UNIVERSITÄT BIELEFELD

Studies in Analytical Reproducibility: the Conquaire Project

Philipp Cimiano, Christian Pietsch, Cord Wiljes (Eds.)

Bielefeld University

Studies in Analytical Reproducibility: the Conquaire Project

Editors:

Philipp Cimiano, Christian Pietsch, Cord Wiljes

2020

Acknowledgements

This work is part of the Conquaire project, which is funded by the German National Science Foundation (DFG). DFG-funded research project number 277747081 (German Abstract): http://gepris.dfg.de/gepris/projekt/277747081
This work was supported by the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277) at Bielefeld University, which is funded by the German Research Foundation (DFG)."

Funded by
DFG Deutsche Forschungsgemeinschaft
German Research Foundation

Cimiano, P., Pietsch, C., Wiljes, C. eds: Studies in Analytical Reproducibility: the Conquaire Project. Bielefeld (2020).

ISBN 978-3-943363-06-7
DOI https://doi.org/10.4119/unibi/2942780

Philipp Cimiano https://orcid.org/0000-0002-4771-441X
Christian Pietsch https://orcid.org/0000-0001-8778-1273
Cord Wiljes https://orcid.org/0000-0003-2528-5391

License

Contents

Preface

This book is a direct result of the *Conquaire* (Continuous Quality Control for Research Data to Ensure Reproducibility) project, which was funded by the DFG between 2016 and 2019. The goal of the project was to understand in how far principles from continuous quality control and test-driven development as nowadays being state-of-the-art in software engineering can be applied to the management of research data to increase its quality and potential for re-use.

In order to arrive at such an understanding, we have been closely working together with researchers from different disciplines at Bielefeld University ranging from biology, over chemistry, economics, linguistics, psychology through to computer science / robotics. All in all, we have been working with eight research groups and have defined a case study in reproducibility with each of these groups. Within these use cases we have aimed at reproducing one central part of a previously published research article. In doing this, we have limited ourselves to reproducing the computational analysis leading to the particular result, as reproducing the actual experiments would have been outside of the scope of the Conquaire project.

The book that lies in front of you documents these eight case studies and describes what we have done to reproduce the specific results. In most cases, reproducing the analytical result, in spite of data and scripts being available, has only been possible by close interaction and guidance by the authors of the original publication, which in all cases are direct co-authors of the chapter describing our reproducibility experiments.

The work conducted in the case studies has provided us with a detailed understanding of the analytical workflows used by all the case study partners and has allowed us to get a deep understanding of barriers and challenges in reproducing published results. Thus, we can give a number of clear recommendations at the end of the book, representing the lessons learned from the practical attempt to reproduce a number of published results.

This exercise in understanding requirements and problems for analytical reproducibility would not have been possible without funding by DFG. Most critically, it would not have been possible without the effort and dedication of the eight research groups we have worked with for the last three years. We would like to thank all of them for their patience with us and for bearing with us while walking on the sometimes stony path of achieving reproducibility. We thank all of the research groups for providing us data, scripts, describing their workflows, etc. All of these groups have been engaged in this project because they were interested in finding how to improve their workflows to make their results trans-

parent, reproducible and thus better accessible to the scientific community. We thank all of these groups for engaging in the project in spite of the risk that comes with a higher level of transparency and exposure. By being transparent, one risks that others can discover some flaws in the way things have been done. This is quite a risk in science, a risk that nevertheless we have to take as progress in science should always be weighted higher than the consequences for particular individuals.

We would like to thank all the student researchers involved in Conquaire who have supported the activities of reproducing results. We would like to thank in particular Lukas Biermann and Fabian Herrmann as they have been central to the success of many case studies, having worked day-to-day with many of the above mentioned research groups and having developed central pieces of the Conquaire infrastructure for supporting continuous quality control of research data. We would also like to thank Vidya Ayer, who has been working on the project since its start. She has been key in pulling together the different chapters that this book consists of and provided a very early draft version of a manuscript for the book. Finally, we would like to thank John P. McCrae for contributing to the Conquaire project proposal. Many of the key ideas of Conquaire go back to him.

It has been a pleasure and very rewarding to work with all these scientists and learning about their very specific research questions, goals, and methods. We hope that you find this book as exciting to read as it was for us to edit it.

Bielefeld, 29th September 2019

Philipp Cimiano, Christian Pietsch, Cord Wiljes

1 | Introduction

Abstract

Data quality, FAIR-ness and reproducibility are important criteria to fulfill in the research lifecycle. They allow scientists to validate each others' results and are thus key to the integrity of science. Within the DFG-funded Conquaire project, we have strived to develop a detailed understanding of the challenges involved in ensuring data quality, achieving FAIR-ness and computational reproduction of results. As a way to derive requirements and lessons learned, we have analyzed in detail the research practices and workflows of a number of research groups participating in the Conquaire project from different disciplines. On the basis of these case studies, we have identified categories of reproducibility and attempted to highlight challenges and propose best practices. This chapter briefly introduces the goals of the Conquaire project which has focused on the reproduction of the so called analytical phase of an experiment. We briefly introduce the research disciplines and research groups involved and present a summary of our overall findings. The remainder of the book describes the different use cases in detail and concludes with some recommendations for best practices.

1.1 Motivation

Reproducibility is a cornerstone of the scientific process. While the reproduction of an experiment can be extremely difficult, the ability to reproduce the (computational) analysis of the data that supported a certain conclusion (e.g. the validation of a hypothesis) should be a minimum requirement on every piece of published research. We call this type of reproducibility *analytical reproducibility.*

An illustrative example of data sharing and independent validation of scientific results by others, or in this particular case refutation, is conveyed by a recent case. In 2010, Harvard researchers Kenneth Rogoff and Carmen Reinhart published their paper *"Growth in a Time of Debt"* [9]. They had analyzed historical data from 20 industrialized countries since WWII and concluded that economic crises arise when the size of a country's debt rises above 90% of the Gross Domestic Product (GDP). This result had a significant impact on political decisions worldwide, until in 2013 the student Thomas Herndon tried to replicate the analysis. He contacted Reinhart and Rogoff and they provided him with the actual working spreadsheet. Inspection of this spreadsheet disclosed several serious errors, which rendered the results invalid[1]. The above case is thus a very good example of the efficiency of the scientific process, albeit a rather simple one in which both the data and the analytical procedures used to analyze the data were available in the form of a spreadsheet and could thus be directly analyzed and modified. In many scientific disciplines, reproducing research results is more complicated as research can involve advanced and high-tech devices needed to measure certain phenomena, complex experimental protocols, data in different formats (structured vs. unstructured), different modalities (text, annotations, video, audio, 3D data) etc., which make it difficult to reproduce a certain scientific experiment.

While reproducing an experiment can be very challenging, as a baseline, given the primary or derived data resulting from an experiment, the reproduction of the (computational) analysis procedures that yielded a particular result should be feasible. In fact, an important step in the generation of scientific results lies in the computational analysis of the primary data or derived secondary data. In most cases, software packages (such as SPSS, R, Excel) are used in this part of the process to test a hypothesis by performing some computational or statistical analyses of the primary or derived data. We will refer to this part of the process as *analytical phase*. While being able to fully reproduce an experiment can be extremely difficult, reproducing the analytical phase seems more feasible as it would essentially require access to the primary and/or derived data as well as to the analytical tools used by the researchers to derive some result.

Thus, a significant step forward towards supporting reproducibility in science would be *analytical reproducibility*, which consists of making sure that a third party researcher is able to reproduce the computational and statistical analysis performed on primary and derived data to yield a particular conclusion, thus being able to independently verify the results and conclusion. A crucial question is how research data infrastructures should be extended to support *analytical reproducibility*, *data sharing* and thus *independent validation* of (analytical) research results.

As a prerequisite for a research result or scientific paper to be analytically reproducible and useful for other researchers, the following conditions need to be met:

1. the primary or secondary data is available,

2. the data is syntactically well-formed and ready-to-use,

3. the data is appropriately documented,

4. the analysis procedures (e.g. scripts) that were used to process or analyze the data are available, and

[1]See the correction published by C. Reinhart on her website: `http://www.carmenreinhart.com/user_uploads/data/36_data.pdf`

5. these analytic procedures can be run on the data to reproduce the actual result published in a paper.

