Jahresbericht 2021 des Lehrstuhls für Informatik 2 (Programmiersysteme)

1 Mitarbeiterinnen und Mitarbeiter

Michael Baer, M. Sc. (bis 30.09.2021), Julian Brandner, M. Sc., Hon.-Prof. Dr.-Ing. Bernd Hindel, Marius Kamp, M. Sc. (bis 30.06.2021), 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), Dipl.-Ing. Frank Deserno (seit 01.06.2021), Margit Zenk (Sekretariat).

Gäste und externes Lehrpersonal am Lehrstuhl: Dr.-Ing. Josef Adersberger, Veronika Dashuber, M. Sc., Tobias Feigl, M. Sc., Dr.-Ing. Martin Jung (Lehrbeauftragter), Florian Lautenschlager, M. Sc., Dr.-Ing. Christopher Mutschler, Dr.-Ing. Klaudia Dussa-Zieger (Lehrbeauftragte).

2 Überblick

Wir liefern ingenieurwissenschaftliche Antworten für Software-Ingenieure, die **parallele Software** im industriellen Rahmen für **Multicore-Rechner**, für daraus bestehende verteilte Systeme, sowie für vernetzte eingebettete Systeme entwickeln. Wir arbeiten **Programm-Code-basiert**, erstellen lauffähige **Prototypen** und **evaluieren** diese quantitativ und qualitativ. Eckpunkte unserer Forschungsthemen:
(a) Wir arbeiten an **Programmiermodellen** für **heterogene** Parallelität und erzeugen dafür portablen und effizienten Code für Multicores, GPUs, Acceleratoren, Mobile Geräte, FPGAs u.ä.
(b) Wir unterstützen die **Parallelisierung** von Software für Multicore-Rechner. Unsere Werkzeuge analysieren **Code-Repositories** und helfen dem Entwickler bei der **Migration** und **Refaktorisierung**.
(c) Wir analysieren Programme. Unsere **Code-Analysewerkzeuge** sind schnell, interaktiv, inkrementell und arbeiten teilweise selbst parallel. Sie finden Wettlaufsituationen, konkurrierende Ressourcenzugriffe etc. im Code und zeigen dem Entwickler Verbesserungsvorschläge punktgenau und in der Entwicklungsumgebung an.
(d) Wir **testen** parallelen Code und **diagnostizieren** Problemursachen. Unsere Werkzeuge erzeugen Testdaten, finden Ursachen von erratischem Laufzeitverhalten und schützen gegen **Authentizitätsangriffe**.

3 Forschung

3.1 AnaCoRe – *Analyse von Code-Repositories*

Bei der Weiterentwicklung von Software führen die Entwickler oftmals sich wiederholende, ähnliche Änderungen durch. Dazu gehört beispielsweise die Anpassung von Programmen an eine veränderte Bibliotheksschnittstelle, die Behebung von Fehlern in funktional ähnlichen Komponenten sowie die Parallelisierung von sequentiellen Programmteilen. Wenn jeder Entwickler die nötigen Änderungen selbst erarbeiten muss, führt dies leicht zu fehlerhaften Programmen, beispielsweise weil weitere zu ändernde Stellen übersehen werden. Wünschenswert wäre stattdessen ein automatisiertes Verfahren, das ähnliche Änderungen erkennt und mit dieser Wissensbasis Software-Entwickler bei weiteren Änderungen unterstützt.

Änderungsextraktion
In 2017 entwickelten wir ein neues Vorschlagssystem mit Namen ARES (Accurate REcommendation System). Verglichen mit bisherigen Ansätzen erzeugt es genauere Vorschläge, da seine Algorithmen Code-Verschiebungen während der Muster- und Vorschlagserzeugung berücksichtigen. Der Ansatz basiert darauf, dass zwei Versionen eines Programms miteinander verglichen werden. Das Werkzeug extrahiert dabei automatisch, welche Änderungen sich zwischen den beiden Versionen ergeben haben, und leitet daraus generalisierte Muster aus zu ersetzenden Code-Sequenzen ab. Diese Muster können anschließend von ARES dazu verwendet werden, analoge Änderungen für den Quellcode anderer Programme automatisch vorzuschlagen.
Zur Extraktion der Änderungen verwenden wir ein baumbasiertes Verfahren. Im Jahr 2016 wurde ein neuer Algorithmus (MTDIFF) für solche baumbasierten Verfahren entwickelt und gut sichtbar publiziert, der die Genauigkeit der Änderungsbestimmung verbessert.

Symbolische Ausführung von Code-Fragmenten
Im Jahr 2014 wurde ein neues Verfahren zur symbolischen Code-Ausführung namens SYFEX entwickelt, welches die Ähnlichkeit des Verhaltens zweier Code-Teilstücke bestimmt. Mit diesem Verfahren soll eine Steigerung der Qualität der Verbesserungsvorschläge erreicht werden. Abhängig von der Anzahl und Generalität der Muster in der Datenbank kann SIFE ohne das neue Verfahren unpassende Vorschläge liefern. Um dem Entwickler nur die passenden Vorschläge anzuzeigen, wird das semantische Verhalten des Vorschlags mit dem semantischen Verhalten des Musters aus der Datenbank verglichen. Weichen beide zu sehr voneinander ab, wird der Vorschlag aus der Ergebnismenge entfernt. Die Besonderheit von SYFEX besteht darin, dass es auf herausgelöste Code-Teilstücke anwendbar ist und keine menschliche Vorkonfiguration benötigt.
SYFEX wurde im Jahr 2015 verfeinert und auf Code-Teilstücke aus Archiven von verschiedenen Software-Projekten angewendet. Der Schwerpunkt im Jahr 2016 lag auf einer Untersuchung, inwieweit SYFEX zum semantischen Vergleich von Abgaben eines Programmierwettbewerbs geeignet ist. In den Jahren 2017 und 2018 wurde SYFEX optimiert. Des Weiteren wurde mit der Erstellung eines Datensatzes semantisch ähnlicher Methoden aus quelloffenen Software-Archiven begonnen, der im Jahr 2019 veröffentlicht wurde.
Verfahren zur symbolischen Ausführung beruhen auf Algorithmen zur Erfüllbarkeitsprüfung von logisch-mathematischen Ausdrücken, um zulässige Ausführungspfade in einem Programm zu bestimmen. Oftmals beanspruchen diese Algorithmen einen großen Teil der aufgewendeten Rechenzeit. Um diese Erfüllbarkeitsprüfung zu beschleunigen, wurde in den Jahren 2019 und 2020 mit einer Technik experimentiert, um komplizierte Ausdrücke durch einfachere Ausdrücke mit gleicher Bedeutung zu ersetzen. Hierbei werden die einfacheren Ausdrücke durch ein Verfahren zur Programmsynthese aufgedeckt. Im Jahr 2020 wurde diese Programmsynthese um ein neuartiges Verfahren ergänzt, das für eine bestimmte Menge an Operationen bereits vorab ermitteln kann, ob sich damit ein Ausdruck mit gleicher Bedeutung wie der kompliziertere Quellausdruck bilden lässt. Unsere im Jahr 2021 erschienene wissenschaftliche Publikation beschreibt dieses Verfahren und zeigt, dass durch dessen Einsatz die Laufzeit von gängigen Programmsynthetisierern im Mittel um 33% verringert werden kann. Ebenfalls im Jahr 2021 wurde das Verfahren auf weitere Klassen von Programmsyntheseproblemen erweitert.

Detektion von semantisch ähnlichen Code-Fragmenten
SYFEX erlaubt es, die semantische Ähnlichkeit zweier Code-Fragmente zu bestimmen. So ist es damit prinzipiell möglich, Paare oder Gruppen von semantisch ähnlichen Code-Fragmenten (semantische Klone) zu identifizieren. Auf Grund des hohen Laufzeitaufwands verbietet sich der Einsatz von SYFEX – wie auch von anderen Werkzeugen dieser Art – allerdings, um in größeren Code-Projekten nach semantisch ähnlichen Code-Fragmenten zu suchen. Im Jahr 2016 wurde deshalb mit der Entwicklung eines Verfahrens begonnen, mit dessen Hilfe die Detektion semantisch ähnlicher Code-Fragmente beschleunigt werden kann. Grundlage dieses Verfahrens ist eine Reihe von sog. Basiskomparatoren, die zwei Code-

Fragmente jeweils hinsichtlich eines Kriteriums (beispielsweise die Anzahl bestimmter Kontrollstrukturen oder die Beschaffenheit der Kontrollflussgraphen) miteinander vergleichen und dabei möglichst geringen Laufzeitaufwand haben. Diese Basiskomparatoren können anschließend zu einer Hierarchie von Verfahren verknüpft werden. Um damit die semantische Ähnlichkeit zweier Fragmente möglichst genau bestimmen zu können, wird mit Hilfe der Genetischen Programmierung nach Hierarchien gesucht, die die von SYFEX für eine Reihe von Code-Paaren berechneten Ähnlichkeitswerte möglichst gut approximieren. Im Rahmen einer ersten Untersuchung hat sich gezeigt, dass sich das implementierte Verfahren tatsächlich für die Bestimmung von semantisch ähnlichen Code-Paaren eignet.
Die Implementierung dieses Verfahrens wurde in den Jahren 2017 und 2018 weiter verbessert. Zudem spielte die tiefergehende Evaluation des Verfahrens auf Basis von Methodenpaaren aus Software-Archiven sowie von Abgaben für Programmieraufgaben eine wichtige Rolle.

Semantische Code-Suche
Häufig steht die bei der Software-Entwicklung zu implementierende Funktionalität bereits in ähnlicher Form als Teil von Programmbibliotheken zur Verfügung. In vielen Fällen ist es ratsam, diese bereits vorhandene Realisierung zu verwenden statt die Funktionalität erneut zu implementieren, beispielsweise um den Aufwand für das Entwickeln und Testen des Codes zu reduzieren.
Voraussetzung für die Wiederverwendung einer für den Anwendugszweck geeigneten Implementierung ist, dass Entwickler diese überhaupt finden können. Zu diesem Zweck werden bereits heute regelmäßig Code-Suchmaschinen verwendet. Etablierte Verfahren stützen sich dabei insbesondere auf syntaktische Merkmale, d.h. der Nutzer gibt beispielsweise eine Reihe von Schlüsselwörtern oder Variablen- und Methodennamen an, nach denen die Suchmaschine suchen soll. Bei diesen Verfahren bleibt die Semantik des zu suchenden Codes unberücksichtigt. Dies führt in der Regel dazu, dass relevante, aber syntaktisch verschiedene Implementierungen nicht gefunden werden („false negatives“) oder dass syntaktisch ähnliche, aber semantisch irrelevante Ergebnisse präsentiert werden („false positives“). Die Suche nach Code-Fragmenten auf Basis ihrer Semantik ist Gegenstand aktueller Forschung.
Im Jahr 2017 wurde am Lehrstuhl mit der Entwicklung eines neuen Verfahrens zur semantischen Code-Suche begonnen. Der Nutzer spezifiziert dabei die gesuchte Funktionalität in Form von Eingabe-Ausgabe-Beispielen. Mit Hilfe eines aus der Literatur stammenden Verfahrens zur Funktionssynthese wird eine Methode erzeugt, die das durch die Beispiele beschriebene Verhalten möglichst genau realisiert. Diese synthetisierte Methode wird dann mit Hilfe des im Rahmen dieses Forschungsprojekts entwickelten Verfahrens zur Detektion von semantisch ähnlichen Code-Fragmenten mit den Methodenimplementierungen vorgegebener Programmbibliotheken verglichen, um ähnliche Implementierungen zu finden, die dem Nutzer als Ergebnis der Suche präsentiert werden. Eine erste Evaluation der prototypischen Implementierung zeigt die Umsetzbarkeit und Verwendbarkeit des Verfahrens.

