. A concept to reproduce results during the execution was developed: In addition to the test case, a schedule specifies the dynamic behaviour of the threads. Instrumenting the code at previously marked positions and other relevant byte code instructions allows a separate control thread to enforce the schedule. When modifying the source code, the marked positions in the code need to be updated as well to keep them consistent with the test cases. A merging technique similar to the ones used in version control systems shall be used to automatically update the positions.

Up to 2019, this project was a contribution of the Chair of Computer Science 2 (Programming Systems) to the IZ ESI (Embedded Systems Initiative, http://www.esi.fau.de/). In this context, several improvements for the quality of concurrent software were analyzed. The take-away result was that different approaches are applicable and required, but they also often suffer from long analysis times.

Beyond the ESI project, we improved the usefulness of mutation testing by developing a tool for equivalence detection and test case generation. A submitted paper got accepted.

In 2020, we studied and evaluated approaches to detect external race conditions. Whereas in classic race conditions, several threads of the analyzed software fail to work together properly, in external race conditions, software interacts with independent, unknown components. Examples are other programs, the operating system, or even malicious code written by attackers who interferes with the analyzed software.

3.3 AutoCompTest – *Automatic Testing of Compilers*

Compilers for programming languages are very complex applications and their correctness is crucial: If a compiler is erroneous (i.e., if its behavior deviates from that defined by the language specification), it may generate wrong code or crash with an error message. Often, such errors are hard to detect or circumvent. Thus, users typically demand a bug-free compiler implementation.

Unfortunately, research studies and online bug databases suggest that probably no real compiler is bug-free. Several research works therefore aim to improve the quality of compilers. Since the formal verification (i.e., a proof of a compiler's correctness) is often prohibited in practice, most of the recent works focus on techniques for extensively testing compilers in an automated way. For this purpose, the compiler under test is usually fed with a test program and its behavior (or that of the generated program) is checked: If the actual behavior does not match the expectation (e.g., if the compiler crashes when fed with a valid test program), a compiler bug has been found. If this testing process is to be carried out in a fully automated way, two main challenges arise:

- Where do the test programs come from that are fed into the compiler?
- What is the expected behavior of the compiler or its output program? How can one determine if the compiler worked correctly?

While the scientific literature proposes several approaches for dealing with the second challenge (which are also already established in practice), the automatic generation of random test programs still remains a challenge. If all parts of a compiler should be tested, the test programs have to conform to all rules of the respective programming language, i.e., they have to be syntactically and semantically correct (and thus compilable). Due to the large number of rules of "real" programming languages, the generation of such compilable programs is a non-trivial task. This is further complicated by the fact that the program generation has to be as efficient as possible: Research suggests that the efficiency of such an approach significantly impacts its effectivity – in a practical scenario, a tool can only be used for detecting compiler bugs if it can generate many (and large) programs in short time.

The lack of an appropriate test program generator and the high costs associated with the development of such a tool often prevent the automatic testing of compilers in practice. Our research project therefore aims to reduce the effort for users to implement efficient program generators.

In 2018, we started the development of such a tool. As input, it requires a specification of a programming language's syntactic and semantic rules by means of an abstract attribute grammar. Such a grammar allows for a short notation of the rules on a high level of abstraction. Our newly devised algorithm then generates test programs that conform to all of the specified rules. It uses several novel technical ideas to reduce its expected runtime. This way, it can generate large sets of test programs in acceptable time, even when executed on a standard desktop computer. A first evaluation of our approach did not only show that it is efficient and effective, but also that it is versatile. Our approach detected several bugs in the C compilers gcc and clang (and achieved a bug detection rate which is comparable to that of a state-of-the-art C program generator from the literature) as well as multiple bugs in different SMT solvers. Some of the bugs that we detected were previously unknown to the respective developers.

In 2019, we implemented additional features for the definition of language specifications and improved the efficiency of our program generator. These two contributions considerably increased the throughput of our tool. By developing additional language specifications, we were also able to uncover bugs in compilers for the programming languages Lua and SQL. The results of our work led to a publication that we submitted at the end of 2019 (and which has been accepted by now). Besides the work on our program generator, we also began working on a test case reduction technique. It reduces the size of a randomly generated test program that triggers a compiler bug since this eases the search for the bug's root cause.