Analytical reproducibility is often hampered by the fact that one of the above conditions is not met. In order to be reproducible, research data needs to have a certain quality which we operationalize in the context of Conquaire as its readiness to be re-used by others, e.g. to reproduce the computational analyses described in a scientific publication. Typically, researchers have no institutional support nor resources to ensure data quality in the above sense. Thus, curating data to make it fit for publication and sharing requires substantial resources that researchers do not typically receive credit for. If the data is published, this is typically done in a delayed fashion after the research project or dissertation work has been concluded.

However, it is well-known from other areas (e.g. software engineering) that quality control is better taken into account at the start of the research project. Continuous integration [1] (p. 209) in software engineering is aimed at increasing quality of software by specifying a number of tests that the software should pass in order to continuously monitor compliance with these. Drawing inspiration from software engineering, principles of continuous integration could be applied to research data management to realize continuous monitoring of data quality and ensure that at each step in the research cycle, the data fulfills a number of defined tests. Ultimately, as a final test on the quality of the research data, the proof that third parties can reproduce and validate the (computational) analyses that produced a certain result could be seen as final culmination of a continuous data quality assurance cycle.

In our previous research data management efforts, we learned that researchers are generally willing to create data of high quality, share this data and make their results reproducible as part of their duties as a researcher and to meet expectations of their community. However, to ensure reproducible research, extensive effort [2], maintenance and documentation is required, which rarely happens due to the demanding challenges related to data processing, validation and publishing. Therefore, researchers need to be supported in this process by an appropriate institutional infrastructure that hosts their data and implements corresponding workflows that allow them to ensure research data quality and reproducibility along the whole research lifecycle. Such an infrastructure that supports continuous research data quality monitoring and at the end makes the data publicly available to allow for reproduction of the computational analysis is not yet available. The DFG-funded Conquaire project had the goal of examining case studies as a basis to develop best practices and prototypically implement institutional support to ease the work of scientists in making their work reproducible by third parties.

1.2 Overview of Conquaire Infrastructure and Workflow for Research Data Management

For an academic institution, it is important to have an infrastructure-based approach when creating research data management services for researchers. Within the Conquaire project, we have developed a number of best practices and workflows that support researchers in ensuring reproducibility and quality of their research data, concentrating on the analytical phase of an experiment.

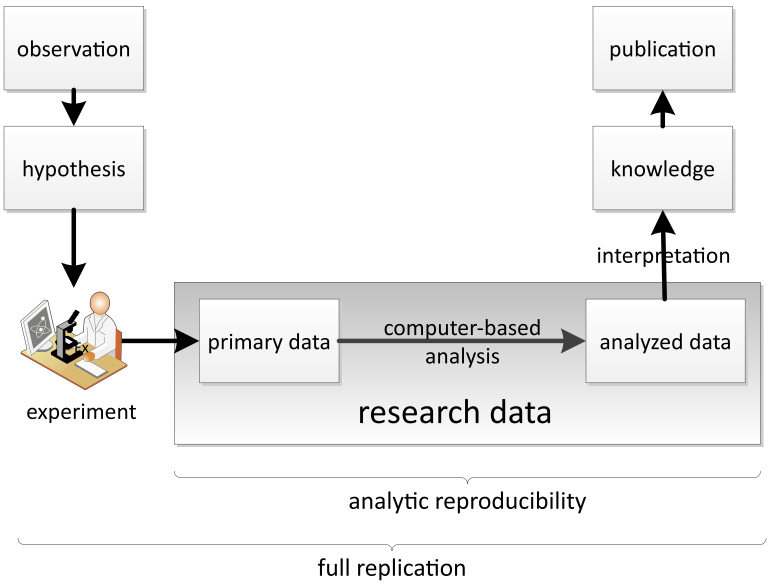


Figure 1.1: Schematic description of Conquaire workflow for research data management

Conquaire envisions that scientists commit their data and scripts early in the research cycle into a distributed version control system (DVCS) such as Git, a content-addressable key-value data store based file system. A University-wide installation offers various advantages for collaboration: regular data backups that are version controlled, hence retrievable, and security features for data that cannot be corrupted.

The Conquaire project decided to adopt Git as the DVCS , largely to take advantage of features that ensure a distributed collaborative environment. GitHub, a social site for software development, uses the Git DVCS as the underlying technology to create a cloud-hosted platform for sharing program code and related technical artifacts. With several collaborative features, the site is free for open-source projects and the intrinsic social features make it very popular among programmers, scientists and technical people wanting to share their work and collaborate. A Stackoverflow survey[2] ranked Git usage at 69.3%, almost double than the second source control - SVN at 36.9%, making Git the front runner among distributed version control systems.

Since GitHub is a cloud-hosted platform, we looked for alternative free and open source software (FOSS) implementations that could be installed on the University infrastructure. We found Gitlab, a free software framework implementation of a web-based Git-repository manager that supports self-hosting with features similar to GitHub[3], i.e. an issue-tracker, wiki, CI/CD pipeline, etc. that was layered around the user with different permission levels. These variable permission layers for different feature access plays an important role in collaborating and sharing knowledge across physical boundaries. Like GitHub, the collaborative features of Gitlab include allowing a user to make multiple *commits*, *pull requests*, make changes and edit their documents, create forks or branches, revert to an old version, and/or merge those changes into the *master* branch.

When a user makes a Git commit, it consists of three steps that involve Git creating, (i) a tree graph in order to represent the content of the files being committed to the project, (ii) a commit object that is stored and tracked in the **.Git/objects** folder, and (iii) an object that points to the current branch at the new commit object.

To record the current state of the repository, Git creates a tree graph from the index, which records the location and content of every file within the project repository. The tree graph is composed of two types of objects: *blobs* and *trees*. The command *Git add* stores *blobs* that represent the content of files; while *trees* are stored when a *commit* is made and it represents a directory in the working copy.

Thus, the distributed features of the key-value data store ensure that the Git history stores the old version, the new version, and an interim version that the user stores in their forked (or, working copy) version. The Git project environment aids data sharing and reproducibility when a user checks research data into a version controlled repository, by ensuring they can reproduce the exact state of the project over a timeline. Thus, multiple users can easily collaborate without fear of their work being erased or overwritten thanks to many Git features for collaboration like merging, fetching, pulling changes to a local branch, branching, stashing, pushing changes, tagging objects, etc.

Conquaire Workflow

The architecture of the Conquaire quality control system is depicted in Figure 1.2, where the workflow consists of two integrated streams:

[2]`https://insights.stackoverflow.com/survey/2015`

[3]`https://conquaire.uni-bielefeld.de/2018/04/17/Git/`

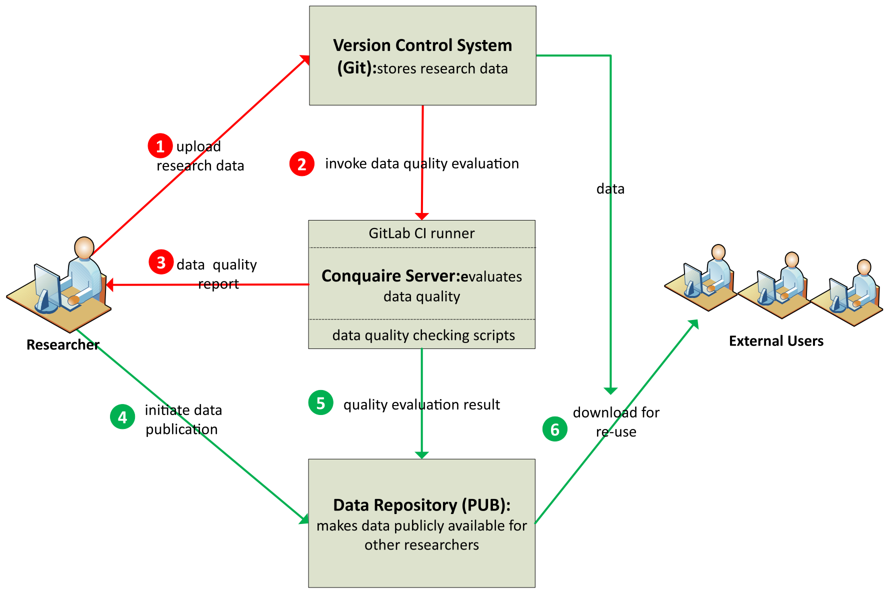

Version Control System (Git): stores research data
1 upload research data
2 invoke data quality evaluation
data
GitLab CI runner
Conquaire Server: evaluates data quality
data quality checking scripts
3 data quality report
Researcher
External Users
4 initiate data publication
5 quality evaluation result
6 download for re-use
Data Repository (PUB): makes data publicly available for other researchers

Figure 1.2: Research data acquisition and processing pipeline.