Cluster-Bildung von ähnlichen Code-Änderungen
Voraussetzung für die Erzeugung generalisierter Änderungsmuster ist es, die Menge aller aus einem Quelltext-Archiv extrahierten Code-Änderungen in Teilmengen zueinander ähnlicher Änderungen aufzuteilen. Im Jahr 2015 wurde diese Erkennung ähnlicher Änderungen im Rahmen eines neuen Werkzeugs C3 verbessert. In einem ersten Schritt wurden verschiedene Metriken für den paarweisen Ähnlichkeitsvergleich der extrahierten Code-Änderungen implementiert und evaluiert. Darauf aufbauend wurden aus der Literatur bekannte Clustering-Algorithmen evaluiert und neue Heuristiken zur automatisierten Bestimmung der jeweiligen Parameter implementiert, um das bisherige naive Verfahren zur Identifizierung ähnlicher Änderungen zu ersetzen. Mit den im Rahmen von C3 implementierten Verfahren konnte im Vergleich zum bisherigen Ansatz eine deutliche Verbesserung erzielt werden. So können mit den neuen Verfahren mehr Gruppen ähnlicher Änderungen identifiziert werden, die sich für die Weiterverarbeitung im Rahmen von SIFE zur Generierung von Vorschlägen eignen.
Die zweite Verbesserung zielt darauf ab, die erhaltenen Gruppen ähnlicher Änderungen zusätzlich automatisiert zu verfeinern. Zu diesem Zweck wurden verschiedene Verfahren aus dem Umfeld des maschi-

nellen Lernens zur Ausreißererkennung untersucht, um Änderungen, die fälschlicherweise einer Gruppe zugeordnet wurden, wieder zu entfernen.
Im Jahr 2016 wurde C3 um eine weitere Metrik zum Vergleich zweier Code-Änderungen erweitert, die im Wesentlichen den textuellen Unterschied zwischen den Änderungen (wie er beispielsweise von dem Unix-Werkzeug 'diff' erzeugt wird) bewertet. Des Weiteren wurde das in C3 implementierte Verfahren im Rahmen eines Konferenzbeitrags veröffentlicht. In diesem Zusammenhang wurde auch der zur Evaluation des Verfahrens erzeugte Datensatz von Gruppen ähnlicher Änderungen unter einer Open-Source-Lizenz veröffentlicht, siehe https://github.com/FAU-Inf2/cthree. Dieser kann zukünftigen Arbeiten als Referenz oder Eingabe dienen. Außerdem wurden prototypisch Verfahren implementiert, mit denen die Ähnlichkeitsberechnung und das Clustering in C3 inkrementell erfolgen können. Diese erlauben es, dass bei neuen Änderungen, die zu einem Software-Archiv hinzugefügt werden, die zuvor bereits berechneten Ergebnisse weiterverwendet werden können und nur ein Teil der Arbeit wiederholt werden muss.

3.2 AuDeRace – *Automatische Erkennung von Wettlaufsituationen*

Große Softwareprojekte, an denen hunderte Entwickler arbeiten, sind schwer zu überblicken und enthalten viele Fehler. Zum Testen solcher Software haben sich automatisierte Tests bewährt, die Teilbereiche der Software (Unit-Tests) oder die komplette Software (Systemtest) testen und Fehler reproduzierbar erkennen können. Dieses Vorgehen funktioniert gut bei sequentiellen Programmen, die ein deterministisches Verhalten aufweisen. Moderne Software enthält jedoch vermehrt Parallelität. Durch diese Nebenläufigkeit treten etliche neue, teils schwer zu lokalisierende, Fehler auf, die nicht mehr durch herkömmliche Testverfahren sicher detektiert werden können. Dazu gehören vor allem Verklemmungen und gleichzeitige Zugriffe auf die selbe Speicherstelle. Ob ein solcher Fehler auftritt, hängt von dem konkreten Ausführungsplan der Aktivitätsfäden ab. Dieser unterscheidet sich jedoch bei jeder Ausführung und ist auch stark von dem darunterliegenden System abhängig. Somit treten entsprechende Fehler normalerweise nicht bei jedem Testlauf auf, je nach Testsystem sogar niemals. Aus diesem Grund sind die herkömmlichen Testverfahren nicht mehr ausreichend für moderne, nebenläufige Software.
In dem Projekt AuDeRace werden Verfahren entwickelt, die Nebenläufigkeitsfehler reproduzierbar, effizient und mit möglichst geringen Aufwand für Entwickler erkennen können. Unter anderem wird eine Technik entwickelt, die es ermöglicht, Tests um einen Ablaufplan zu erweitern, durch den die Ausführung weiterhin deterministisch bleibt. Ein weiteres generelles Problem an Tests bleibt dabei aber bestehen: Der Entwickler muss sich zunächst Testfälle überlegen und diese anschließend implementieren. Insbesondere im Kontext von Parallelität ist es aber extrem schwer, sich das Verhalten von Programmen vorzustellen und problematische Stellen zu identifizieren. Deshalb sollen kritische Stellen automatisiert erkannt werden, bevor überhaupt entsprechende Tests geschrieben wurde. Es existieren bereits Ansätze, potentiell kritische Stellen zu detektieren, allerdings werden dabei entweder sehr viele fälschliche Warnungen generiert oder die Analyse ist immens zeitintensiv. Ziel dieses Projektes ist es, durch eine geeignete Kombination aus statischer und dynamischer Analyse, die Anzahl an falschen Erkennungen zu reduzieren, bzw. langsame Analysen zu beschleunigen, sodass die Verfahren nicht nur in kleinen Testbeispielen funktionieren, sondern auch komplexe Softwareprojekte effizient analysieren können.

Im Jahr 2016 wurden bestehende Ansätze und Programme auf ihre Tauglichkeit untersucht. Dabei wurde ein vielversprechendes Verfahren ausgemacht, das mithilfe von Model Checking und vorgegebenen Bedingungen Ablaufpläne konstruiert, die ungewolltes Verhalten erzeugen. Allerdings zeigten die Ergebnisse ein deutliches Problem hinsichtlich eines produktiven Einsatzes, da in sinnvoller Zeit nur sehr kleine Programme analysiert werden konnten. Daher beschäftigte sich das Projekt damit, die Programme um möglichst viele Anweisungen zu kürzen, die nichts mit den gesuchten Wettlaufsituationen zu tun haben. Dadurch sollen spätere Analysen beschleunigt werden.

Im Jahr 2017 wurden die Untersuchungen zur automatischen Reduktion von Programmen weitergeführt. Außerdem wurde erforscht, ob sich die existierende Technik des Mutationstestens auch auf nebenläufige Programme erweitern lässt. Die Ergebnisse zeigen, dass es durchaus möglich ist, Tests für eine parallele Software qualitativ zu bewerten. Allerdings muss teilweise auf Heuristiken zurückgegriffen werden, um auch größere Projekte in vertretbarer Zeit bewerten zu können.

Im Jahr 2018 lag der Fokus auf einer deterministischen Ausführung von Testfällen. Es wurde ein Konzept erarbeitet, um reproduzierbare Ergebnisse bei der Ausführung zu erzielen: Ein zusätzlicher Ablaufplan spezifiziert das Ablaufverhalten der verschiedenen Aktivitäten. Eine Instrumentierung von vorher markierten Positionen im Code und anderen relevanten Bytecode-Instruktionen soll hier während der Ausführung entsprechend die Kontrolle an eine Verwaltungsaktivität abgeben, die den Ablauf steuert. Damit die Markierungen (Angabe von Positionen im Quellcode) auch nach Änderungen am Quellcode gültig bleiben, sollen diese automatisch auf eine neue Version angepasst werden können. Eine Merge-Technik ähnlich derer zu Versionsverwaltungssystemen soll hier eingesetzt werden.

Das Projekt stellte bis 2019 einen Beitrag des Lehrstuhls Informatik 2 zum IZ ESI (Embedded Systems Initiative, http://www.esi.fau.de/) dar. In diesem Rahmen wurden verschiedene Ansätze zur Verbesserung der Qualität nebenläufiger Software untersucht. Zusammenfassend wurde festgestellt, dass verschiedenartige Verfahren notwendig und realisierbar sind, allerdings meistens die hohen Laufzeiten ein großes Problem darstellen.

Über das ESI Projekt hinaus wurde die Verwendung von Mutationstests durch die Entwicklung eines Werkzeugs zur Äquivalenzerkennung und Testfallgenerierung verbessert. Eine entsprechende Publikation wurde eingereich und angenommen.

Im Jahr 2020 evaluierten wir Ansätze und Techniken zur Erkennung von externen Wettlaufsituationen. Während bei klassischen Wettlaufsituationen mehrere Aktivitätsfäden eines Programms nicht richtig miteinander arbeiten, treten externe Wettlaufsituationen bei der Interaktion der Software mit anderen, unabhängigen und unbekannten Komponenten auf. Das können andere gleichzeitig laufende Programme sein, das Betriebssystem oder sogar Code eines böswilligen Angreifers, der gezielt mit der analysierten Software interferiert.

Im Jahr 2021 wurde ein Verfahren entwickelt, um statisch Wettlaufsituationen bei der Verwendung von externen Ressourcen zu erkennen. Ein häufiges Muster ist, dass Programme die Eigenschaften von Dateien abrufen und später auf Grund der vorherigen Überprüfungen auf die Dateien erneut zugreifen. Ändert sich der Zustand der Datei in der Zwischenzeit, kann eine Vielzahl von Problemen auftreten (time-of-check to time-of-use). Neben unerwartetem Verhalten können jedoch auch Angreifer durch geeignetes Anpassen der Datei, gezielt Fehlverhalten auslösen oder das System kompromittieren. Mit dem entworfenen Verfahren sollen solche Problemstellen in einer Software automatisch erkannt werden.