In 2020, we focussed on language-agnostic techniques for the automatic reduction of test programs. The scientific literature has proposed different reduction techniques, but since there is no conclusive comparison of these techniques yet, it is still unclear how efficient and effective the proposed techniques really are. We identified two main reasons for this, which also hamper the development and evaluation of new techniques. Firstly, the available implementations of the proposed reduction techniques use different implementation languages, program representations and input grammars. Therefore, a fair comparison of the proposed techniques is almost impossible with the available implementations. Secondly, there is no collection of (still unreduced) test programs that can be used for the evaluation of reduction techniques. As a result, the published techniques have only been evaluated with few test programs each, which compromises the significance of the published results. Furthermore, since some techniques have only been evaluated with test programs in a single programming language, it is still unclear how well these techniques generalize to other programming languages (i.e., how language-agnostic they really are). To close these gaps, we initiated the development of a framework that contains implementations of the most important reduction techniques and that enables a fair comparison of these techniques. In addition, we also started to work on a benchmark that already contains about 300 test programs in C and SMT-LIB 2 that trigger about 100 different bugs in real compilers. This benchmark not only enables conclusive comparisons of reduction techniques but also reduces the work for the evaluation of future techniques. Some first experiments already exposed that there is no reduction technique yet that performs best in all cases.

In this year, we also investigated how the random program generator that has been developed in the context of this research project can be extended to not only detect functional bugs but also performance problems in compilers. A new technique has been developed within a thesis that first generates a set of random test programs and then applies an optimization technique to gradually mutate these programs. The goal is to find programs for which the compiler under test has a considerably higher runtime than a reference implementation. First experiments have shown that this approach can indeed detect performance problems in compilers.

3.4 Holoware – *Cooperative Exploration and Analysis of Software in a Virtual/Augmented Reality Appliance*

Understanding software has a large share in the programming efforts of a software systems, up to 30% in development projects and up to 80% in maintenance projects. Therefore, an efficient and effective way for comprehending software is neccessary in a modern software engineering workplace. Three-dimensional software visualization already boosts comprehension and efficiency, so utilization of latest virtual reality techniques seems natural.
Within the scope of the Holoware project, we create an environment to **cooperatively explore and analyze a software project using virtual/augmented reality techniques as well as artificial intelligence algorithms**.
The software project in question is being visualized in said virtual reality, such that multiple participants can simultaneously explore and analyze the software. They can cooperate by communicating about their findings. Different participants benefit from different perspectives on the software, which is augmented by domain specific additional information. This provides them with an intuitive access to the structure and behaviour of the software.
Various use cases are possible, for example the cooperative analysis of a run time anomaly in a team of domain experts. The domain experts can see the same static structure, augmented with domain specific and detailed information. In the VR environment, they can share their findings and cooperate using their different expertises.
In addition, the static and dynamic properties of the software system are analyzed. Static properties include source code, static call relationships or metrics such as LoC, cyclomatic complexity, etc. Dynamic properties can be grouped into logs, traces, runtime metrics, or configurations that are read in at runtime. The challenge lies in aggregating, analyzing, and correlating this wealth of information.
An anomaly and significance detection is developed that automatically detects both structural and runtime anomalies. In addition, a prediction system is set up to make statements about component health.

This makes it possible, for example, to predict which components are at risk of failing in the near future. Previously, the log entries were added to the traces, creating a detailed picture of the dynamic call relationships. These dynamic relationships are mapped to the static call graph because they describe calls that do not result from the static analysis (for example, REST calls across several distributed components).

In 2018, the following significant contributions have been made:

- Development of a functional VR visualization prototype for demonstration and research purposes.
- Mapping between dynamic run time data and static structure (required by later analysis and visualization tasks).
- First draft and implementation of the trace anomaly detection by an unsupervised learning procedure. Evaluation and further improvements will follow in the coming months.

In 2019 we achieved the following improvements:

- Extension of the prototype to display dynamic software behaviour.
- Cooperative (remote-)usability of the visualization prototype.
- Interpretation of commit messages for anomaly detection.
- Clustering system calls according to use cases.

Our paper "Towards Collaborative and Dynamic Software Visualization in VR" has been accepted for publication at the International Conference on Computer Graphics Theory and Applications (VISIGRAPP) 2020. It presents the efficiency of our prototype at increasing the software understanding process.