A. Data preparation and quality checking (marked red)

- **Step 1:** The researcher uploads data to the version control system server. This can be done by the GitLab Browser-based frontend, from the shell using Git commands or with any other available Git-GUI (e.g. GitHub Desktop, Tortoise Git).

- **Step 2:** Uploading one or more files onto the Git-Server automatically triggers the Gitlab CI-runner, which executes the quality checking procedures on the Conquaire quality checking server. These fetch the necessary files from the Git-repository and perform quality checks.

- **Step 3:** The result of the quality check is returned to the researcher. It gives a detailed analysis of all files that were commited and provides a report on which tests were passed or failed by the data. The researcher may then correct the data according to the test results and resubmit it to the Git-repository. This cycle can be iterated as long as it is necessary.

B. Data publication und Re-Use (Steps 4-6, marked green)

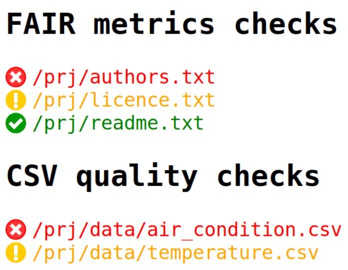
- **Step 4:** If a researcher decides to publish the data, she/he can choose to do this on the publication repository PUB[4], hosted by Bielefeld University, which allows for data publication and has a direct interface to Gitlab. For the publication of a dataset, the researcher only needs to enter the URL of the GitLab repository and some basic metadata (creator, year, license) into the PUB interface.

- **Step 5:** PUB will automatically fetch the results of the Conquaire quality checks from the Conquaire server, and select and display the results visually via quality badges.

- **Step 6:** If a visitor wishes to download the data, it is fetched from GitLab.

An important dimension of the workflow/architecture consists in automatic quality checks that can be applied to research data. Every time the researcher commits changes to the Git repository, the in-built CI runner in Gitlab automatically executes a pipeline that is described in a YAML file inside the root directory of the project. This pipeline performs different tests that check whether all the necessary background information is provided (FAIR metrics[5]) and the data has valid syntax and semantics. The quality checking tests are implemented in Python-3x and cover two open file types that are commonly used, i.e., **.csv** and **.xml**, with the possibility of an extension to work with other file types as well.

In each case, the quality testing framework searches for specific file types and checks if the files can be parsed and whether the data fulfills some predefined criteria. The researcher must provide **.fmt** or **.dtd** files with their own specifications (i.e., data value types and ranges) to evaluate the quality of their data. For every file and every commit into their Git repository, a log file (bearing the commit hash) is created that lists errors and warnings that were reported by the test.



FAIR metrics checks

✗ /prj/authors.txt
! /prj/licence.txt
✓ /prj/readme.txt

CSV quality checks

✗ /prj/data/air_condition.csv
! /prj/data/temperature.csv

Figure 1.3: Example feedback from quality tests.

[4]https://pub.uni-bielefeld.de/
[5]http://fairmetrics.org/

At the end, the feedback is provided directly to the researcher, helping them to find errors and correct them right away before publishing the results. The feedback is stored on the server and an e-mail with an overall summary of the test results is sent to the researcher. In addition to the personal feedback, a colored badge icon in the PUB entry is created, representing the overall quality to other researchers who want to work with the data. The badge is based on the worst test result across the repository and has three possible categories: valid data (green), well-formed data (yellow), and erroneous data (red). This is to encourage researchers to make sure that only high-quality data is published to guarantee reproducibility.

We have applied the the workflow mentioned before to a number of specific case studies from different disciplines in Conquaire. The case studies have been instrumental in validating the concept and workflow proposed by Conquaire but also in understanding the practical feasibility, obstacles, etc. of the workflow when applied to concrete cases. In these case studies, we aimed for reproducing one particular result from an already published paper. This historical perspective might look a bit odd at first sight.

However, we realized quickly that we would not be able to change the current workflows followed by researchers in the middle of a research project as this would be too disruptive and would delay their research process, a price that none of the groups involved in Conquaire as case study partners would be willing to pay. Therefore, with each of these case study partners, we identified one central result that Conquaire would try to reproduce. This book describes all these case studies and derives lessons learned and makes recommendations on best practices to follow to ensure reproducibility of analytical workflows.

1.3 Case Studies in Computational Reproducibility

Here, we briefly describe the eight case studies analyzed in Conquaire and in particular the main results that were reproduced from the published paper for each research group.

Biological Cybernetics: The **Biological Cybernetics** research group at Bielefeld University led by Prof. Volker Dürr researches the adaptive locomotion abilities of stick and leaf insects. In collaboration with the group, we attempted to reproduce the main results of the paper by Theunissen et al. [6] with the title *'Comparative whole-body kinematics of closely related insect species with different body morphology'*. In this paper, the authors investigated the walking behaviour of three different species of stick insects. This was done by recording whole-body kinematics of the animals, using a commercial marker-based motion capture system and custom written MATLAB scripts. The main objective of the study was to relate inter-species differences in kinematics to differences in overall morphology, including features such as leg-to-body-length ratio that were not an obvious result of phylogenetic or ecological divergence. The researchers discovered major differences related to antenna length, segment lengths of thorax and head, and the ratio of leg length over body length, with the long-legged Medauroidea having the strongest difference in intra-leg coordination of multiple joints, leg posture, and time courses of leg joint angles. We discuss an experiment in reproducing the main results of this paper in Chapter 3.

Neurobiology: The **Neurobiology Research Group** at Bielefeld University led by Prof. Martin Egelhaaf works on understanding flight and navigation behaviour of bumblebees. The goal was to reproduce results from a paper [7] with the title *'Taking a goal-centered dynamic snapshot as a possibility for local homing in initially naive bumblebees'*, studying the first learning flights of bumblebees, which are highly individual and variable. The Conquaire project reproduced the analytical workflows and was able to reproduce the 3D trajectory generated by using triangulation from two 2D trajectories. This case study is reported in more detail in Chapter 4.

Atmospheric Chemistry: The **Physical Chemistry** group at Bielefeld University led by Prof. Thomas Koop investigates the conditions under which ice nucleation occurs. In this book, we describe a case study consisting in the replication of the central result in the paper *'BINARY: an optical freezing array for assessing temperature and time dependence of heterogeneous ice nucleation'* by Budke and Koop [3]. The study investigates the conditions under which ice nucleation occurs using Snomax®, a commercial ice inducer containing freeze-dried nonviable bacterial cells from Pseudomonas syringae as a test substance for the investigation of heterogeneous ice nucleation processes. The main result of the paper was an analysis of the relation between the ratio of nucleations in dependence of different temperature ranges. The original analytical workflow was implemented via the GUI of OriginPro. In chapter 5, we describe how this workflow could be reproduced using a Python script.

Psycholinguistics: In our collaboration with the **Psycholinguistics** research group at Paderborn University led by Prof. Katharina Rohlfing, we attempted to independently reproduce the main findings of the paper by Namikou et al. with the title *'Evidence for early comprehension of action verbs'* [8]. The reproduced study adopted a preferential looking time paradigm and conducted a so called paired-picture trial in which a verb under investigation was semantically associated to one of two pictures shown, the target picture, and another picture, the so called *'confounder'*. Using an eye tracker, the difference between proportion of looking times at the matching image before the verb was spoken compared to looking times after the verb was spoken was measured. As a result, the study showed positive differences for 10-month olds, sign of early understanding of the verbs. For 9-month olds, in contrast, the study was not able to reliably demonstrate verb understanding. We describe this reproduciblity experiment in Chapter 7.