3.3 AutoCompTest – *Automatisiertes Testen von Übersetzern*

Übersetzer für Programmiersprachen sind äußerst komplexe Anwendungen, an die hohe Korrektheitsanforderungen gestellt werden: Ist ein Übersetzer fehlerhaft (d.h. weicht sein Verhalten vom dem durch die Sprachspezifikation definierten Verhalten ab), so generiert dieser u.U. fehlerhaften Code oder stürzt bei der Übersetzung mit einer Fehlermeldung ab. Solche Fehler in Übersetzern sind oftmals schwer zu bemerken oder zu umgehen. Nutzer erwarten deshalb i.A. eine (möglichst) fehlerfreie Implementierung des verwendeten Übersetzers.

Leider lassen sowohl vergangene Forschungsarbeiten als auch Fehlerdatenbanken im Internet vermuten, dass kein real verwendeter Übersetzer fehlerfrei ist. Es wird deshalb an Ansätzen geforscht, mit deren

Hilfe die Qualität von Übersetzern gesteigert werden kann. Da die formale Verifikation (also der Beweis der Korrektheit) in der Praxis oftmals nicht möglich oder rentabel ist, zielen viele der Forschungsarbeiten darauf ab, Übersetzer möglichst umfangreich und automatisiert zu testen. In den meisten Fällen erhält der zu testende Übersetzer dabei ein Testprogramm als Eingabe. Anschließend wird das Verhalten des Übersetzers bzw. des von ihm generierten Programms überprüft: Weicht dieses vom erwarteten Verhalten ab (stürzt der Übersetzer also beispielsweise bei einem gültigen Eingabeprogramm mit einer Fehlermeldung ab), so wurde ein Fehler im Übersetzer gefunden. Soll dieser Testvorgang automatisiert stattfinden, ergeben sich zwei wesentliche Herausforderungen:

- Woher kommen die Testprogramme, auf die der Übersetzer angewendet wird?
- Was ist das erwartete Verhalten des Übersetzers bzw. des von ihm erzeugten Codes? Wie kann bestimmt werden, ob das tatsächliche Verhalten des Übersetzers korrekt ist?

Während die wissenschaftliche Literatur diverse Lösungen für die zweite Herausforderung vorstellt, die auch in der Praxis bereits etabliert sind, stellt die automatisierte Generierung zufälliger Testprogramme noch immer eine große Hürde dar. Damit Testprogramme zur Detektion von Fehlern in allen Teilen des Übersetzers verwendet werden können, müssen diese allen Regeln der jeweiligen Programmiersprache genügen, d.h. die Programme müssen syntaktisch und semantisch korrekt (und damit übersetzbar) sein. Auf Grund der Vielzahl an Regeln „echter“ Programmiersprachen stellt die Generierung solcher übersetzbarer Programme eine schwierige Aufgabe dar. Dies wird zusätzlich dadurch erschwert, dass das Programmgenerierungsverfahren möglichst effizient arbeiten muss: Die wissenschaftliche Literatur zeigt, dass die Effizienz eines solchen Verfahrens maßgebliche Auswirkungen auf seine Effektivität hat – nur wenn in kurzer Zeit viele (und große) Programme generiert werden können, kann das Verfahren sinnvoll zur Detektion von Übersetzerfehlern eingesetzt werden.

In der Praxis scheitert das automatisierte Testen von Übersetzern deshalb oftmals daran, dass kein zugeschnittener Programmgenerator verfügbar ist und die Entwicklung eines solchen einen zu hohen Aufwand bedeutet. Ziel unseres Forschungsprojekts ist daher die Entwicklung von Verfahren, die den Aufwand für die Implementierung von effizienten Programmgeneratoren reduzieren.

Im Jahr 2018 haben wir mit der Entwicklung eines entsprechenden Werkzeugs begonnen. Als Eingabe dient eine Spezifikation der syntaktischen und semantischen Regeln der jeweiligen Programmiersprache in Form einer abstrakten Attributgrammatik. Eine solche erlaubt eine knappe Notation der Regeln auf hohem Abstraktionsniveau. Ein von uns neu entwickelter Algorithmus erzeugt dann Testprogramme, die allen spezifizierten Regeln genügen. Der Algorithmus nutzt dabei diverse technische Ideen aus, um eine angemessene Laufzeit zu erreichen. Dies ermöglicht die Generierung großer Testfallmengen in vertretbarer Zeit, auch auf üblichen Arbeitsplatzrechnern. Eine erste Evaluation hat nicht nur gezeigt, dass unser Verfahren sowohl effektiv als auch effizient ist, sondern auch dass es flexibel einsetzbar ist. So haben wir mit Hilfe unseres Verfahrens nicht nur Fehler in den C-Übersetzern gcc und clang entdeckt (unser Verfahren erreicht dabei eine ähnliche Fehleraufdeckungsgüte wie ein sprachspezifischer Programmgenerator aus der wissenschaftlichen Literatur), sondern auch diverse Bugs in mehreren SMT-Entscheidern. Einige der von uns entdeckten Fehler waren den jeweiligen Entwicklern zuvor noch unbekannt.

Im Jahr 2019 haben wir zusätzliche Features für das Schreiben von Sprachspezifikationen implementiert und die Effizienz des Programmgenerierungsverfahrens gesteigert. Durch die beiden Beiträge konnte der Durchsatz unseres Werkzeugs deutlich gesteigert werden. Des Weiteren haben wir mit Hilfe neuer Sprachspezifikationen Fehler in Übersetzern für die Programmiersprachen Lua und SQL aufgedeckt. Die Ergebnisse unserer Arbeit sind in eine Ende 2019 eingereichte (und inzwischen angenommene) wissenschaftliche Publikation eingeflossen. Neben der Arbeit an unserem Verfahren zur Programmgenerierung haben wir außerdem mit der Arbeit an einem Verfahren zur Testfallreduzierung begonnen. Das Verfahren reduziert die Größe eines zufällig generierten Testprogramms, das einen Fehler in einem Übersetzer auslöst, um die Suche nach der Ursache des Fehlers zu vereinfachen.

Im Jahr 2020 lag der Fokus des Forschungsprojekts auf sprachunabhängigen Verfahren zur automatischen Testfallreduzierung. Die wissenschaftliche Literatur schlägt unterschiedliche Reduzierungsverfahren vor. Da es bislang allerdings keinen aussagekräftigen Vergleich dieser Verfahren gibt, ist unklar, wie effizient und effektiv die vorgeschlagenen Reduzierungsverfahren tatsächlich sind. Wie wir festgestellt haben, gibt es dafür im Wesentlichen zwei Gründe, die darüber hinaus auch die Entwicklung und Evaluation neuer Verfahren erschweren. Zum einen verwenden die vorhandenen Implementierungen der vorgeschlagenen Reduzierungsverfahren unterschiedliche Implementierungssprachen, Programmrepräsentationen und Eingabegrammatiken. Mit diesen Implementierungen ist ein fairer Vergleich dieser Verfahren deshalb kaum möglich. Zum anderen gibt es keine umfangreiche Sammlung von (noch unreduzierten) Testprogrammen zur Evaluation von Reduzierungsverfahren. Dies hat zur Folge, dass die publizierten Reduzierungsverfahren jeweils nur mit sehr wenigen Testprogrammen evaluiert wurden, was die Aussagekraft der präsentierten Ergebnisse beeinträchtigt. Da in manchen Fällen darüber hinaus nur Testprogramme in einer einzigen Programmiersprache verwendet wurden, ist bislang unklar, wie gut diese Verfahren für Testprogramme in anderen Programmiersprachen funktionieren (wie sprachunabhängig diese Verfahren also tatsächlich sind). Um diese Lücken zu schließen, haben wir im Jahr 2020 mit der Implementierung eines Frameworks begonnen, das die wichtigsten Reduzierungsverfahren beinhaltet und so einen fairen Vergleich dieser Verfahren ermöglicht. Außerdem haben wir mit der Erstellung einer Testfallsammlung begonnen. Diese beinhaltet bereits etwa 300 Testprogramme in den Sprachen C und SMT-LIB 2, die etwa 100 unterschiedliche Fehler in realen Übersetzern auslösen. Diese Testfallsammlung erlaubt nicht nur aussagekräftigere Vergleiche von Reduzierungsverfahren, sondern verringert außerdem den Aufwand für die Evaluation zukünftiger Verfahren. Anhand erster Experimente konnten wir feststellen, dass es bislang kein Reduzierungsverfahren gibt, dass in allen Fällen am besten geeignet ist.

Außerdem haben wir uns im Jahr 2020 mit der Frage beschäftigt, wie das im Rahmen des Forschungsprojekts entstandene Framework zur Generierung zufälliger Testprogramme erweitert werden kann, um neben funktionalen Fehlern auch Performance-Probleme in Übersetzern finden zu können. Im Rahmen einer Abschlussarbeit ist dabei ein Verfahren entstanden, das eine Menge zufällig generierter Programme mit Hilfe von Optimierungstechniken schrittweise so verändert, dass die resultierenden Programme im getesteten Übersetzer deutlich höhere Laufzeiten als in einer Referenzimplementierung auslösen. Erste Experimente haben gezeigt, dass so tatsächlich Performance-Probleme in Übersetzern gefunden werden können.

Im Jahr 2021 haben wir die Implementierung der wichtigsten Reduzierungsverfahren aus der wissenschaftlichen Literatur sowie die Erstellung einer Testfallsammlung für deren Evaluation abgeschlossen. Aufbauend darauf haben wir außerdem einen quantitativen Vergleich der Verfahren durchgeführt; soweit wir wissen, handelt es sich dabei um den mit Abstand umfangreichsten und aussagekräftigsten Vergleich bisheriger Reduzierungsverfahren. Unsere Ergebnisse zeigen, dass es bislang kein Verfahren gibt, das in allen Anwendungsfällen am besten geeignet wäre. Außerdem konnten wir feststellen, dass es für alle Verfahren zu deutlichen Ausreißern kommen kann, und zwar sowohl hinsichtlich Effizienz (also wie schnell ein Reduzierungsverfahren ein Eingabeprogramm reduzieren kann) als auch Effektivität (also wie klein das Ergebnis eines Reduzierungsverfahrens ist). Dies deutet darauf hin, dass es noch Potenzial für die Entwicklung weiterer Reduzierungsverfahren in der Zukunft gibt, und unsere Ergebnisse liefern einige Einsichten, was dabei zu beachten ist. So hat sich beispielsweise gezeigt, dass ein Hochziehen von Knoten im Syntaxbaum unabdingbar für die Generierung möglichst kleiner Ergebnisse (und damit eine hohe Effektivität) ist und dass eine effiziente Behandlung von Listenstrukturen im Syntaxbaum notwendig ist. Die Ergebnisse unserer Arbeit sind in eine im Jahr 2021 eingereichte und angenommene Publikation eingeflossen.