In 2020, our paper "A Layered Software City for Dependency Visualization" was accepted at the International Conference on Computer Graphics Theory and Applications (VISIGRAPP) 2021 and later received the "Best Paper Award". We demonstrated that our Layered Layout for Software Cities simplifies the analysis of software architecture and outperforms the standard layout by far. We successfully concluded the research project with a final prototype and the resulting publications.

3.5 ORKA-HPC – *OpenMP for reconfigurable heterogenous architectures*

High-Performance Computing (HPC) is an important component of Europe's capacity for innovation and it is also seen as a building block of the digitization of the European industry. Reconfigurable technologies such as Field Programmable Gate Array (FPGA) modules are gaining in importance due to their energy efficiency, performance, and flexibility.
There is also a trend towards heterogeneous systems with accelerators utilizing FPGAs. The great flexibility of FPGAs allows for a large class of HPC applications to be realized with FPGAs. However, FPGA programming has mainly been reserved for specialists as it is very time consuming. For that reason, the use of FPGAs in areas of scientific HPC is still rare today.
In the HPC environment, there are various programming models for heterogeneous systems offering certain types of accelerators. Common models include OpenCL (http://www.opencl.org), OpenACC (https://www.openacc.org) and OpenMP (https://www.OpenMP.org). These standards, however, are not yet available for the use with FPGAs.

Goals of the ORKA project are:

1. Development of an OpenMP 4.0 compiler targeting heterogeneous computing platforms with FPGA accelerators in order to simplify the usage of such systems.
2. Design and implementation of a source-to-source framework transforming C/C++ code with OpenMP 4.0 directives into executable programs utilizing both the host CPU and an FPGA.

3. Utilization (and improvement) of existing algorithms mapping program code to FPGA hardware.
4. Development of new (possibly heuristic) methods to optimize programs for inherently parallel architectures.

In 2018, the following important contributions were made:

- Development of a source-to-source compiler prototype for the rewriting of OpenMP C source code (cf. goal 2).
- Development of an HLS compiler prototype capable of translating C code into hardware. This prototype later served as starting point for the work towards the goals 3 and 4.
- Development of several experimental FPGA infrastructures for the execution of accelerator cores (necessary for the goals 1 and 2).

In 2019, the following significant contributions were achieved:

- Publication of two peer-reviewed papers: "OpenMP on FPGAs - A Survey" and "OpenMP to FPGA Offloading Prototype using OpenCL SDK".
- Improvement of the source-to-source compiler in order to properly support OpenMP-target-outlining for FPGA targets (incl. smoke tests).
- Completion of the first working ORKA-HPC prototype supporting a complete OpenMP-to-FPGA flow.
- Formulation of a genome for the pragma-based genetic optimization of the high-level synthesis step during the ORKA-HPC flow.
- Extension of the TaPaSCo composer to allow for hardware synchronization primitives inside of TaPaSCo systems.

In 2020, the following significant contributions were achieved:

- Improvement of the Genetic Optimization.
- Engineering of a Docker container for reliable reproduction of results.
- Integration of software components from project partners.
- Development of a plugin architecture for Low-Level-Platforms.
- Implementation and integration of two LLP plugin components.
- Broadening of the accepted subset of OpenMP.
- Enhancement of the test suide.

3.6 ParCAn – *Parallel code analysis on a GPU*

In compiler construction there are analyses that propagate information along the edges of a graph and modify it, until a fix point is reached and the information no longer changes. In this project we built the ParCAn framework to accelerate such analyses by exploiting the massive parallelism of graphic cards.

In 2020, we successfully completed this research project. We demonstrated that parallelizing the particularly cost-intensive data flow analyses can speed-up the compilation process of up to 31%. Thus, our research leads the way towards parallelized compilers that meet the requirements of today's software projects. The importance of this research topic was underlined by a „Best Paper Award" at the renowned „Compiler Construction" conference, see references.

The use of the GPU as the target architecture raised other research-related questions that were also published.