Applied Computational Linguistics: The Applied Computational Linguistics at Bielefeld University is led by Prof. David Schlangen and performs research in the area of dialogue systems. Within Conquaire, our goal was to reproduce the main results of the published paper *'Joint, Incremental Disfluency Detection and Utterance Segmentation from Speech'* [5], published in the proceedings of the international conference on the European chapter of the Association for Computational Linguistics (EACL). This paper was concerned with the task of disfluency detection and utterance segmentation and proposed a simple deep learning system working on transcripts and Automatic Speech Recognizer (ASR) output. For this purpose, the Dialogue Systems Group at Bielefeld University developed a library that relies on a mixed data model of live *automatic speech recognition (ASR)* data and a cleaned text corpus of open data to evaluate the performance of deep learning systems for disfluency detection in conversational systems and related tasks on speech data. We describe our experiments with this library on reproducing the results of the above mentioned paper in Chapter 8.

Neuro-Cognitive Psychology With the **neuro-cognitive psychology** research group at Bielefeld University, we reproduced the results for their published manuscript entitled *'Expectation violations in sensorimotor sequences: shifting from Long-Term Memory (LTM)-based attentional selection to visual search'* [4]. The research of the group focuses on the area of visual attention, eye movements, working memory, transsaccadic learning, and sensorimotor learning. The group works on understanding visual processing in humans via controlled behavioral experiments in laboratory environments alongside real-world studies. The main result of the article mentioned above was that the finding that expectation violations in a well-learned sensorimotor sequence in humans caused a regression from LTM-based attentional selection to visual search. We describe our efforts to partially reproduce these experimental results in Chapter 9.

Economic Theory And Computational Economics: The Economic Theory And Computational Economics (ETACE) group lead by Prof. van Hoog applies agent modeling approaches to study dynamic equilibrium models resulting from the interaction of heterogeneous rational agents, allowing insights into the application of different industrial policy measures in different regions, the existence of varying spatial frictions on goods and labour markets, the spatial dynamics of industrial activity, technical change and growth, the micro- and macro-prudential regulations and their effects on micro-fragility and macro-financial stability, and financiation of the real sector and the need for productive credit for economic development. Conquaire supported the ETACE group in implementing the FLAViz library that implements a data analytic processing pipeline allowing the computational analysis and visualization of simulation data generated in the FLAME environment. This library is a key step towards ensuring computational reproducibility of the analyses of the available simulation data; we describe it in more detail in Chapter 6.

Robotics and Cognitive Systems Research: The Central Facility Lab group led by Dr. Wachsmuth at the Cognitive Interaction Technology Excellence Center at Bielefeld University is concerned with aspects of system and software engineering in robotic and cognitive systems. Within the context of Conquaire, we attempted to reproduce a human-robot experiment that studied the well-known Joint Simon effect [10]. Using the end-to-end experimental workflow described in this chapter it was possible for a psychologist from Indiana University who was not an expert in robotics to reproduce an experiment originally carried out at Bielefeld University. We regard this as a clear success story of experimental reproducibility and see this as a best practice of reproducibility.

1.4 Analysis

1.4.1 Levels of Reproducibility

As a result of the project, Conquaire has developed a taxonomy of categories of analytical reproducibility:

1. **Publication Only:** No data or analysis code is available, neither publicly nor elsewhere beyond the written publication.

2. **Non-reproducible:** Data and analysis code is available, but results can not be reproduced because reproduction requires manual intervention, e.g. changing the order of script execution, unknown script parameters, imported libraries have changed and code can not be run, etc.

3. **Limited analytical reproducibility:** Results are in principle reproducible, but reproduction is hindered by: 1) the need to use commercial or proprietary software instead of open and free software (software lock-in) or 2) data is in non-standard formats and has to be transformed into common formats, 3) reproducing the analytical workflow requires interaction with or guidance by the original authors due to insufficient documentation.

4. **Full analytical reproducibility:** Data is ready to use and in standard formats, analysis code is available and documented and can be run without any modifications; software used is free and open. Results are thus fully reproducible without any need of modification, data conversion, etc.

5. **Sustainable analytical reproducibility:** Stability in environment (hardware and software) ensures full analytical reproducibility over a long term period.

1.4.2 Data formats used by case study partners

Our observation of the data formats and data sizes shows that three groups used proprietary software formats and the rest used open data formats. One research group required high-performance computing (HPC) facilities, unavailable at Bielefeld University, to generate large amounts of simulated data with a visualization pipeline to process their data, while the rest of the groups had a long tail of research data with a multitude of formats used for images, video and audio data. However, research groups that had raw data in the form of large videos, images and audio could not use Gitlab to store the raw data due to its file size limit of 2GB per file with the GitLFS extension. The most common data formats were CSV (and .tsv), Matlab (.m, .mat), Python (.py), XML, YAML (.yml), and Jupyter (.ipynb) notebooks.

1.4.3 Tools used by case study partners

Most research groups had a combination of technical tools used for processing and analyzing data. The most common programming language was Python (Theano, Pandas, Numpy, Scipy, scikit/sklearn, jupyter, etc.), closely followed by Matlab, for analysis and visualization. The technical stack also included other tools like MySQL, C, Java and their libraries for processing tasks. The proprietary toolkits included Origin and SPSS that handled the data processing spectrum, from analysis to visualization.

1.4.4 Reproducibility Analysis

From our eight case studies, we regard one as clearly fulfilling the criteria of *full analytical reproducibility*, i.e. in the case of the human-robot-interaction experiment attempting to reproduce the Joint Simon effect in a study involving multiple sites. All other case studies can be categorized as examples of at least *limited analytical reproducibility* with some of them reaching the status of full analytical reproducibility by the end of the project's lifetime.

The reproduction of the work from the Applied Computational Linguistics group on disfluency detection featured a high level of reproducibility as code and data were available in a public Git repository. Nevertheless, we could not reconstruct the exact version of the software used for the experiments, so that the results are only an approximation of the results of the original paper. If the version used in the experiments described in the paper was directly referenceable, we would definitely have a case of *full analytical reproducibility* as we could reproduce the results after minimal interaction with the authors of the original paper.

In some of these cases, the analytical pipeline relied on proprietary and commercial software such as Matlab, SPSS or OriginPro (this was the case for the Biological Cybernetics, Atmospheric Chemistry and Neuro-Cognitive Psychology group) as well as commercial operating systems (Microsoft) to run their analytical pipelines. In most of the cases the pipeline could be reproduced by open source tools such as R with some effort, but there is no guarantee that results are equivalent as implementations of statistical / analytical methods could potentially produce different results. In three cases (Neurobiology, Psychology, Atmospheric Chemistry), we were able to re-implement the analyses using open source tools (Python) and reproduce the results exactly. Yet, this was only possible due to intense guidance by the authors of the original article. By the end of the project, these three projects could be regarded as reaching the status of full analytical reproducibility.

The case study conducted with the Economic Theory And Computational Economics group deviates from the other case studies in that we did not reproduce a specific result, but jointly developed a library that allows to plot results from large-scale economic simulations, essentially allowing to project variables and visualize them. We still regard this as an example of limited analytical reproducibility as the data is in principle available, but too large to be stored in a Git repository. With the developed library, we enable users to analyse data from existing simulation data generated in the FLAME environment. This library thus fulfills a generic purpose and would allow for full reproducibility if the data would be available.

For the case study with the Biological Cybernetics group, we have a case of limited reproducibility due to the fact that not all the data has been made publicly available, but only sample data. Further, the analytical toolchain relies on a commercial tool, Matlab. The same holds for the use case with the Neuro-Cognitive Psychology group, whose analytical workflow also builds on a commercial system, the SPSS Software. Nevertheless, for both projects the data are openly available and recoding the scripts in free and open languages is only a matter of further effort.

In sum, we found no case of non-reproducibility or publication-only. In all of these use cases, the central results from the selected papers could be reproduced, albeit with some effort. In some cases, they could be reproduced under strong guidance of the authors and with commercial software only. In three cases, we could fully implement the analytical workflow using open-source tools, Python in particular, so that three out of eight cases can be counted as cases of full analytical reproducibility.

1.5 Summary

The Conquaire project has analyzed in detail eight case studies in computational reproducibility involving research groups from areas as varied as computer science / robotics, psychology, (computational) linguistics, biology, chemistry and economics. On the basis of accompanying the work of these groups over three years, it has developed a detailed understanding of the variety and heterogeneity of analytical research workflows.

In terms of infrastructure, Conquaire has developed infrastructure on top of a Git system that allows researchers to commit their data early in the research process into a distributed versioning system, with the benefit of providing a backup service but most importantly versioning the data and making different versions of the data referenceable. The project has also implemented continuos integration principles on top of the Git system, allowing research to define tests that their data have to pass as a basis to ensure data quality. It has implemented a badge system that publishes the results of the tests via the Bielefeld University PUB system to create incentives for researchers to make their data consistent and ready to be reused by others.