Außerdem haben wir im Jahr 2021 untersucht, ob bzw. wie sich die Effektivität unseres Programmgenerierungsverfahrens steigern lässt, wenn bei der Generierung die Überdeckung der zugrundeliegenden Grammatik berücksichtigt wird. Im Rahmen einer Abschlussarbeit wurden dazu unterschiedliche, aus der

wissenschaftlichen Literatur stammende kontextfreie Überdeckungsmetriken für den Anwendungsfall adaptiert sowie implementiert und evaluiert. Dabei hat sich gezeigt, dass die Überdeckung hinsichtlich einer kontextfreien Metrik nur bedingt mit der Fehleraufdeckung korreliert. In zukünftigen Arbeiten sollte deshalb untersucht werden, ob Überdeckungsmetriken, die auch kontextsensitive, semantische Eigenschaften berücksichtigen, besser für diesen Anwendungsfall geeignet sind.

3.4 Holoware – *Kooperative Exploration und Analyse von Software in einer Virtual/Augmented Reality Appliance*

Der Aufwand für das Verstehen von Software umfasst in Entwicklungsprojekten bis zu 30% und in Wartungsprojekten bis zu 80% der Programmieraufwände. Deshalb wird in modernen Arbeitsumgebungen zur Software-Entwicklung eine effiziente und effektive Möglichkeit zum Software-Verstehen benötigt. Die dreidimensionale Visualisierung von Software steigert das Verständnis der Sachverhalte deutlich, und damit liegt eine Nutzung von Virtual-Reality-Techniken nahe. Im Rahmen des Holoware Projekts schaffen wir eine Umgebung, in der Software mit Hilfe von VR/AR (Virtual/Augmented Reality) und Technologien der Künstlichen Intelligenz (KI) kooperativ exploriert und analysiert werden kann. In dieser virtuellen Realität wird ein Software-Projekt oder -verbund dreidimensional visualisiert, sodass mehrere Benutzer gleichzeitig die Software gemeinsam und kooperativ erkunden und analysieren können. Verschiedene Nutzer können dabei aus unterschiedlichen Perspektiven und mit unterschiedlich angereicherten Sichten profitieren und erhalten so einen intuitiven Zugang zur Struktur und zum Verhalten der Software. Damit sollen verschiedene Nutzungsszenarien möglich sein, wie z.B. die Anomalieanalyse im Expertenteam, bei der mehrere Domänenexperten gemeinsam eine Laufzeitanomalie der Software analysieren. Sie sehen dabei die selbe statische Struktur der Software, jeder Experte jedoch angereichert mit den für ihn relevanten Detail-Informationen. Im VR-Raum können sie ihre Erkenntnisse kommunizieren und so ihre unterschiedliche Expertise einbringen.

Darüber hinaus werden die statischen und dynamischen Eigenschaften des Software-Systems analysiert. Zu den statischen Eigenschaften zählen beispielsweise der Source-Code, statische Aufrufbeziehungen oder auch Metriken wie LoC, zyklomatische Komplexität o. Ä. Dynamische Eigenschaften lassen sich in Logs, Ablaufspuren (Traces), Laufzeitmetriken oder auch Konfigurationen, die zur Laufzeit eingelesen werden, gruppieren. Die Herausforderung liegt darin, diese Vielzahl an Informationen zu aggregieren, analysieren und korrelieren. Es wird eine Anomalie- und Signifikanz-Detektion entwickelt, die sowohl Struktur- als auch Laufzeitauffälligkeiten automatisch erkennt. Zudem wird ein Vorhersagesystem aufgebaut, das Aussagen über die Komponentengesundheit macht. Dadurch kann beispielsweise vorhergesagt werden, welche Komponente gefährdet ist, demnächst auszufallen. Bisher werden die Ablaufspuren um die Log-Einträge angereichert, wodurch ein detailliertes Bild der dynamischen Aufrufbeziehungen entsteht. Diese dynamischen Beziehungen werden auf den statischen Aufrufgraph abgebildet, da sie Aufrufe beschreiben, die aus der statischen Analyse nicht hervorgehen (beispielsweise REST-Aufrufe über mehrere verteilte Komponenten).

Im Jahr 2018 konnten folgende wesentlichen Beiträge geleistet werden:

- Entwicklung eines funktionsfähigen VR-Visualisierungsprototyps zu Demonstrations- und Forschungszwecken.
- Mapping von dynamischer Laufzeitdaten auf die statische Struktur als Grundlage für deren Analyse und Visualisierung.
- Entwurf und Implementierung der Anomalieerkennung von Ablaufspuren durch ein Unsupervised-Learning-Verfahren.

Im Jahr 2019 konnten weitere Verbesserungen erreicht werden:

- Erweiterung des Prototyps um die Darstellung dynamischen Software-Verhaltens.
- Kooperative (Remote-)Nutzung des Visualisierungsprototyps.
- Auswertung von Commit-Nachrichten zur Anomalieerkennung.
- Clustering der Aufrufe eines Systems nach Anwendungsfällen.

In dem Papier „Towards Collaborative and Dynamic Software Visualization in VR“, das auf der International Conference on Computer Graphics Theory and Applications (VISIGRAPP) 2020 angenommen wurde, haben wir die Wirksamkeit unseres Prototyps zur Effizienzsteigerung beim Software-Verstehen gezeigt. Im Jahr 2020 wurde unser Papier „A Layered Software City for Dependency Visualization“ auf der International Conference on Computer Graphics Theory and Applications (VISIGRAPP) 2021 angenommen und mit dem „Best Paper“-Award ausgezeichnet. Wir konnten belegen, dass das von uns entwickelte Layered Layout für Software-Städte das Analysieren von Software-Architektur vereinfacht und das Standard-Layout bei weitem übertrifft. Der finale Prototyp und die Publikationen, die im Rahmen des Forschungsprojektes entstanden sind, führten zu einem erfolgreichen Projektabschluss.

Nach Auslaufen der offiziellen Projektförderung durften wir in 2021 eine erweiterte Version des Award-Papiers („Static And Dynamic Dependency Visualization in a Layered Software City“) als Zeitschriftenartikel zur Begutachtung einreichen. Hier stellen wir eine Nacht-Ansicht der Stadt vor, in der die dynamischen Aufrufbeziehungen als Bögen visualisiert werden. Wir widmeten uns also einem zentralen, noch offenen Punkt: der Visualisierung von dynamischen Abhängigkeiten. In dem Papier „Trace Visualization within the Software City Metaphor: A Controlled Experiment on Program Comprehension“ auf der IEEE Working Conference on Software Visualization (VISSOFT) haben wir dynamische Abhängigkeiten innerhalb der Software-Stadt über Licht-Intensitäten aggregiert dargestellt und konnten zeigen, dass diese Darstellung hilfreicher ist als alle Abhängigkeiten zu zeichnen. Auch für dieses Papier wurden wir zur Einreichung eines erweiterten Artikels „Trace Visualization within the Software City Metaphor: Controlled Experiments on Program Comprehension“ zur Begutachtung aufgefordert. Wir zeigen dort eine erweiterte Darstellung dynamischer Abhängigkeiten und färben Bögen basierend auf HTTP Statuscodes.

3.5 ORKA-HPC – *OpenMP für rekonfigurierbare heterogene Architekturen*

High-Performance Computing (HPC) ist ein wichtiger Bestandteil für die europäische Innovationskapazität und wird auch als ein Baustein bei der Digitalisierung der europäischen Industrie gesehen. Rekonfigurierbare Technologien wie Field Programmable Gate Array (FPGA) Module gewinnen hier wegen ihrer Energieeffizienz, Performance und ihrer Flexibilität immer größere Bedeutung.
Es wird außerdem zunehmend auf HPC-Systeme mit heterogenen Architekturen gesetzt, auch auf solche mit FPGA-Beschleunigern. Die große Flexibilität dieser FPGAs ermöglicht es, dass eine große Klasse von HPC-Applikationen mit FPGAs realisiert werden kann. Allerdings ist deren Programmierung bisher vorwiegend Spezialisten vorbehalten und sehr zeitaufwendig, wodurch deren Verwendung in Bereichen des wissenschaftlichen Höchstleistungsrechnens derzeit noch selten ist.
Im HPC-Umfeld gibt es verschiedenste Programmiermodelle für heterogene Rechnersysteme mit einigen Typen von Beschleunigern. Gängige Programmiermodelle sind zum Beispiel OpenCL (opencl.org), OpenACC (openacc.org) und OpenMP (OpenMP.org). Eine produktive Verwendbarkeit dieser Standards für FPGAs ist heute jedoch noch nicht gegeben.

Ziele des ORKA Projektes sind:

1. Nutzung des OpenMP-4.0-Standards als Programmiermodell, um ohne Spezialkenntnisse heterogene Rechnerplattformen mit FPGAs als rekonfigurierbare Architekturen durch portable Implementierungen eine breitere Community im HPC-Umfeld zu erschließen.

2. Entwurf und Implementierung eines Source-to-Source-Frameworks, welches C/C++-Code mit OpenMP-4.0-Direktiven in ein ausführbares Programm transformiert, das die Host-CPUs und FPGAs nutzt.
3. Nutzung und Erweiterung existierender Lösungen von Teilproblemen für die optimale Abbildung von Algorithmen auf heterogene Systeme und FPGA-Hardware.
4. Erforschung neuer (ggf. heuristischer) Methoden zur Optimierung von Programmen für inhärent parallele Architekturen.

Im Jahr 2018 wurden folgende wesentlichen Beiträge geleistet:

- Entwicklung eines source-to-source Übersetzerprototypen für die Umschreibung von OpenMP-C-Quellcode (vgl. Ziel 2).
- Entwicklung eines HLS-Übersetzerprototypen, der in der Lage ist, C-Code in Hardware zu übersetzen. Dieser Prototyp bildet die Basis für die Ziele 3 und 4.
- Entwicklung mehrerer experimenteller FPGA-Infrastrukturen für die Ausführung von Beschleunigerkernen (nötig für die Ziele 1 und 2).

Im Jahr 2019 wurden folgende wesentlichen Beiträge geleistet:

- Veröffentlichung zweier Papiere: „OpenMP on FPGAs - A Survey“ und „OpenMP to FPGA Offloading Prototype using OpenCL SDK“.
- Erweiterung des source-to-source Übersetzerprototypen um OpenMP-Target-Outlining (incl. Smoke-Tests).
- Fertigstellung des technischen Durchstichs für den ORKA-HPC-Prototypen (OpenMP-zu-FPGA-Übersetzer).
- Benchmark-Suite für die quantitative Leistungsanalyse von ORKA-HPC.
- Erweiterung des source-to-source Übersetzerprototypen um das Genom für die genetische Optimierung der High-Level-Synthese durch Einstellen von HLS-Pragmas.
- Prototypische Erweiterung des TaPaSCo-Composers um ein (optionales) automatisches Einfügen von Hardware-Synchronisationsprimitiven in TaPaSCo-Systeme.