Some analyses store their information in a global data structure that can be modified by all threads simultaneously. Especially the high number of concurrent threads on a GPU demands for efficient synchronization. Thus, as part of the research project we implemented an efficient framework for establishing mutual exclusion, see the LNCS-paper in the references. Previous approaches inevitably resulted in deadlocks when the GPU is fully utilized. Moreover, by using a variant of the inspection-execution paradigm we further improved the efficiency of the framework. Another research topic considered the efficiency of graph structures on GPUs. At its core, ParCAn implements a graph traversal algorithm. The program to be translated is converted into a graph, the control flow graph (CFG), on which the analyses are executed. Due to the large number of parallel accesses, the CFG represents a critical data structure for the performance of ParCAn. For this reason, we conducted an extensive study comparing the performance of graph data structures. We used the results to determine the best possible data structure to represent the CFG. We derived general criteria that allow to make assumptions about the performance of a data structure under certain conditions. Even outside of the context of ParCAN, developers can use these criteria, represented as a decision tree, to choose the most appropriate data structure for their static graph algorithms. The results of the study were presented at the GPGPU workshop, see references.

3.7 RuNN – *Recurrent Neuronal Networks (RNNs) for Real-Time Estimation of Nonlinear Motion Models*

With the growing availability of information about an environment (e.g., the geometry of a gymnasium) and about the objects therein (e.g., athletes in the gymnasium), there is an increasing interest in bringing that information together profitably (so-called information fusion) and in processing that information. For example, one would like to reconstruct physically correct animations (e.g., in virtual reality, VR) of complex and highly dynamic movements (e.g., in sports situations) in real-time. Likewise, e.g., manufacturing plants of the industry, which suffer from unfavorable environmental conditions (e.g., magnetic field interference or missing GPS signal), benefit from, e.g., high-precision goods location. Typically, to describe movements, one uses either poses that describe a "snapshot" of a state of motion (e.g., idle state, stoppage), or a motion model that describes movement over time (e.g., walking or running). In addition, human movements may be identified, detected, and sensed by different sensors (e.g., on the body) and mapped in the form of poses and motion models. Different types of modern sensors (e.g., camera, radio, and inertial sensors) provide information of varying quality.
In principle, with the help of expensive and highly precise measuring instruments, the extraction of the poses and resp. of the motion model, for example, from positions on small tracking areas is possible without errors. Positions, e.g., of human extremities, can describe or be described by poses and motion models. Camera-based sensors deliver the required high-frequency and high-precision reference measurements on small areas. However, as the size of the tracking surface increases, the usability of camera-based systems decreases (due to inaccuracies or occlusion issues). Likewise, on large areas radio and inertial sensors only provide noisy and inaccurate measurements. Although a combination of radio and inertial sensors based on Bayesian filters achieves greater accuracy, it is still inadequate to precisely sense human motion on large areas, e.g., in sports, as human movement changes abruptly and rapidly. Thus, the resulting motion models are inaccurate.
Furthermore, every human movement is highly nonlinear (or unpredictable). We cannot map this nonlinearity correctly with today's motion models. Bayes filters describe these models but these (statistical) methods break down a nonlinear problem into linear subproblems, which in turn cannot physically represent the motion. In addition, current methods produce high latency when they require accuracy.
Due to these three problems (inaccurate position data on large areas, nonlinearity, and latency), today's

methods are unusable, e.g., for sports applications that require short response times. This project aims to counteract these nonlinearities by using machine learning methods. The project includes research on recurrent neural networks (RNN) to create nonlinear motion models. As modern Bayesian filtering methods (e.g., Kalman and Particle filters) and other statistical methods can only describe the linear portions of nonlinear human movements (e.g., the relative position of the head w.r.t. trunk while walking or running) they are thus physically not completely correct.
Therefore, the main goal is to evaluate how machine learning methods can describe complex and nonlinear movements. We therefore examined whether RNNs describe the movements of an object physically correctly and support or replace previous methods. As part of a large-scale parameter study, we simulated physically correct movements and optimized RNN procedures on these simulations. We successfully showed that, with the help of suitable training methods, RNN models can either learn physical relationships or shapes of movement.

This project addresses three key topics:

I. A basic implementation investigates how and why methods of machine learning can be used to determine models of human movement.