From a conceptual point of view, Conquaire has developed a taxonomy of reproducibility levels corresponding to different levels on the spectrum of reproducibility. We distinguish between 5 levels of reproduciblity: Publication Only, Non-reproducible research, Limited analytical reproducibility, Full analytical reproducibility and Sustainable analytical reproducibility. Our analysis of eight use cases allows us to conclude that only one of the projects considered came close to fullfilling the requirements of Sustainable Analytical reproducibility; three projects satisfy the criteria of *full analytical reproducibility.* In all case studies, we could successfully reproduce one central result from one of the published papers. The main obstacles for analytical reproducibility found were i) the lack of documentation and thus reliance on guidance by the original authors, ii) the reliance on some manual steps in the analytical workflow (e.g. clicking on a GUI) , iii) the reliance on non-open and commercial software, and iv) lack of information about which particular version of software and/or data was used to generate a specific result.

From our point of view, the project has been a success. First, we were able to reproduce more results than originally expected. Second, the detailed analysis of existing workflows has lead to a thorough understanding of the complexity and heterogeneity of involved analytical workflows. The fact that the data and scripts were available in a university-wide Git system is already a big success, as it makes research artifacts directly accessible. The use of social rewards was an interesting idea to explore, yet it remains to be seen if this sort of incentive-creating mechanisms is accepted by the community of researchers.

Overall, the Conquaire project has come to the conclusion that there is hope for improving the state of affairs regarding the reproducibility of research results if we provide institutional support for scientists to provide their code and data into an institutional repository if not a public repository as a first step to making artifacts referenceable and accessible in line with the FAIR principles. There is clearly a difficult and challenging agenda in front of us to make this happen at broad scale, but the Conquaire project has provided proof-of-concept that reproduction is feasible.

References

[1] Booch, G. (1991). *Object Oriented Design: With Applications.* Benjamin/Cummings.

[2] Buckheit, Jonathan B.; Donoho, D. L. (2005). On the computations analyzing natural optic flow: quantitative model analysis of the blowfly motion vision pathway. *WaveLab and Reproducible Research*, 25(27):6435–6448.

[3] Budke, C. and Koop, T. (2015). BINARY: an optical freezing array for assessing temperature and time dependence of heterogeneous ice nucleation. *Atmospheric Measurement Techniques*, 8(2):689–703.

[4] Foerster, R. M. and Schneider, W. X. (2015). Expectation violations in sensorimotor sequences: shifting from ltm-based attentional selection to visual search. *Annals of the New York Academy of Sciences*, 1339:45–59.

[5] Hough J, S. D. (2017). Joint, incremental disfluency detection and utterance segmentation from speech. In *Proceedings of European Chapter of the Association for Computational Linguistics (EACL).*

[6] LM, T., HH, B., and V, D. (2015). Comparative whole-body kinematics of closely related insect species with different body morphology. *J Exp Biol*, 218:340-352.

[7] Lobecke, A., Kern, R., and Egelhaaf, M. (2018). Taking a goal-centred dynamic snapshot as a possibility for local homing in initially naïve bumblebees. *The Journal of experimental biology*, 221(Pt 2):jeb.168674.

[8] Nomikou, I., Rohlfing, K., Cimiano, P., and Mandler, J. (2018). Evidence for early comprehension of action verbs. *Language Learning and Development.*

[9] Reinhart, C. and Rogoff, K. (2005). Growth in a time of debt. *American Economic Review Papers and Proceedings*, 100(2):573–578.

[10] Stenzel, A., Chinellato, E., Bou, M. A. T., del Pobil, Á. P., Lappe, M., and Liepelt, R. (2012). When humanoid robots become human-like interaction partners: corepresentation of robotic actions. *Journal of Experimental Psychology: Human Perception and Performance*, 38(5):1073.

2 | Conquaire Infrastructure for Continuous Quality Control

Fabian Herrmann[1], Christian Pietsch[2], Philipp Cimiano[1]

1 – Semantic Computing Group, Faculty of Technology & Cognitive Interaction Technology Excellence Center, Bielefeld University
2 – Bielefeld University Library, Bielefeld University

Abstract

In this chapter, we briefly describe the Git-based infrastructure that has been implemented as a result of the Conquaire project to support analytical reproducibility. The infrastructure implemented relies on principles of continuous integration as used in software engineering projects. Importantly, we rely on a distributed version control system (DVCS) to store computational artifacts that are key to reproducing the analytical results of an experiment. Using a DVCS has the benefit that artifacts can be versioned and each version can be uniquely addressed via a revision number. The DVCS implementation we rely on is Git, extended by GitLab as a web interface and collaboration platform. The heart of the infrastructure implemented in the Conquaire project is the so called *Conquaire server*. We assume that each research project deposits relevant data and code in a Git repository. In the background, a GitLab CI Runner on the Conquaire server is triggered by Git push events on the local GitLab server and executes a number of tests on the data and runs the code or script to reproduce a particular result. By this, we ensure that results can be reproduced independently of the original researchers on a separate machine. In this chapter, we briefly describe the infrastructure implemented and how tests are automatically executed when updates are committed to the Git repository.

2.1 Introduction

Principles of continuous integration have long been applied in software engineering to increase the quality of software artifacts and to prevent issues and failures due do integration of code developed in a distributed fashion by multiple developers in large software engineering projects. The heart of any continuous integration setup is typically a so-called *integration server* that runs a number of tests once updates of the software are committed and pushed, and possibly rejects the committed changes if they do not pass a number of tests. In continuous integration, software developers are encouraged to submit smaller changes in regular intervals to prevent errors and the well known *'integration hell'*.

Inspired by continuous integration principles, in Conquaire we have attempted to transfer these principles from the domain of software engineering into the domain of research data management. The starting point for any continuous integration is the availability of a repository into which data and code can be committed. Thus, a central part of the Conquaire architecture is a Distributed Version Control System (DVCS) that allows researchers to deposit their artifacts into a central repository. An important advantage of such a repository is that data and code can be versioned and each version can be uniquely referenced by a specific revision number. This allows to pinpoint and reference the exact version of code and data that was used to obtain a certain result, a central element of reproducibility.

Within Conquaire, we selected Git as a DVCS and GitLab as a web interface and collaboration platform to implement a university-wide repository allowing researchers to store their digital and computational research artifacts, code and data in particular. A key component of the Conquaire architecture is the so called *Conquaire server*, which in Conquaire acts as a continuous integration server. Upon a new commit, the GitLab CI Runner on the Conquaire server executes a number of predefined tests on code and data and runs code or scripts on data with the goal of reproducing a specific result. A central goal of Conquaire is to support the reproduction of a certain result independently on a separate machine that is out of the direct control of the original researchers.

The Conquaire server applies a number of quality checks on the data and publishes the results of these tests on a web server, sending an email to the person that committed the data to inform about the result of the test. Thus, researchers can get informed on the fact whether there are any problems with their data so that they can react early. We distinguish in Conquaire between generic tests that are specific for a certain file format (e.g. CSV or XML) and tests that are specific to the particular research projects. After tests are run, corresponding badges are generated indicating whether the tests were passed or not and rendered within a report that summarizes the results of the test. Conquaire is thus using principles from gamification to score the quality of data and thus create incentives for researchers to strive for high quality data that passes all tests.

In this chapter, we briefly describe the Conquaire approach to continuous integration as well as the core pieces and modules of the infrastructure implemented as part of the Conquaire project to verify quality of the data. In Section 2.2, we motivate our choice for Git and GitLab. In Section 2.3, we describe how we have implemented the Conquaire continuous integration infrastructure.

2.2 Why we use Git and GitLab

2.2.1 Git

One of the inspirations for this project came from the observation that GitHub had become popular not just among software developers, but also among other knowledge workers such as scientists. GitHub, as the name suggests, is built around Git (although second-class support for SVN was added later). So, of course, we looked at Git first, but we avoided committing ourselves to Git in the project proposal because a fair evaluation of all options was to be part of the project. We have to admit that we did not conduct a deep and thorough research to find alternatives. Git is the dominant versioning software today, and there is no foreseeable competitor. According to a survey by the popular question and answer website StackOverflow in 2015[1], out of 16,694 participants who answered this question, 69.3% used Git, 36.9% used SVN, 12.2% used TFS, 7.9% used Mercurial, 4.2% used CVS, 3.3% used Perforce, 5.8% used some other versioning software, and 9.3% used no versioning software at all. Other studies[2] come to similar conclusions.