Im Jahr 2020 wurden folgende wesentlichen Beiträge geleistet:

- Weiterentwicklung der Genetischen Optimierung.
- Aufbau eines Docker-Containers für zuverlässige Reproduzierbarkeit der Ergebnisse.
- Integration der Softwarekomponenten der Projektpartner.
- Plugin-Architektur für Low-Level-Platformen.
- Implementation und Integration zweier LLP-Plugin-Komponenten.
- Erweiterung des akzeptierten OpenMP-Sprachstandards.
- Erweiterung der Test-Suite.

Im Jahr 2021 wurden folgende wesentlichen Beiträge geleistet:

- Erweiterung der Benchmark-Suite.
- Erweiterung der Test-Infrastruktur.
- Erfolgreicher Projektabschluss mit Live-Demo für den Projektträger.
- Evaluation des Projekts.
- Veröffentlichung der Publikation „ORKA-HPC - Practical OpenMP for FPGAs“.

- Veröffentlichung des Quell-Codes und der Disseminationsumgebung auf Github.
- Erweiterung des akzeptierten OpenMP-Sprachstandards um neue OpenMP-Klauseln für die Steuerung der FPGA-bezogenen Transformationen.
- Weiterentwicklung der Genetischen Optimierung.
- Untersuchung des Verhältnisses von HLS-Leistungsschätzwerten und tatsächlichen Leistungskennzahlen.
- Aufbau eines linearen Regressionsmodells für die Vorhersage der tatsächlichen Leistungskennzahlen auf Basis der HLS-Schätzwerte.
- Entwicklung von Infrastruktur für die Übersetzung von OpenMP-Reduktionsklauseln.
- Erweiterung um die Übersetzung vom OpenMP-Pragma „parallel for“ in ein paralleles FPGA-System.

3.6 RuNN – *Rekurrente Neuronale Netze (RNNs) zur echtzeitnahen Bestimmung nichtlinearer Bewegungsmodelle*

Mit wachsender Verfügbarkeit von Information über eine Umgebung (z.B. eine Sporthalle) und über die Objekte darin (z.B. Sportler in der Halle) steigt das Interesse, diese Informationen gewinnbringend zusammenzuführen (sog. Information Fusion) und zu verarbeiten. Zum Beispiel will man physikalisch korrekte Animationen (z.B. in der virtuellen Realität) von komplexen und hochdynamischen Bewegungen (z.B. in Sportsituationen) in Echtzeit rekonstruieren. Ebenso könnten z.B. auch Fertigungsanlagen der Industrie, die unter ungünstigen Umgebungsverhältnissen leiden (bspw. Magnetfeldinterferenzen oder fehlendes GPS-Signal), von bspw. hochpräziser Warenortung profitieren. Typischerweise verwendet man, um Bewegungen zu beschreiben, entweder Posen, die einen „Snapshot“ eines Bewegungszustands beschreiben (z.B. Stillstand), oder ein Bewegungsmodell, welches eine Bewegung im zeitlichen Verlauf beschreibt (z.B. Laufen oder Rennen). Außerdem können menschliche Bewegungen durch unterschiedliche Sensoren (z.B. am Körper) erfasst und in Form von Posen und Bewegungsmodellen abgebildet werden. Dabei liefern verschiedene Typen von modernen Sensoren (bspw. Kamera-, Funk- und Inertial-Sensoren) Informationen von unterschiedlicher Qualität.
Prinzipiell ist mit Hilfe teurer und hochpräziser Messinstrumente die Extraktion der Posen und resp. des Bewegungsmodells bspw. aus Positionen (Positionen, z.B. menschlicher Extremitäten, können Posen und Bewegungsmodelle beschreiben oder durch diese beschrieben werden) auf kleinen Trackingflächen fehlerfrei möglich. Kamerabasierte Sensorik liefert dabei die benötigten hochfrequenten hochpräzisen Referenzmessungen auf kleinen Flächen. Allerdings sinkt mit zunehmender Größe der Trackingfläche die Tauglichkeit kamerabasierter Systeme (auf Grund von Ungenauigkeiten oder Problemen durch Verdeckung). Ebenso liefern Funk- und Inertial-Sensoren nur verrauschte und ungenaue Messungen auf großen Flächen. Eine auf Bayes‘schen Filtern basierende Kopplung von Funk- und Inertial-Sensoren erzielt zwar eine höhere Genauigkeit. Diese ist aber noch immer unzureichend, um z.B. im Sport menschliche Bewegungen (abrupte und schnelle Bewegungsänderungen) auf großen Flächen sensorisch präzise zu erfassen. Damit sind die resultierenden Bewegungsmodelle ungenau.
Ferner ist jede menschliche Bewegung hochgradig nichtlinear (resp. nicht vorhersagbar). Diese Nichtlinearität lässt sich mit Hilfe heutiger Bewegungsmodelle, wie sie bspw. durch Bayes‘schen Filter beschrieben werden, nicht korrekt abbilden, da diese (statistischen) Methoden ein nichtlineares Problem in lineare Teilprobleme herunterbrechen, die wiederum die Bewegung nicht physikalisch korrekt repräsentieren können. Darüber hinaus erzeugen aktuelle Verfahren hohe Latenz, wenn Genauigkeit gefordert ist. Aufgrund dieser drei Probleme (ungenaue Positionsdaten auf großen Flächen, Nichtlinearität und Latenz) sind heutige Verfahren bspw. für Sportanwendungen unbrauchbar, die kurze Antwortzeiten fordern. Im Rahmen dieses Projekts wird mit Hilfe von Methoden des maschinellen Lernens diesen Nichtlinearitäten

entgegengewirkt. So umfasst das Projekt die Erforschung rekurrenter neuronaler Netze (RNN) zur Bestimmung nichtlinearer Bewegungsmodelle. Nichtlineare menschliche Bewegungen (z.B. die Lage des Kopfes zum Rumpf während des Laufens oder Rennens), können mittels moderner Bayes'scher Filterverfahren (z.B. Kalman- und Partikel-Filter) und anderer statistischer Methoden nur durch ihre linearen Anteile und somit physikalisch nicht vollständig korrekt beschrieben werden.
Daher ist das Kernziel, zu evaluieren, wie Methoden des maschinellen Lernens zur Beschreibung von komplexen und nichtlinearen Bewegungen eingesetzt werden können. Es wurde deshalb untersucht, ob RNNs die Bewegungen eines Objektes physikalisch korrekt beschreiben und bisherige Methoden unterstützen oder ersetzen können. Im Rahmen einer großangelegten Parameterstudie wurden physikalische korrekte Bewegungen simuliert und auf diesen Simulationen RNN-Verfahren optimiert. Es konnte erfolgreich gezeigt werden, dass RNN-Modelle mit Hilfe geeigneter Trainingsverfahren entweder physikalische Zusammenhänge oder Bewegungsformen erlernen.

Im Rahmen dieses Projekts werden drei wesentliche Themen bearbeitet:

I. Eine Basisimplementierung untersucht, wie und warum Methoden des maschinellen Lernens zur Bestimmung von Bewegungsmodellen von Menschen eingesetzt werden können.

Im Jahr 2018 wurde zunächst ein tieferes Verständnis der Ausgangssituation und Problemstellung aufgebaut. Mit Hilfe verschiedener Basisimplementierungen (unterschiedlicher Bewegungsmodelle) wurde untersucht (1) wie sich unterschiedliche Bewegungen (z.B. Menschen: Laufen, Rennen, Slalom und Fahrzeuge: Mäander, Zig-Zag) auf Messungenauigkeiten der verschiedenen Sensorfamilien auswirken, (2) wie sich Messungenauigkeiten verschiedener Sensorfamilien (z.B. sichtbare Orientierungsfehler, hörbare Störgeräusche und bewusste künstliche Messfehler) auf die menschliche Bewegung auswirken und (3) wie sich verschiedene Filtermethoden zur Fehlerkorrektur (Balanceakt zwischen Genauigkeit und Latenz) auf die Bewegung und Sensoren auswirken. Darüber hinaus konnte (4) gezeigt werden, wie Messungenauigkeiten (bedingt durch den Einsatz aktueller Bayes'scher Filterverfahren) mit der menschlichen Körperhaltung (bspw. Gangapparat) nichtlinear korrelieren und wie Auswirkungen der Messfehler auf die Gesundheit (Simulatorkrankheit) mittels maschinellen Lernens vorhergesagt werden können. Es wurden Methoden des maschinellen und tiefen Lernens zur Bewegungserfassung (Mensch: Kopf, Körper, obere und untere Extremität; Fahrzeug: ein- und zweiachsig) und Bewegungsrekonstruktion (5) auf Basis von Inertial-, Kamera- und Funksensoren studiert und verschiedene Methoden zur Merkmalsextraktion (bspw. SVM, DT, k-NN, VAE, 2D-CNN, 3D-CNN, RNN, LSTMs, M/GRU) untersucht. Diese wurden u. A. zu verschiedenen hybriden Filtermodellen verschaltet, um extrahierte Merkmale um zeitliche und kontextsensitive Bewegungsinformationen anzureichern und so möglicherweise genauere, robustere und echtzeitnahe Bewegungsmodelle zu erstellen. So konnten (6) Bewegungsmodelle für mehrachsige Fahrzeuge (Gabelstapler) auf Basis von Inertial-, Funk- und Kameradaten gelernt werden, die auf unterschiedliche Umgebungen, respektive Trackingflächen (Größe, Form und sensorische Struktur bspw. Magnetfeld, Mehrwege, Texturierung und Beleuchtung) generalisieren. Weiter (7) konnte ein tieferes Verständnis der Auswirkungen von nicht konstant beschleunigten Bewegungsmodellen auf Funksignale untersucht werden. Auf Basis dieser Erkenntnisse konnte ein LSTM Modell angelernt werden, das unterschiedliche Bewegungsgeschwindigkeiten und Bewegungsformen eines einachsigen Roboters (Segway) nahe Echtzeit und genauer als herkömmliche Verfahren vorhersagen kann.
Im Jahr 2019 wurde festgestellt, dass diese Modelle auch die menschliche Bewegung (menschliches Bewegungsmodell) vorhersagen können. Weiter wurde im Jahr 2019 festgestellt, dass die LSTM Modelle zur Laufzeit entweder vollständig autark oder als Stützstellen in Lokalisierungsschätzern (bspw. Pedestrian Dead Reckoning, PDR, Methoden) integriert werden können.