In 2018, we first established a deeper understanding of the initial situation and problem definition. With the help of different basic implementations (different motion models) we investigated (1) how different movements (e.g., humans: walk, run, slalom; vehicles: meander, zig-zag) affect measurement inaccuracies of different sensor families, (2) how measurement inaccuracies of different sensor families (e.g., visible orientation errors, audible noise, and deliberated artificial errors) affect human motion, and (3) how different filter methods for error correction (that balance accuracy and latency) affect both motion and sensing. In addition, we showed (4) how measurement inaccuracies (due to the use of current Bayesian filtering techniques) correlate nonlinearly with human posture (e.g., gait apparatus) and predictably affect health (simulator sickness) through machine learning.
We studied methods of machine and deep learning for motion detection (humans: head, body, upper and lower extremity; vehicles: single- and bi-axial) and motion reconstruction (5) based on inertial, camera, and radio sensors, as well as various methods for feature extraction (e.g., SVM, DT, k-NN, VAE, 2D-CNN, 3D-CNN, RNN, LSTM, M/GRU). These were interconnected into different hybrid filter models to enrich extracted features with temporal and context-sensitive motion information, potentially creating more accurate, robust, and close to real-time motion models. In this way, these mechanics learned (6) motion models for multi-axis vehicles (e.g., forklifts) based on inertial, radio, and camera data, which generalize for different environments or tracking surfaces (with varying size, shape, and sensory structure, e.g., magnetic field, multipath, texturing, and illumination). Furthermore (7), we gained a deeper understanding of the effects of non-constant accelerated motion models on radio signals. On the basis of these findings, we trained an LSTM model that predicts different movement speeds and motion forms of a single-axis robot (i.e., Segway) close to real-time and more accurately than conventional methods.
In 2019, we found that these models can also predict human movement (human movement model). We also determined that the LSTM models can either be fully self-sufficient at runtime or integrated as support points into localization estimates, e.g., into Pedestrian Dead Reckoning (PDR) methods.

II. Based on this, we try to find ways to optimize the basic implementation in terms of robustness, latency, and reusability.

In 2018, we used the findings from I. (1-7) to stabilize so-called (1) relative Pedestrian Dead Reckoning (PDR) methods using motion classifiers. These enable a generalization to any environment. A deeper radio signal understanding (2) allowed to learn long-term errors in RNN-based motion models. This improves the position accuracy, stability, and a near real-time prediction. First experiments showed the robustness of the movement models (3) with the help of different real (unknown to the models) movement trajectories for one- and two-axis vehicles. Furthermore, we investigated (4) how hybrid filter

models (e.g., interconnection of feature extractors such as 2D/3D-CNNs and time-series trackers such as RNNs-LSTM) provide more accurate, more stable, and filtered (outlier-corrected) results.
In 2019, we showed that models of the RNN family extrapolate movements into the future so that they compensate for the latency of the processing pipeline and beyond. Furthermore, we examined the explainability, interpretability, and robustness of the models examined here, and their reusability on the human movement.
With the help of a simulator, we generated physically correct movements, e.g., positions of pedestrians, cyclists, cars, and planes. Based on this data, we showed that RNN models can interpolate between different types of movement and can compensate for missing data points, interpret white and random noise as such, and can extrapolate movements. The latter enables processing-specific latency to be compensated and enables human movement to be predicted from radio and inertial data in real time.
Novel RNN architecture. Furthermore, in 2019, we researched a new architecture, or topology, of a neural network, that balances the strengths and weaknesses of flat neural networks (NN) and recurrent networks. We found this optimal NN for determining physically correct movement in a large-scale parameter study. In particular, we also optimized the model architecture and parameters for human-centered localization. These optimal architectures predict human movement far into the future from as little sensor information as possible. The architecture with the least localization error combines two DNNs with an RNN.
Interpretability of models. In 2019, we examined the functionality of this new model. For this purpose, we researched a new process pipeline for the interpretation and explanation of the model. The pipeline uses the mutual information flow and the mutual transfer entropy in combination with various targeted manipulations of the hidden states and suitable visualization techniques to describe the state of the model at any time, both subjectively and objectively. In addition, we adapted a variational auto-encoder (VAE) to better visualize and interpret extracted features of a neural network. We designed and parameterized the VAE such that the reconstruction error of the signal is within the range of the measurement noise and at the same time forced the model to store disentangled features in its latent space. This disentanglement enabled the first subjective statements about the interrelationships of the features that are really necessary to optimally code the channel state of a radio signal.
Compression. In 2019, we discovered a side effect of the VAE that offers the possibility of decentralized preprocessing of the channel information directly on the antenna. This compression then reduces the data traffic, lowers the communication load, and thus increases the number of possible participants in the communication and localization in a closed sensor network.
Influence of the variation of the input information. In 2019, we also examined how changes in the input sequence length of a recurrent neural network affect the learning success and the type of results of the model. We discovered that a longer sequence persuades the model to be a motion model, i.e., to learn the form of movement, while with shorter sequences the model tends to learn physical relationships. The optimal balance between short and long sequences represents the highest accuracy.
We investigated speed estimation using the new method. When used in a PDR model, this increased the position accuracy. An initial work in 2019 has examined in detail which methods are best suited to estimate the speed of human movement from a raw inertial signal. A new process, a combination of a one-dimensional CNN and a BLSTM, has replaced the state of the art.
In 2020, we optimized the architecture of the model,with regard to its prediction accuracy and investigated the effects of a deep fusion of Bayesian and DL methods on the prediction accuracy and robustness.
Optimization. In 2020, we improved the existing CNN and RNN architecture and proposed the fusion of ResNet and BLSTM. We replaced the CNN with a residual network to extract deeper and higher quality features from a continuous data stream. We showed that this architecture entails higher computing costs, but surpasses the accuracy of the state-of-the-art. In addition, the RNN architecture can be scaled down to counter the blurring of the context vector of the LSTM cells with very long input sequences, as the remaining ResNet network offers more qualitative features. **Deep Bayesian Method.** In 2020 we investigated