Teaching researchers how to use a versioning software that is not widely used (such as Mercurial or Perforce) or is limited to one operating system (such as TFS) or is obsolete (such as CVS) was out of the question as we will not always be there to support them. Eventually, when they require help from other colleagues or their system administrator, Git will most likely be one of the versioning software they will know and provide support for. Of course, there are other criteria besides popularity that must be considered. A clear benefit of distributed versioning systems is that they can be used offline as they maintain the full history, including branches, locally. This is crucial to ensure long-term availability of data as lots of copies keep stuff safe, as the saying goes. As SVN is not a distributed versioning system, this alternative is ruled out. It goes without saying that any software used for archiving should be open source and freely licensed (FOSS). At the very least, its storage format must be documented openly. Freely available source code is a very precise way of documenting a storage format. Table 2.2 summarizes the main features that lead us to the decision to use Git to implement a university-wide distributed version control system.

We see two main disadvantages of using Git: (1) problems related learnability/usability and (2) lack of support of large files. Regarding learnability, finding out how hard it is for non-technical users to learn to use Git will be one of the outcomes of this project. Our working hypothesis is that for versioning research data, it is sufficient to learn a small subset of Git, which should not be too challenging. With respect to large files, the problem is that Git was originally not intended to be used with large files. The same is true for most versioning

[1] `https://insights.stackoverflow.com/survey/2015`
[2] `https://rhodecode.com/insights/version-control-systems-2016`

software name	popularity	actively maintained	distributed	cross-platform	FOSS
CVS	low	no	no	yes	yes
Git	high	yes	yes	yes	yes
Mercurial	low	yes	yes	yes	yes
Perforce	low	yes	no	yes	no
SVN	medium	yes	no	yes	yes
TFS	low	yes	no	no	no

Table 2.2: Features of different versioning systems

systems. They are intended for tracking changes that are caused by intellectual efforts: these rarely result in large files directly. Still, we want to include large files such as video recordings when documenting research projects. By large in this context, we mean a file larger than 50 MB. GitHub for instance warns users when pushing a file larger than 50 MB and does not accept files larger than 100 MB. Video files will often be larger than 100 MB. Fortunately, a free (MIT-licensed) and open-source extension to Git called Git Large File Storage (or Git LFS) can be used to alleviate this problem. It works around Git's size limitations by uploading large files to a separate storage area while tracking only metadata about these large files inside Git.

Using Git on the command line can be demanding. In our experience, graphical user interfaces (GUIs) that promise a more intuitive interaction style with Git often do not live up to expectations. Instead, we recommend a web interface.

2.2.2 GitLab

The web interface we use for Git is GitLab. GitLab started out as a GitHub clone, and became popular very quickly because it is available as a freely licensed community edition that includes source code needed for running a GitLab instance on premises.

Other web interfaces for Git such as Gogs and its derivatives were ruled out early on because they do not offer crucial enterprise features such as single sign-on (SSO) via LDAP or SAML2. Rolling out our source code hosting facility university-wide is part of the Conquaire project goals, so we needed to integrate with the Shibboleth-based identity management system of Bielefeld University. GitLab's SAML2 authentication method does just that.

GitLab proved to be an excellent choice because the makers of GitLab added the right features as our project progressed. For example, GitLab CI evolved from a simple continuous integration tool to a very powerful one, culminating in Auto DevOps, a feature set that provides a range of quality checks for software source code – not unlike what Conquaire provides for research data. However, Auto DevOps arrived only towards the end of the Conquaire project, so it did not influence our design decisions.

2.3 Conquaire Continuous Quality Control Infrastructure

2.3.1 Overview

A part of the Conquaire project was the development of automated data quality tests. The quality checks are integrated into the *GitLab* platform from the University of Bielefeld. The checks are written in *Python 3.6* and use the *lxml* package[3] for parsing XML files as the only external requirement. The pipeline of the quality check is shown in Figure 2.1 below. All steps are described in sections below.

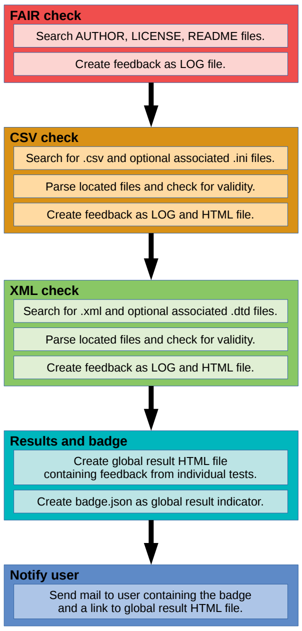

FAIR check
Search AUTHOR, LICENSE, README files.
Create feedback as LOG file.
CSV check
Search for .csv and optional associated .ini files.
Parse located files and check for validity.
Create feedback as LOG and HTML file.
XML check
Search for .xml and optional associated .dtd files.
Parse located files and check for validity.
Create feedback as LOG and HTML file.
Results and badge
Create global result HTML file containing feedback from individual tests.
Create badge.json as global result indicator.
Notify user
Send mail to user containing the badge and a link to global result HTML file.

Figure 2.1: Workflow of Conquaire Quality Check.

By adding a preconfigured YAML file (in this case: *.gitlab-ci.yml*) to a repository on the GitLab instance, the checks are automatically executed via a continuous integration runner on the GitLab server.
The runner creates a docker container. As the docker image we use the python:3.6-alpine image because it is lightweight and only contains an installed version of Python 3.6. In addition to that, we install the *lxml* package and a *SMTP*[4]

[3]`https://lxml.de/`
[4]`https://wiki.debian.org/sSMTP`

instance to notify the user about the results from a check. The user is informed via email about the result of applying the test. The mail contains information about the repository and a URL to a HTML site containing the detailed feedback which can be rendered by any browser. The mail also shows the user the overall test result which is displayed as a badge icon. The same icon is displayed in PUB if the user decides to create a data publication.

2.3.2 Example of pre-configured YAML file

The pre-configured file has to be stored in the root folder of the repository. For each commit to the repository, it is automatically executed by the CI runner and performs the Conquaire quality checks for the given repository. The user only has to change the value of the `-d` parameter as it represents the local path to the data inside the repository. In the given example, a folder named *data* inside the repository contains the files which should be tested.

```
quality-check:
  # Use smallest docker python image.
  image: python:3.6-alpine
  before_script:
    # Create temporary mail configuration files.
    - mkdir /etc/ssmtp
    - echo "root=${GITLAB_USER_EMAIL}" > /etc/ssmtp/ssmtp.conf
    - echo "mailhub=conquaire.uni-bielefeld.de" >>
        /etc/ssmtp/ssmtp.conf
    - echo "hostname=gitlab-runner.conquaire.uni-bielefeld.de"
        >> /etc/ssmtp/ssmtp.conf
    # Install lxml and ssmtp package for sending feedback mail.
    - apk add py3-lxml ssmtp
  script:
    # Execute quality checking pipeline.
    - /usr/bin/python3 /opt/conquaire/quality_checks/src/main.py
        -f /var/www/html/feedback/
        -l "https://conquaire.uni-bielefeld.de/feedback/"
        -r "$(pwd)"
        -d "data"
        -gn "${GITLAB_USER_NAME}"
        -ge "${GITLAB_USER_EMAIL}"
        -gu "${CI_PROJECT_URL}"
        -gp "${CI_PROJECT_PATH}"
        -gs "${CI_COMMIT_SHA}"
  # Choose docker CI runner on gitlab server.
  tags:
    - dockerexec
```

The whole pipeline is executed in a docker container and makes use of continuous integration variables provided by GitLab. They are automatically filled with the information from the users GitLab profile.

2.3.3 Quality checks

The Conquaire Quality Check pipeline involves a variety of tests that are automatically performed on the Git repository. Each time a commit occurs, the GitLab CI runner calls our pipeline, and several scripts are executed to guarantee that the provided data is in the best possible state. The three main checks that are implemented are the FAIR check, the CSV check, and the XML check. The pipeline is designed to be very modular and flexible to make it as easy as possible to extend it with further checks, i.e., for additional file types.