II. Darauf aufbauend soll versucht werden, wie diese Basis hinsichtlich ihrer Robustheit, Latenz und Wiederverwendbarkeit zu optimieren ist.

Im Jahr 2018 konnten die Erkenntnisse aus I. (1-7) genutzt werden, um sogenannte (1) relative Pedes-

trian Dead Reckoning (PDR) Verfahren mit Hilfe von Bewegungsklassifizierern zu stabilisieren. Diese konnten eine Generalisierung auf beliebige Umgebungen ermöglichen. Das tiefere Funksignalverständnis (2) ermöglichte das Abbilden von Langzeitfehlern in RNN-basierten Bewegungsmodellen, um die Positionsgenauigkeit und Stabilität zu verbessern und nahe Echtzeit vorherzusagen. Die Robustheit der Bewegungsmodelle (3) konnte in ersten Versuchen mit Hilfe verschiedener realer (den Modellen unbekannter) Bewegungstrajektorien für ein- und zweiachsige Fahrzeuge gezeigt werden. Weiter wurde untersucht, (4) wie hybride Filtermodelle (bspw. Verschaltung von Merkmalsextraktoren 2D/3D-CNN und Zeitreihe RNN-LSTM) sowohl genauere, als auch stabilere und gefilterte (um Ausreißer korrigierte) Ergebnisse liefert.
Im Jahr 2019 wurde gezeigt, dass Modelle der RNN Familie in der Lage sind, Bewegungen in die Zukunft zu extrapolieren, so dass diese die Latenz der Verarbeitungskette und darüber hinaus kompensieren. Weiter wurde im Jahr 2019 die Erklärbarkeit, Interpretierbarkeit und Robustheit der hier untersuchten Modelle und die Wiederverwendbarkeit auf die menschliche Bewegung untersucht. Mit Hilfe eines Simulators wurden im Jahr 2019 physikalisch korrekte Bewegungen, z.B. Positionen von Fußgängern, Fahrradfahrern, Autos und Flugzeugen erzeugt. Auf Basis dieser Daten wurde gezeigt, dass RNN Modelle zwischen unterschiedlichen Bewegungstypen interpolieren können. Weiter wurde gezeigt, dass RNN Modelle fehlende Datenpunkte kompensieren, weißes und zufälliges Rauschen als solches interpretieren und Bewegungen in die Zukunft extrapolieren können. Letzteres ermöglicht die Kompensation von verarbeitungsspezifischer Latenz und ermöglicht eine Vorhersage der menschlichen Bewegung aus Funk- und Inertial-Daten in harter Echtzeit.
Neue RNN Architektur. Ferner wurde im Jahr 2019 eine neue Architektur, bzw. Topologie, eines neuronalen Netzes erforscht, welches die Stärken und Schwächen von flachen neuronalen Netzen und rekurrenter Netzen so kompensiert, dass eine optimales NN zur Bestimmung physikalisch korrekter Bewegung in einer großangelegten Parameterstudie gefunden werden konnte.
Architektur Optimierung. Es wurde im Jahr 2019 eine großangelegte Studie zur Optimierung der Modellparameter für die Mensch-zentrierte Lokalisierung durchgeführt. Diese optimalen Architekturen können die menschliche Bewegung aus möglichst wenig Sensorinformationen weit in die Zukunft voraussagen. Die Architektur mit dem geringsten Lokalisierungsfehler kombiniert zwei DNNs mit einem RNN.
Interpretierbarkeit von Modellen. Dieses neue Modell wurde im Jahr 2019 auf seine Funktionsweise untersucht. Dazu wurde eine neuartige Prozesskette zur Interpretation und Erklärung des Modelles erforscht. Die Prozesskette nutzt den Fluss der gegenseitigen Information und die gegenseitige Übertragungsentropie in Kombination mit verschiedenen gezielten Manipulationen der versteckten Zustände und geeigneten Visualisierungstechniken, um den Zustand des Modelles zu jedem Zeitpunkt zu bestimmen.
Darüber hinaus wurde im Jahr 2019, um extrahierte Merkmale eines neuronalen Netzes besser zu visualisieren und zu interpretieren, ein „Variational Auto-Encoder“ (VAE) adaptiert. Der VAE wurde so gestaltet und parametrisiert, dass der Rekonstruktionsfehler des Signals innerhalb des Messrauschens liegt und das Modell gleichzeitig gezwungen wird, entwirrte Merkmale im latenten Raum abzulegen. Dieses Entwirren ermöglicht erste subjektive Aussagen über die Zusammenhänge der Merkmale, die wirklich nötig sind, um den Kanalzustand eines Funksignals optimal zu kodieren.
Kompression. Dabei wurde im Jahr 2019 ein netter Seiteneffekt des VAEs entdeckt. Ein solcher VAE bietet die Möglichkeit der dezentralen Vorverarbeitung der Kanalinformationen direkt an der Antenne. Diese Komprimierung führt dann zu weniger Datenverkehr, erzeugt weniger Kommunikationslast und erhöht somit die Anzahl möglicher Teilnehmer an der Kommunikation und Lokalisierung in einem abgeschlossenen Sensornetz.
Einfluss der Variation der Eingabeinformationen. Weiter wurde im Jahr 2019 untersucht, wie sich Änderungen der Inputsequenzlänge eines rekurrenten neuronalen Netzes auf den Lernerfolg und die Art der Ergebnisse des Modells auswirken. Es wurde entdeckt, dass eine längere Sequenz das Modell überredet, eher ein Bewegungsmodell i.S.v. der Form der Bewegung zu erlernen, während kürzere Sequenzen dazu tendieren physikalische Zusammenhänge zu erlernen. Die höchste Genauigkeit erreicht man mit der

optimalen Balance zwischen kurzen und langen Sequenzen.
Es wurde im Jahr 2019 eine Geschwindigkeitsschätzung mittels des neuen Verfahrens untersucht. Diese floss dann direkt in ein PDR Modell ein, um die Positionsgenauigkeit zu erhöhen. Eine erste Arbeit im Jahr 2019 dazu hat im Detail untersucht, welche Verfahren am besten geeignet sind, um eine ungerichtete Geschwindigkeit der menschlichen Bewegung aus einem rohen Intertialsignal zu schätzen. Ein neues Verfahren, eine Kombination aus einem eindimensionalen CNN und einem BLSTM, hat hier den Stand der Technik abgelöst.
Im Jahr 2020 wurden die Modellarchitektur hinsichtlich der Vorhersagegenauigkeit optimiert und die Auswirkungen einer tiefen Kombination von Bayes'schen und DL-Methoden auf die Vorhersagegenauigkeit und Robustheit untersucht.
Optimierung. Im Jahr 2020 wurde die bestehende CNN- und RNN-Architektur verbessert und ein ResNet-BLSTM vorgeschlagen. Das CNN wurde durch ein Restnetzwerk ersetzt, um tiefere und qualitativ hochwertigere Merkmale aus einem kontinuierlichen Datenstrom zu extrahieren. Es konnte gezeigt werden, dass diese Architektur höhere Rechenkosten mit sich bringt, aber die genauesten, über den Stand der Technik hinausgehenden Ergebnisse liefert. Zusätzlich kann die RNN-Architektur verkleinert werden, um dem Ausbleichen des Kontextvektors der LSTM-Zellen entgegenzuwirken, da das verbleibende Netzwerk optimalere Merkmale bietet.
Tiefe Bayes Verfahren. Im Jahr 2020 wurde untersucht, ob Methoden der RNN-Familie bestimmte Bewegungseigenschaften aus aufgezeichneten Bewegungsdaten extrahieren können, um die starren Mess-, Rausch- und Übergangsverteilungen eines Kalman-Filters (KF) zu ersetzen. Eine Studie konnte zeigen, dass hochoptimierte LSTM-Zellen robustere (geringe Fehlervarianz der Prädiktionen) und präzisere (Positionsgenauigkeit) Trajektorien rekonstruieren können als ein ebenso hochoptimierter KF. Das tiefe Ineinandergreifen von LSTM in KF, sogenanntes tiefe Bayes Verfahren, lieferte die robustesten und präzisesten Positionen und Trajektorien. Diese Studie zeigte auch, dass von allen Methoden, die auf realistischen synthetischen Daten trainiert wurden, die tiefe Bayes-Methode am wenigsten reale Daten benötigt, um sich an eine neue unbekannte Domäne anzupassen.

III. Abschließend soll eine Demonstration der Machbarkeit erprobt werden.

Im Jahr 2018 wurde im Rahmen einer Großstudie mit sozialwissenschaftlichem Hintergrund das weltgrößte virtuelle Dinosauriermuseum eröffnet. Es konnte gezeigt werden, dass ein vorausgewähltes (auf das Einsatzszenario optimiertes) Bewegungsmodell die menschliche Bewegung robust und genau (i.S.v. kein signifikanter Einfluss auf die Simulatorkrankheit) abbilden resp. vorhersagen kann. Dieses wird als Basis für Vergleichstest für weitere Modelle (mensch-zentriert und generalisierend auf unterschiedliche Umgebungen) genutzt.
Im Jahr 2019 wurden auf Basis der erzielten Forschungsergebnisse in I und II zwei neue Live-Demonstratoren entwickelt. (1) Eine Modelleisenbahn, welche in variablen Geschwindigkeiten eine Landschaft mit Tunnel durchquert. Dabei repräsentiert der Tunnel realistische und typische Umgebungscharakteristika, die zu nichtlinearen Mehrwegeausbreitungen eines zu lokalisierenden Funksenders führen und letztendlich zu fehlerhaften Positionsbestimmung. Dieser Demonstrator zeigt, dass die im Rahmen des Forschungsprojektes erforschten RNN Verfahren sowohl auf komplexen Kanalimpulsantworten, als auch auf dimensionsreduzierten Antwortzeiten hochgenau und robust lokalisieren können und darüber hinaus bessere Ergebnisse als herkömmliche Kalman-Filter liefern. (2) Der zweite Demonstrator dient zur Visualisierung der Bewegung der oberen Extremität eines Menschen. Dabei wurde die menschliche Bewegung mit kostengünstiger Inertialsensorik erfasst, klassifiziert und Bewegungsparameter abgeleitet. Eine grafische Oberfläche visualisiert nahe Echtzeit die Bewegung und die abgeleiteten Parameter.
Die geplante Generalisierbarkeit, bspw. der mensch-zentrierten Modelle, und die Anwendbarkeit von RNN-basierten Verfahren in unterschiedlichen Umgebungen konnte mittels (1) und (2) demonstriert werden.
Im Jahr 2019 konnten folgende Anwendungen der vorgeschlagenen Methode beforscht und entwickelt