whether methods of the RNN family can extract certain movement properties from recorded movement data to replace the measurement-, process-, and transition-noise distributions of a Kalman filter (KF). We showed that highly optimized LSTM cells can reconstruct trajectories more robust (low error variance) and more precise (positional accuracy) than an equally highly optimized KF. The deep coupling of LSTM in KF, so-called Deep Bayes, provided the most robust and precise positions and trajectories. This study also showed that methods trained on realistic synthetic data, the Deep Bayesian method, needed the least real data to adapt to a new unknown domain, e.g., unknown motion shapes and velocity distribution.

III. Finally, a demonstration of feasibility shall be tested.

In 2018, a large-scale social science study opened the world's largest virtual dinosaur museum and showed that (1) a pre-selected (application-optimized) model of human movement robustly and accurately (i.e., without a significant impact on simulator sickness) maps human motion, resp. predicts it. We used this as a basis for comparison tests with other models that are human-centered and generalize to different environments.
In 2019, we developed two new live demonstrators that are based on the research results achieved in I and II. (1) A model railway that crosses a landscape with a tunnel at variable speeds. The tunnel represents realistic and typical environmental characteristics that lead to nonlinear multipath propagation of a radio transmitter to be located and ultimately to an incorrectly determined position. This demonstrator shows that the RNN methods researched as part of the research project can localize highly precisely and robustly, both on complex channel impulse responses and on dimensionally reduced response times, and also deliver better results than conventional Kalman filters. (2) We used the second demonstrator to visualize the movement of a person's upper extremity. We recorded human movement using inexpensive inertial sensors attached to both arm joints, classified using machine-based and deep learning, and derived motion parameters. A graphic user interface visualizes the movement and the derived parameters in near real time.
The planned generalizability, e.g., of human-centered models and the applicability of RNN-based methods in different environments, has been demonstrated using (1) and (2). In 2019, we applied the proposed methods in the following applications:
Application: Radio Signal. We classified the channel information of a radio system hierarchically. We translated the localization problem of a Line of Sight (LoS) and Non Line of Sight (NLoS) classifier into a binary problem. Hence, we now can precisely localize a position within a meter, based on individual channel information from a single antenna if the environment provides heterogeneous channel propagation.
Furthermore, we simulated LoS and NLoS channel information and used it to interpolate between different channels. This enables the providers of radio systems to respond to changing or new environments in the channel information a-priori in the simulation. By selectively retraining the models with the simulated knowledge, we enabled more robust models.
Application: Camera and Radio Signal. We have shown how the RNN methods relate to information from other sensor families, e.g., video images, by combining radio and camera systems when training a model, the two sensor information streams merge smoothly, even in the event of occlusion of the camera. This yields a more robust and precise localization of multiple people.
Application: Camera Signal. We used an RNN method to examine the temporal relationships between events in images. In contrast to the previous work, which uses heterogeneous sensor information, this network only uses image information. However, the model uses the image information in such a way that it interprets the images differently: as spatial information, i.e., a single image, and as temporal information, i.e., several images in the input. This splitting implies that individual images can be used as two fictitious virtual sensor information streams to recognize results spatially (features) and to better predict them temporally (temporal relationships). Another work uses camera images to localize the camera itself. For this purpose, we built a new processing pipeline that breaks up the video signal over time and learns