Every check begins with searching the repository and generating a list of every file with the specific type using the bash `find` command. For each file that was found, the corresponding test script is called to perform the actual checks and generate a log file with errors and warnings that were observed. The details of the three specific checks are described below. In the end, an overall feedback file is created, showing the results of the checks with links to the log files, making it possible to look into the data and correct it if necessary. The contributor is informed about the results of the pipeline via email.

FAIR check

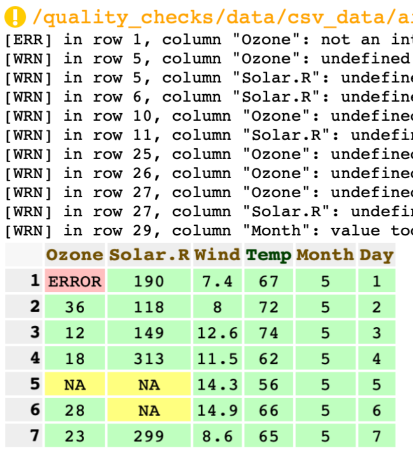
In our adaptive implementation of the FAIR metrics[5], we check if the three necessary files exist in the repository: the AUTHORS, LICENSE, and README files.
The files have to be placed in the root directory of the repository to fulfill the test condition. The files have to have either no extension, plain text (`.txt`) or markdown (`.md`).
We suggest to save the files as markdown files. The markdown file type is used as a standard in GitLab and many other websites because it has an easy to learn syntax and can be displayed in a web browser.

The AUTHORS file should contain a list of all the contributors and their emails for the possibility to contact them. The LICENSE file should describe how the data can be further used and distributed by other researchers, either by declaring one of the common licenses or providing their own. The README file should contain every other information that is related to the data and necessary or helpful to understand the research that was done, e.g., a description of the data or the experiment to obtain it.

[5] `http://fairmetrics.org/`

CSV check

In the CSV file format (`.csv`), data is organized as a table with comma separated values. The first step in the CSV check is to test whether the file can be opened and the table is well-formed, i.e., it has a header and a consistent number of rows and columns.
The researcher can provide an additional format declaration file (`.ini`) with his own specifications of the data, e.g., the type of the column and the expected range of the values. The quality check reports a warning if a required entry is missing or a value is out of range or has a wrong type, e.g., a non-numeric value in a numeric column.



```
/quality_checks/data/csv_data/airquality2.csv
[ERR] in row 1, column "Ozone": not an integer
[WRN] in row 5, column "Ozone": undefined entry
[WRN] in row 5, column "Solar.R": undefined entry
[WRN] in row 6, column "Solar.R": undefined entry
[WRN] in row 10, column "Ozone": undefined entry
[WRN] in row 11, column "Solar.R": undefined entry
[WRN] in row 25, column "Ozone": undefined entry
[WRN] in row 26, column "Ozone": undefined entry
[WRN] in row 27, column "Ozone": undefined entry
[WRN] in row 27, column "Solar.R": undefined entry
[WRN] in row 29, column "Month": value too large
```

	Ozone	Solar.R	Wind	Temp	Month	Day
1	ERROR	190	7.4	67	5	1
2	36	118	8	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
5	NA	NA	14.3	56	5	5
6	28	NA	14.9	66	5	6
7	23	299	8.6	65	5	7

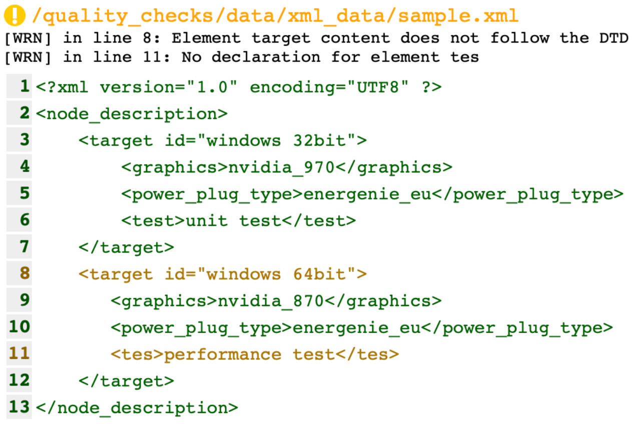
Figure 2.2: Example result of the CSV check.

The log file lists all the errors and warnings that were found, and the row and column in which they occurred. In addition to that, the corresponding cell is marked in the table, allowing to conveniently find problematic entries, as seen in the example in Figure 2.2.

XML check

In the XML file format (`.xml`), data is organized as a tree structure using tag-based markup language. The first step in the XML check is to test whether the file can be opened and the document is well-formed, i.e., the syntax of the markup language tags is correct.
The researcher can provide an additional doctype definition file (`.dtd`) with his own specifications of the data, e.g., the required attributes and some restrictions to the values. The quality check reports an error if there is a mismatch of opening and closing tags, and a warning if the specifications are not fulfilled,

e.g., a value is missing.



```
/quality_checks/data/xml_data/sample.xml
[WRN] in line 8: Element target content does not follow the DTD
[WRN] in line 11: No declaration for element tes
 1 <?xml version="1.0" encoding="UTF8" ?>
 2 <node_description>
 3     <target id="windows 32bit">
 4         <graphics>nvidia_970</graphics>
 5         <power_plug_type>energenie_eu</power_plug_type>
 6         <test>unit test</test>
 7     </target>
 8     <target id="windows 64bit">
 9        <graphics>nvidia_870</graphics>
10        <power_plug_type>energenie_eu</power_plug_type>
11        <tes>performance test</tes>
12     </target>
13 </node_description>
```

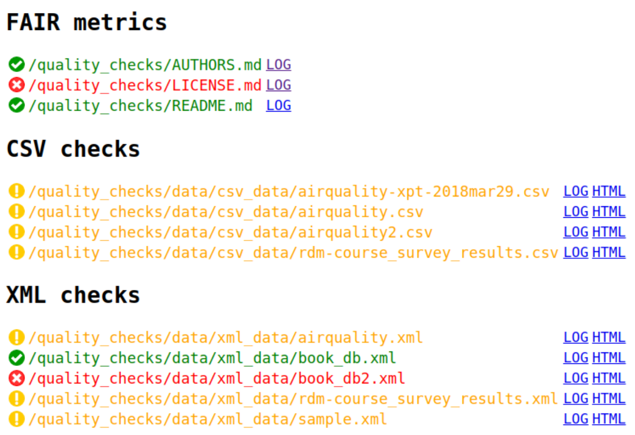
Figure 2.3: Example result of the XML check.

The log file lists all the errors and warnings that were found, as well as the line in which they occurred. In addition to that, the corresponding line is marked in the document, allowing to conveniently find problematic entries, as seen in the example in Figure 2.3.

After successful execution, the Conquaire quality check pipeline produces an overall result HTML file which contains visual feedback of all individual tests and links to the resulting log and optional HTML files. An example is provided in Figure 2.4. The feedback shows one of three different colors and badges. A green badge represents a successful test result, i.e., the data is valid. A yellow badge indicates well-formed data and the log files can contain some warnings. A red badge indicates not well-formed data or missing FAIR files. The user should check the log files and fix the errors before submitting a data publication. The URL of the overall result is provided to the user via email. This mail is sent automatically after every commit. In addition to that, an overall badge icon is created. This badge is equivalent to the badge of the worst individual check result. This badge is displayed in PUB[6] if the user decides to create a data publication from the repository. The badge is equal to one of the three different symbols shown in Figure 2.4.
Thus, the Conquaire quality checks are designed to help the researchers to clean up data, remove inconsistencies and make it fit for use by others.