werden:
Anwendung: Funksignal. Es wurden die Kanalinformationen eines Funksystems hierarchisch derart klassifiziert, dass das Lokalisierungsproblem eines Line of Sight (LoS) und Non Line of Sight (NLoS) Klassifizierers in ein binäres Problem übertragen werden konnte. So kann rein auf Basis einzelner Kanalinformationen einer einzelnen Antenne eine Position auf einen Meter genau lokalisiert werden, wenn die Umgebung heterogene Kanalausbreitung breitstellt.
Ferner wurden LoS und NLoS Kanalinformationen simuliert und genutzt, um zwischen unterschiedlichen Kanälen zu interpolieren. Dies ermöglicht den Anbietern von Funksystemen, auf sich ändernde oder neue Umgebungen in den Kanalinformationen bereits a-priori in der Simulation einzugehen. Durch selektives Nachtrainieren der Modelle mit dem simulierten Wissen werden robustere Modelle ermöglicht.
Anwendung: Kamera- und Funksignal. Weiter konnte gezeigt werden, wie sich die RNN Methoden auf Informationen anderer Sensorfamilien, z.B. Videobilder, übertragen lassen. Eine Kombination von Funk- und Kamerasystemen ermöglichte es, ein Modell zu trainieren, welches selbst in Verdeckungsfällen der Kamera eine reibungslose Fusion der beiden Sensorinformationen schafft und zu einer robusteren und genaueren Lokalisierung mehrerer Personen führt.
Anwendung: Kamerasignal. In einer weiteren Arbeit wurde ein RNN-Verfahren genutzt, um die zeitlichen Zusammenhänge von Ereignissen in Bildern zu untersuchen. Im Gegensatz zu der vorangegangenen Arbeit, die heterogene Sensorinformationen nutzt, nutzt dieses Netz lediglich Bildinformationen. Das Modell nutzt die Bildinformationen allerdings so, dass es die Bilder unterschiedlich interpretiert: als räumliche Informationen i.S.v. ein einzelnes Bild, und als temporale Information i.S.v. mehrere Bilder im Input. Dieses Aufsplitten führt dazu, dass einzelne Bilder quasi als zwei fiktive virtuelle Sensorinformationsströme genutzt werden können, um Ergebnisse räumlich (Merkmale) zu erkennen und temporal (zeitliche Zusammenhänge) besser vorhersagen zu können.
Eine weitere Arbeit nutzt Kamerabilder, um die Kamera selbst zu lokalisieren. Dazu wurde eine neue Verarbeitungskette erschaffen, welche das Videosignal über die Zeit aufbricht und absolute und relative Informationen in unterschiedlichen neuronalen Netzen erlernt und deren Ausgabe in einem Fusionsnetz zu einer optimalen Pose zusammenführt.
Anwendung: EEG-Signal. In einem Kooperationsprojekt konnten die hier erforschten Methoden auf weitere Sensordaten angewendet werden. Es konnten Beta- und Gammawellen des menschlichen Gehirns in unterschiedlichen emotionalen Zuständen aufgezeichnet werden und diese von einem RNN genutzt werden, um die Emotionen einer Testperson in 90% aller Fälle aus rohen EEG Daten korrekt vorherzusagen. **Anwendung: Simulatorkrankheit.** In einer weiteren Arbeit konnte gezeigt werden, wie sich die Visualisierung in VR auf die menschliche Wahrnehmung und Bewegungsanomalien, respektive Simulatorkrankheit, auswirkt und wie sich die hier erforschten neuronalen Netze ausnutzen lassen, um die Effekte vorherzusagen.
Im Jahr 2020 wurden auf Basis der erzielten Forschungsergebnisse in II ein neuer Live-Demonstrator entwickelt.
Anwendung: Gangrekonstruktion in VR. Im Jahr 2020 wurde das vorhandene CNN-RNN-Modell verwendet, um die Bewegung des Menschen, nämlich den Gangzyklus und die Gangphasen, vorherzusagen. Dabei wurden Sensordaten eines am Kopf montierten Trägheitssensors verwendet, um einen virtuellen Avatar in VR in Echtzeit zu visualisieren. Es konnte gezeigt werden, dass das DL-Modell deutlich geringere Latenzen aufweist als der Stand der Technik, da es Gangphasen früher erkennen und zukünftige genauer vorhersagen kann. Dies geht jedoch zu Lasten des erforderlichen Rechenaufwands und damit der erforderlichen Hardware.

Das Projekt wurde im Jahr 2021 erfolgreich abgeschlossen. Im Jahr 2021 wurden im Rahmen einer erfolgreich abgeschlossenen Dissertation wesentliche Erkenntnisse aus dem Projektverlauf verknüpft und Schlussfolgerungen gezogen sowie zahlreiche Forschungsfragen aufgegriffen und beantwortet.

Im Rahmen des Forschungsvorhabens wurden mehr als 15 Qualifikationsarbeiten, 6 Patentfamilien und

mehr als 20 wissenschaftliche Publikationen erfolgreich abgeschlossen und veröffentlicht. Kernbeitrag des Projekts ist die Kenntnis der Anwendbarkeit und Tücken rekurrenter neuronaler Netze (RNN), ihrer unterschiedlichen Zelltypen und Architekturen in unterschiedlichen Anwendungsgebieten. Fazit: Die Möglichkeit der RNN-Familie, mit Dynamiken in Datenströmen umzugehen, z.B. Ausfällen, Verzögerungen und unterschiedlichen Sequenzlängen in Zeitreihendaten, macht sie heute in einer Vielzahl von Anwendungsbereichen unverzichtbar.

Das Projekt wird im Rahmen von Seminaren an der FAU und außeruniversitären Forschungsaktivitäten bei Fraunhofer IIS im Rahmen des ADA Lovelace Center fortgeführt.

3.7 SoftWater – *Software-Wasserzeichen*

Unter Software-Wasserzeichnen versteht man das Verstecken von ausgewählten Merkmalen in Programme, um sie entweder zu identifizieren oder zu authentifizieren. Das ist nützlich im Rahmen der Bekämpfung von Software-Piraterie, aber auch um die richtige Nutzung von Open-Source Projekten (wie zum Beispiel unter der GNU Lizenz stehende Projekte) zu überprüfen. Die bisherigen Ansätze gehen davon aus, dass das Wasserzeichen bei der Entwicklung des Codes hinzugefügt wird und benötigen somit das Verständnis und den Beitrag der Programmierer für den Einbettungsprozess. Ziel unseres Forschungsprojekts ist es, ein Wasserzeichen-Framework zu entwickeln, dessen Verfahren automatisiert beim Übersetzen des Programms Wasserzeichen sowohl in neu entwickelte als auch in bestehende Programme hinzufügen. Als ersten Ansatz untersuchten wir eine Wasserzeichenmethode, die auf einer symbolischen Ausführung und anschließender Funktionsynthese basiert.
Im Jahr 2018 wurden im Rahmen von zwei Bachelorarbeiten Methoden zur symbolischen Ausführung und Funktionssynthese untersucht, um zu ermitteln, welche sich für unseren Ansatz am Besten eignet.
Im Jahr 2019 wurde ein Ansatz auf Basis der LLVM Compiler Infrastruktur untersucht, der mittels konkolischer Ausführung (concolic execution, eine Kombination aus symbolischer und konkreter Ausführung) ein Wasserzeichen in einem ungenutzten Hardwareregister versteckt. Hierzu wurde der LLVM-Registerallokator dahingehend verändert, dass er ein Register für das Wasserzeichen freihält.
Im Jahr 2020 wurde das inzwischen LLWM genannte Rahmenprogramm für das automatische Einfügen von Software-Wasserzeichen in Quellcode auf Basis der LLVM Compiler Infrastruktur um weitere dynamische Verfahren erweitert. Grundlage der hinzugefügten Verfahren sind, unter anderem, das Ersetzen/Verschleiern von Sprungadressen sowie Modifikationen des Aufrufgraphen.
Im Jahr 2021 wurde das Rahmenprogramm LLWM um weitere angepasste, bereits in der Literatur bekannte, dynamische Verfahren sowie um das eigene Verfahren erweitert, das wir nun IR-Mark nennen Die hinzugefügten Verfahren basieren unter anderem auf der Umwandlung von bedingten Konstrukten in semantisch äquivalenten Schleifen oder auf Integrieren von Hashfunktionen, die die Funktionalität des Programms unverändert lassen, die Widerstandsfähigkeit aber erhöhen. IR-Mark wählt nun nicht nur gezielt die wenigen Funktionen aus, in denen die Registerverwendung bei der Code.Erzeugung verändert wird, sondern umfasst nun auch dynamische Aspekte um in den freigehaltenen Registern sinnvoll erscheinende Tarnwerte zu berechnen. Ein Artikel über LLWM und IR-Mark konnte publiziert werden.

3.8 CS4MINTS – *Informatik als Grundlage eines erfolgreichen MINT-Studiums entlang der Bildungskette fördern*

Die fortschreitende Digitalisierung verändert neben dem Arbeitsmarkt auch die Bildungslandschaft. Mit Mitteln aus dem DigitalPakt Schule werden u.a. im Zuge des Programms BAYERN DIGITAL II gravierende Veränderungen in der Informatikausbildung vorangetrieben, die wiederum neue Herausforderungen auf den verschiedenen Bildungsebenen mit sich ziehen. Das Projekt CS4MINTS begegnet eben diesen

Herausforderungen entlang der Bildungskette und knüpft an bereits im Rahmen des Projekts MINTer-AKTIV gestartete Maßnahmen, wie u.a. der Stärkung der Begegnung zunehmender Heterogenität der Studierenden im Zuge der Einführungsveranstaltung in die Informatik, an.

So wird im Zuge der **Begabtenförderung** das Frühstudium in Informatik aktiv für Mädchen beworben und das Angebot explizit erweitert. Durch frühzeitiges Vorgehen gegen genderspezifische Stereotypen bzgl. Informatik sowie eine Erweiterung des Fortbildungsangebots um gendersensitiven Informatikunterricht in allen Schularten, soll langfristig ein signifikanter Anstieg des Frauenanteils im Fach Informatik erreicht werden.

Durch die Erweiterung des Pflichtfachs Informatik in allen Schulen besteht außerdem ein großer Bedarf an geeigneten Unterrichtskonzepten und einer Stärkung der **LehrerInnenbildung**. Hierfür soll im Projektzeitraum ein regionales Netzwerk aufgebaut werden, um universitär erarbeitete und evaluierte Unterrichtsideen zur Stärkung von MINT im curricularen und extra-curricularen Rahmen zur Verfügung zu stellen.