absolute and relative information in different neural networks and merges their outputs into an optimal pose in a fusion network.
Application: EEG Signal. In a cooperation project we applied the researched methods to other sensor data. We recorded beta- and gamma-waves of the human brain in different emotional states. When used to train an RNN, it correctly predicted the emotions of a test person in 90% of all cases from raw EEG data.
Application: Simulator Sickness. We have shown how the visualization in VR affects human perception and movement anomalies, resp. simulator sickness, and how the neural networks researched here can be used to predict the effects.
In 2020, we developed a new live demonstrator based on the research results achieved in II.
Application: Gait reconstruction in VR. In 2020, we used the existing CNN-RNN model to predict human movement, namely gait cycle and gait phases, using sensor data from a head-mounted inertial sensor to visualize a virtual avatar in VR in real time. We showed that the DL model has significantly lower latencies than the state-of-the-art, since it can recognize gait phases earlier and predict future ones more precisely. However, this is at the expense of the required computing effort and thus the required hardware.

3.8 SoftWater – *Software Watermarking*

Software watermarking means hiding selected features in code, in order to identify it or prove its authenticity. This is useful for fighting software piracy, but also for checking the correct distribution of open-source software (like for instance projects under the GNU license). The previously proposed methods assume that the watermark can be introduced at the time of software development, and require the understanding and input of the author for the embedding process. The goal of our research is the development of a watermarking framework that automates this process by introducing the watermark during the compilation phase into newly developed or even into legacy code. As a first approach we studied a method that is based on symbolic execution and function synthesis.
In 2018, two bachelor theses analyzed two methods of symbolic execution and function synthesis in order to determine the most appropriate one for our approach.
In 2019, we investigated the idea to use concolic execution in the context of the LLVM compiler infrastructure in order to hide a watermark in an unused register. Using a modified register allocation, one register can be reserved for storing the watermark.
In 2020, we extended the framework for automatically embedding software watermarks into source code (based on the LLVM compiler infrastructure) with further dynamic methods. The newly introduced methods rely on replacing/hiding jump targets and on call graph modifications.

3.9 CS4MINTS – *Promote computer science as the basis for successful STEM studies along the entire education chain.*

Progressive digitalization is changing not only the job market but also the educational landscape. With funding from the DigitalPakt Schule and in detail from the BAYERN DIGITAL II program, serious changes in computer science education are being driven forward, which entail new challenges at the various levels of education.

The CS4MINTS project addresses these challenges along with the educational levels and ties in with measures already launched as part of the MINTerAKTIV project, such as strengthening the encounter of increasing student heterogeneity in the introductory computer science course.

For example, for **promoting gifted students**, the Frühstudium in computer science is actively promoted for girls, and the offer is explicitly expanded. A significant increase in the proportion of women in computer science is to be achieved in the long term through early action against gender-specific stereotypes regarding computer science and an expansion of the training program to include gender-sensitive computer science instruction in all types of schools.

The expansion of the compulsory subject of computer science in all schools also creates a great need for suitable teaching concepts and a strengthening of **teacher training**. For this purpose, a regional network is to be established during the project period to provide university-developed and evaluated teaching ideas for strengthening STEM in the curricular and extra-curricular settings.

In 2020, we began the initial piloting of the design to automate feedback in the introductory programming exercises. For this purpose, the return values of the JUnit tests of students' solutions were analyzed, and possible sources of errors were investigated. The next step is to work out a way to infer programming errors or student misconceptions based on these return values. Finally, these efforts aim to provide the students with automatically generated, competence-oriented feedback available to them after the programming tasks have been submitted (or, if necessary, already during the development phase). The feedback should show where errors occurred in the program code and point out possible causes.

Concerning handling heterogeneity, we have compared the Repetitorium Informatik (RIP) course content with the Bavarian curriculum of different school types in 2020. Subsequently, the content must be adapted so that first-year students from the most diverse educational backgrounds have equal opportunities to identify possible deficits through the Repetitorium and remedy them. Besides, a daily programming consultation hour was set up for the first time during the Repetitorium in the winter term 2020. Here, participants were able to ask questions and receive feedback on the assignments.

4 Teaching

The Chair for Programming Systems teaches the compulsory modules *Algorithms and Data Structures (AuD)* during the winter term and *Parallel and Functional Programming (PFP)* during the summer term. Since those modules are offered to many other degree programs from different faculties (especially Business & Information Systems resp. International Information Systems, Information and Communication Technology, Mathematics and many more), the number of attending students and examinations almost reached the high scores of the previous terms: 645 students attended AuD during the winter term 2019/20 and 422 students attended PFP during the summer term 2020 – the number of examinations hit 503 in AuD resp. 348 in PFP. The Chair offers different modules on *Compiler Construction*, *Clustercomputing* and *Testing of Software Systems* to students specializing on programming systems. The tutorials *Hallo Welt! für Fortgeschrittene* and *Machine Learning* were also fully booked within short time.
The Chair for Programming Systems supervised three master's thesis and three bachelor's thesis in total during the period under report.

ICPC – *International Collegiate Programming Contest an der FAU*: Since 1977 the International Collegiate Programming Contest (ICPC) takes place every year. Teams of three students try to solve about 13 programming problems within five hours. What makes this task even harder, is that there is only one computer available per team.
In 2020 we conducted two local contests in Erlangen. In the winter semester there was a team contest with teams consisting of at most three students. The main goal of this contest was to interest new students in the contests. We had 23 FAU teams. Before the second contest, as in previous years, in the summer term the seminar "Hello World - Programming for the Advanced" served to prepare students from different disciplines in algorithms and contest problems. In the nationwide contest of the summer term,

that as an exception was held in November in an online format, we selected the students that represent the FAU at the NWERC 2020 (postponed to March 2021). 15 teams with students of computer science, computational engineering, mathematics as well as information and communication technology took the challenge.

5 Publications 2020

[1] Michael Baer, Norbert Oster, and Michael Philippsen. MutantDistiller: Using Symbolic Execution for Automatic Detection of Equivalent Mutants and Generation of Mutant Killing Tests. In *2020 IEEE Intl. Conf. Software Testing, Verification and Validation Workshops (ICSTW)*, pages 294–303, Porto, Portugal, Oct. 2020. IEEE Xplore. doi:10.1109/ICSTW50294.2020.00055.

[2] Tobias Feigl, Sebastian Kram, Philipp Woller, Ramiz H. Siddiqui, Michael Philippsen, and Christopher Mutschler. RNN-aided Human Velocity Estimation from a Single IMU. *Sensors*, 13:1–31, 2020. doi:10.3390/s20133656.

[3] Tobias Feigl, Andreas Porada, Steve Steiner, Christoffer Löffler, Christopher Mutschler, and Michael Philippsen. Localization Limitations of ARCore, ARKit, and Hololens in Dynamic Large-Scale Industry Environments. In Jose Braz Kadi Bouatouch, A. Augusto Sousa, editor, *Proc. 15th Intl. Joint Conf. Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: GRAPP*, pages 307–318, Porto, Portugal, Feb. 2020. SciTePress. doi:10.5220/0008989903070318.

[4] Florian Jung, Veronika Dashuber, and Michael Philippsen. Towards Collaborative and Dynamic Software Visualization in VR. In Jose Braz Andreas Kerren, Christophe Hurter, editor, *Proc. 15th Intl. Joint Conf. Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP*, pages 149 – 156, Porto, Portugal, Feb. 2020. SciTePress. doi:10.5220/0008945201490156.

[5] Patrick Kreutzer, Stefan Kraus, and Michael Philippsen. Language-Agnostic Generation of Compilable Test Programs. In *Proc. Intl. Conf. Software Testing, Verification and Validation (ICST 2020)*, pages 39–50, Porto, Portugal, Oct. 2020. IEEE Xplore. doi:10.1109/ICST46399.2020.00015.

[6] Felix Ott, Tobias Feigl, Christoffer Löffler, and Christopher Mutschler. ViPR: Visual-Odometry-aided Pose Regression for 6DoF Camera Localization. In Computer Vision Foundation (CVF), editor, *Joint Workshop on Long-Term Visual Localization, Visual Odometry and Geometric and Learning-based SLAM*, pages 42–43, June 2020.

[7] Anes Redzepagic, Christoffer Löffler, Tobias Feigl, and Christopher Mutschler. A Sense of Quality for Augmented Reality Assisted Process Guidance. In *Proc. of the 2020 IEEE Intl. Symp. on Mixed and Augmented Reality (ISMAR)*, pages 1–6, Nov. 2020.