Im Jahr 2020 haben wir mit der ersten Pilotierung der Konzeption zur Automatisierung des Feedbacks im Übungsbetrieb der Einführungsveranstaltung in die Informatik begonnen. Dazu wurden die Rückgabewerte der JUnit-Tests von Abgaben analysiert und erste mögliche Fehlerquellen untersucht. Im nächsten Schritt soll eine Möglichkeit ausgearbeitet werden, wie auf Grundlage dieser Rückgabewerte auf Programmierfehler oder Fehlvorstellungen der Studierenden geschlossen werden kann. Ziel dieser Bemühungen ist es schließlich, den Studierenden ein automatisch generiertes, kompetenzorientiertes Feedback zu ermöglichen, welches ihnen nach Abgabe der Programmieraufgaben (oder ggfls. schon während der Erarbeitungsphase) zur Verfügung steht. Das Feedback soll aufzeigen an welcher Stelle Fehler im Programmcode auftraten sowie auf mögliche Ursachen hinweisen.

Im Hinblick auf eine Begegnung von Heterogenität haben wir im Jahr 2020 die Inhalte des Repetitoriums für Informatik (RIP) dem bayerischen Lehrplan verschiedener Schularten gegenübergestellt. Die Inhalte sollen anschließend so angepasst werden, dass StudienanfängerInnen unterschiedlichster Bildungszweige gleiche Chancen haben, mittels des Repetitoriums mögliche Defizite zu erkennen und diese zu beheben. Außerdem wurde im Wintersemester 2020 während der Repetitoriumswoche erstmalig eine tägliche Programmiersprechstunde eingerichtet, in der es den TeilnehmerInnen möglich war Fragen zu stellen sowie Feedback zu den Aufgaben zu erhalten.

Der Einstieg in die Programmierung stellt für viele Studierende eine der großen Hürden zu Beginn des Studiums dar. Um den Programmieranfänger:innen zusätzliches Feedback zu ermöglichen, haben wir im Jahr 2021 das Projekt Feedback+ konzipiert und pilotiert. Im Rahmen von Feedback+ haben die Studierenden die Möglichkeit, Probleme, welche während der Bearbeitung der Übungsaufgaben oder beim Einrichten/Benutzen der Programmierumgebung auftreten, zu dokumentieren und in wöchentlich buchbaren Einzelsprechstunden zusätzliches Feedback zu erhalten. Dazu wurde eine StudOn-Umgebung eingerichtet, in der Probleme systematisch dokumentiert werden können. Eine erste Evaluation in Form von Einzelinterviews mit den teilnehmenden Studierenden lieferte ein durchweg positives Feedback und einen Wunsch auf Fortsetzung des Projektes.

4 Lehre

Der Lehrstuhl für Programmiersysteme bietet im Wintersemester das Pflichtmodul *Algorithmen und Datenstrukturen (AuD)* und im Sommersemester *Parallele und Funktionale Programmierung (PFP)* an. Da diese Module fakultätsübergreifend auch anderen Studiengängen (insbesondere Wirtschaftsinformatik bzw. International Information Systems, Informations- und Kommunikationstechnik, Mathematik u.v.a.)

angeboten werden, erreichten die Hörerzahlen mit 545 (AuD im WS2020/21) bzw. 359 (PFP im SS2021) im Berichtszeitraum erneut nahezu den Rekordwert der Vorsemester, die sich schließlich auch in der hohen Zahl an Prüfungsanmeldungen (411 in AuD bzw. 363 in PFP) niederschlagen. In der Vertiefungsrichtung Programmiersysteme bietet der Lehrstuhl verschiedene Module zu den Themen *Übersetzerbau*, *Clustercomputing* und *Testen von Softwaresystemen* an. Die Seminare *Hallo Welt! für Fortgeschrittene* und *Machine Learning* waren erneut innerhalb kürzester Zeit restlos ausgebucht.
Insgesamt betreute der Lehrstuhl für Programmiersysteme im Berichtsjahr drei Masterarbeiten und drei Bachelorarbeiten.

ICPC – *International Collegiate Programming Contest an der FAU*: Seit 1977 wird der International Collegiate Programming Contest (ICPC) ausgetragen. Dabei sollen Teams aus je drei Studierenden ca. 13 Programmieraufgaben lösen. Als Erschwernis kommt hinzu, dass nur ein Computer pro Gruppe zur Verfügung steht. Die Aufgaben erfordern solide Kenntnisse von Algorithmen aus allen Gebieten der Informatik und Mathematik, wie z.B. Graphen, Kombinatorik, Zeichenketten, Algebra und Geometrie. Bei der Lösung kommt es darauf an, einen effizienten und richtigen Algorithmus zu finden und zu implementieren.
Der ICPC wird jedes Jahr in drei Stufen ausgetragen. Zuerst werden innerhalb der Universitäten in lokalen Ausscheidungen die maximal drei Teams bestimmt, die dann zu den regionalen Wettbewerben entsandt werden. Erlangen liegt seit dem Jahr 2009 im Einzugsbereich des Northwestern European Regional Contest (NWERC), an dem u.a. auch Teams aus Großbritannien, den Benelux-Staaten und Skandinavien teilnehmen. Die Sieger aller regionalen Wettbewerbe der Welt (und einige Zweitplatzierte) erreichen die World Finals, die im Frühjahr des jeweils darauffolgenden Jahres (2022 in Dhaka, Bangladesh) stattfinden.
Der NWERC 2020 wurde Corona-bedingt auf März 2021 verschoben und rein virtuell von Island aus organisiert. Unsere Teams erreichten die Plätze 24, 42 und 50 von 120 teilnehmenden Teams. Im Sommersemester 2021 fand vor dem Wettbewerb zum wiederholten Mal das Hauptseminar "Hallo Welt! - Programmieren für Fortgeschritteneßtatt, um Studierenden verschiedener Fachrichtungen Algorithmen beizubringen und mit ihnen Wettbewerbsaufgaben zu trainieren. Am 26. Juni fand der German Programming Contest 2021 (erneut) als reiner Online-Contest statt, an dem 85 Teams aus ganz Deutschland teilnahmen. Das beste von 18 Teams (44 Teilnehmer) aus Erlangen erreichte Platz 10.
In November fand der NWERC 2021 virtuell statt (nochmals von Island aus organisiert). Das FAU-Team erreichte Platz 49 von 129. Der nächste NWERC findet wieder in den Niederlanden statt und der GCPC im kommenden Sommer dient zur Auswahl der Teilnehmer.

5 Publikationen

[1] Thomas Robert Altstidl, Sebastian Kram, Oskar Herrmann, Maximilian Stahlke, Tobias Feigl, and Christopher Mutschler. Accuracy-Aware Compression of Channel Impulse Responses using Deep Learning. In *Proceedings of the International Conference on Indoor Positioning and Indoor Navigation (IPIN 2021)*, pages 1–8. IEEE Xplore, 2021. doi:10.1109/IPIN51156.2021.9662545.

[2] Veronika Dashuber and Michael Philippsen. Cloud Cost City: A Visualization of Cloud Costs Using the City Metaphor. In Christophe Hurter, Helen Purchase, Jose Braz, and Kadi Bouatouch, editors, *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP) - Volume 3: IVAPP*, pages 173 – 180, Portugal, 2021. SciTePress. doi:10.5220/0010254701730180.

[3] Veronika Dashuber and Michael Philippsen. Trace Visualization within the Software City Metaphor: A Controlled Experiment on Program Comprehension. In *Proceedings of the IEEE Wor-*

king Conference on Software Visualization (VISSOFT), pages 55–64. IEEE, 2021. doi:10.1109/VISSOFT52517.2021.00015.

[4] Veronika Dashuber, Michael Philippsen, and Johannes Weigend. A Layered Software City for Dependency Visualization (Best Paper Award). In Christophe Hurter, Helen Purchase, Jose Braz, and Kadi Bouatouch, editors, *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP*, pages 15 – 26, Portugal, 2021. SciTePress. URL: http://www.ivapp.visigrapp.org, doi:10.5220/0010180200150026.

[5] Tobias Feigl. *Datengetriebene Methoden zur Bestimmung von Position und Orientierung in funk- und trägheitsbasierter Koppelnavigation*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2021. URL: https://nbn-resolving.org/urn:nbn:de:bvb:29-opus4-173550.

[6] Tobias Feigl, Ernst Eberlein, and Sebastian Kram. Robust ToA-Estimation using Convolutional Neural Networks on Randomized Channel Models. In *Proceedings of the International Conference on Indoor Positioning and Indoor Navigation (IPIN 2021)*, pages 1–8. IEEE Xplore, 2021. doi:10.1109/IPIN51156.2021.9662625.

[7] Marius Kamp and Michael Philippsen. Approximate Bit Dependency Analysis to Identify Program Synthesis Problems as Infeasible. In Yakir Vizel Fritz Henglein, Sharon Shoham, editor, *Verification, Model Checking, and Abstract Interpretation (VMCAI 2021)*, volume 12597 of *Lecture Notes in Computer Science (LNCS)*, pages 353 – 375, Cham, Jan 2021. Springer International Publishing. URL: https://i2git.cs.fau.de/i2public/publications/-/raw/master/vmcai2021.pdf, doi:10.1007/978-3-030-67067-2_16.

[8] Patrick Kreutzer, Tom Kunze, and Michael Philippsen. Test Case Reduction: A Framework, Benchmark, and Comparative Study. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME 2021)*, pages 58–69. IEEE, 2021. URL: https://i2git.cs.fau.de/i2public/publications/-/raw/master/ICSME21.pdf, doi:10.1109/ICSME52107.2021.00012.

[9] Florian Mayer, Julian Brandner, Matthias Hellmann, Jesko Schwarzer, and Michael Philippsen. The ORKA-HPC Compiler — Practical OpenMP for FPGAs. In *Proceedings of the 34th International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, 2021. URL: https://lcpc2021.github.io/pre_workshop_papers/Mayer_lcpc21.pdf.

[10] Steffen Meyer, Thomas Windisch, Adrian Perl, Daniel Dzibela, Robert Marzilger, Nicolas Witt, Justus Benzler, Göran Kirchner, and Tobias Feigl. Contact Tracing with the Exposure Notification Framework in the German Corona-Warn-App. In *Proceedings of the International Conference on Indoor Positioning and Indoor Navigation (IPIN 2021)*, pages 1–8. IEEE Xplore, 2021. doi:10.1109/IPIN51156.2021.9662591.

[11] Daniela Novac, Christian Eichler, and Michael Philippsen. LLWM & IR-Mark: Integrating Software Watermarks into an LLVM-based Framework. In *Checkmate '21: Proceedings of the 2021 Research on offensive and defensive techniques in the Context of Man At The End (MATE) Attacks*, pages 35–41, New York, 2021. ACM. doi:10.1145/3465413.3488576.

[12] Maximilian Stahlke, Sebastian Kram, Felix Ott, Tobias Feigl, and Christopher Mutschler. Estimating TOA Reliability with Variational Autoencoders. *IEEE Sensors Journal*, pages 1–6, 2021. Created from Fastlane, Scopus look-up. doi:10.1109/JSEN.2021.3101933.