Fulfilling the FAIR metrics is highly important for the reproducibility of the data as they are necessary to provide other researchers the information and legal

[6]https://pub.uni-bielefeld.de/



```
FAIR metrics

✓ /quality_checks/AUTHORS.md LOG
✗ /quality_checks/LICENSE.md LOG
✓ /quality_checks/README.md  LOG

CSV checks

! /quality_checks/data/csv_data/airquality-xpt-2018mar29.csv  LOG HTML
! /quality_checks/data/csv_data/airquality.csv                LOG HTML
! /quality_checks/data/csv_data/airquality2.csv               LOG HTML
! /quality_checks/data/csv_data/rdm-course_survey_results.csv LOG HTML

XML checks

! /quality_checks/data/xml_data/airquality.xml                LOG HTML
✓ /quality_checks/data/xml_data/book_db.xml                   LOG HTML
✗ /quality_checks/data/xml_data/book_db2.xml                  LOG HTML
! /quality_checks/data/xml_data/rdm-course_survey_results.xml LOG HTML
! /quality_checks/data/xml_data/sample.xml                    LOG HTML
```

Figure 2.4: Example result of the overall result.html.

basis to use the data for their consecutive works. The file type specific checks help finding and fixing errors before releasing the data to the public. This is fundamental for reproducibility as only valid data can be used to recreate the experiment results.

2.4 Summary

In this chapter, we have briefly described how the Conquaire infrastructure implemented at Bielefeld University applies continuous integration principles to support reproduction of analytical results but also ensure high quality and valid data. The basis of the infrastructure is a distributed version control system (DVCS) that stores different versions of computational artifacts. In this chapter, we have argued why we have selected Git as a basis to implement this DVCS at Bielefeld University and why we have selected GitLab as a graphical and web-based user interface to access Git and foster collaboration. We have further described how the Conquaire infrastructure automatically runs a number of quality checks on the data once a new commit has been performed. The user merely has to add a YAML file to the root directory of the repository. This YAML file will trigger the GitLab CI runner to execute a number of standard tests on CSV and XML files to check whether the data is consistent, syntactically well-formed and complies with schema declarations. The results of each test are written into a log file and used to generate a report that is published as a website on a web server. A link to this report is sent to the user committing the data for inspection of the results of the tests, giving access to the detailed logs. Building on principles of gamification and to create incentives for committing ready-to-use-data, the Conquaire systems assigns badges to the data corresponding to whether they passed the tests or not and visualizes these badges in the reports generated and optionally on a PUB page where the data has been published.

During the Conquaire project, we have run a number of Git workshops with all case study partners, confirming our hypothesis that the subset of Git commands that is needed to commit data into the repository can be easily learned by our target population. On the basis of our experience, we can definitely recommend Git, GitLab and our architecture for continuous integration to implement an institutional infrastructure for hosting data and checking their quality as a basis to ensure reproducibility of research results.

3 | Reproducibility of whole-body movement analyses of insects

Yannick Günzel[1], Fabian Hermann[2], Vidya Ayer[2], Philipp Cimiano[2], Volker Dürr[1]

1 – Biological Cybernetics, Faculty of Biology, Bielefeld University
2 – Semantic Computing Group, Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University

Abstract

In this chapter we describe the attempt to reproduce a selected figure of the paper *"Comparative whole-body kinematics of closely related insect species with different body morphology"* [1]. In this paper, the authors investigated the walking behaviour of three different species of stick insects. This was done by recording whole-body kinematics of the animals, using a commercial marker-based motion capture system and custom written MATLAB scripts. The main objective of the study was to relate inter-species differences in kinematics to differences in overall morphology, including features such as leg-to-body-length ratio that were not an obvious result of phylogenetic or ecological divergence. The present chapter describes an effort to reproduce one of the figures that was published in the original study that evaluates climbing behaviour by means of (i) snapshots of body posture, (ii) 3D trajectories of front legs and body, and (iii) the gait pattern of a representative trial. We show that the figure could be reproduced successfully, albeit requiring detailed interaction with the authors as well as use of commercial software. Accordingly, we classify this use case as corresponding to our category limited analytical reproducibility. The data and scripts are available in the following Git repository: `https://gitlab.ub.uni-bielefeld.de/conquaire/biological-cybernetics`.

Keywords

Insect Locomotion, Whole-body kinematics, MATLAB

3.1 Introduction

The overall goal of the Biological Cybernetics lab at Bielefeld University is to understand the mechanisms underlying the control of natural movement and action sequences. To this end, the lab studies the adaptive locomotion abilities of insects with a research focus on the function of active tactile sensing (touch) and distributed proprioception (the sense of posture). A key methodology of the lab is whole-body motion capture of unrestrained walking and climbing insects (e.g., [2] [1]), which was also in the focus of the present data management study. More recently, whole-body motion capture has been combined with ground-reaction force measurements and the corresponding calculation of single-joint torques [3], as well as coincident muscle activity recordings during unrestrained walking [4]. Insects have become important model animals for the study of flexible and adaptive locomotion (e.g., [5] [6]). Although a wide range of behavioural (e.g., [7]), biomechanical (e.g., [8]) and neurophysiological ([9], [10]) studies on insect locomotion have contributed to a detailed understanding of multi-legged locomotion in general, there are very few studies on comparative kinematics of insect walking or climbing. Legged locomotion through natural or naturalistic environments is very complex and variable. Leg kinematics may not only differ strongly among species, but also within the same species it is adaptive and context-dependent. Inter-species differences in locomotion are often difficult to interpret, because both morphological and ecological differences among species may be strong and, as a consequence, confound each other's effects. Moreover, in species from phylogenetically distant taxa, i.e., that diverged a long time ago in evolution, differences in motor behaviour may simply be a result of evolutionarily divergent morphological or physiological constraints. The experimental data of the present case study was taken from a study that is to date the only example of a whole-body kinematics comparison of different insect species [1]. The species compared differed in body morphology, despite close phylogenetic relationship and similar ecology. *Carausius morosus*, *Aretaon asperrimus* and *Medauroidea extradentata* (= *Cuniculina impigra*) belong to the same order of insects (*Phasmatodea*: stick and leaf insects). All three species are flightless and live a herbivorous and nocturnal life style. Accordingly, the main objective of that study was to relate inter-species differences in kinematics to differences in overall morphology, including features such as leg-to-body-length ratio, that were not an obvious result of phylogenetic or ecological divergence. The original study suggests that major differences among species were related to antenna length, segment lengths of thorax and head, and the ratio of leg length over body length.

3.2 Methods

This section describes the material and methods used in the research project published in [1]. After illustrating the overall workflow (subsection 3.2.1), we describe the acquisition of the original experimental data (subsection 3.2.2), the manual editing and annotation procedures (subsection 3.2.3), as well as the secondary data processing (subsection 3.2.4). Note that subsections 3.2.2 to 3.2.4 repeat previously published method section parts of [2] and [1].

3.2.1 Data workflow: acquisition and processing pipeline

The overall data workflow used in this project is summarized in the chart shown in Fig. 3.1 (left column). There were three processing episodes: (i) data acquisition, (ii) manual editing and annotation, and (iii) secondary processing. The coloured boxes illustrate the procedure for recording the different types of data and how it was ultimately processed to reconstruct body and leg kinematics as displayed in Fig. 3 in the paper of Theunissen et al. [1]. The colours of the boxes indicate the software used for a given step in the data processing pipeline (yellow: *Vicon Nexus*; green: *PixeLINK Capture*; blue: *MATLAB*). The boxes and connecting arrows are labelled with the data file types produced, the relative file paths to the corresponding subdirectories, and the names of custom-written MATLAB (MathWorks, Natick, MA, USA) scripts.

3.2.2 Data acquisition: Experimental procedure

For the experiments described in [1], adult stick insects of the species *Carausius morosus* (de Sinéty 1901), *Aretaon asperrimus* (Brunner von Wattenwyl 1907) and *Medauroidea extradentata* (Redtenbacher 1906) were used. Animals were bred in a laboratory culture at Bielefeld University.

In each experimental trial, an animal was placed on a horizontal walkway (40 x 490 mm), along which it walked freely. There were four walking/climbing conditions as characterised by the height of two stairs placed on the walkway: in the flat (walking) condition, the walkway was used without stairs; in the climbing conditions low, middle and high, a staircase with two stairs of step height, h, was placed at the end of the walkway (40 x 200 mm; low: h = 8 mm, middle: h = 24 mm, high: h = 48 mm). The flat walking condition served as the reference condition. The four conditions were presented in a randomised sequence of at least 40 trials, resulting in approximately ten trials per condition per animal. The whole setup was painted in opaque black and was surrounded by black drapery in order to minimise visual contrast. The room was darkened and illuminated only by red light LEDs of the Vicon cameras (see below) and indirect light emanating from a TFT computer monitor.

A marker-based motion capture technique was applied, for which each animal was labelled by 17 or 18 retro-reflective markers (Fig. 3.2). Marker diameter

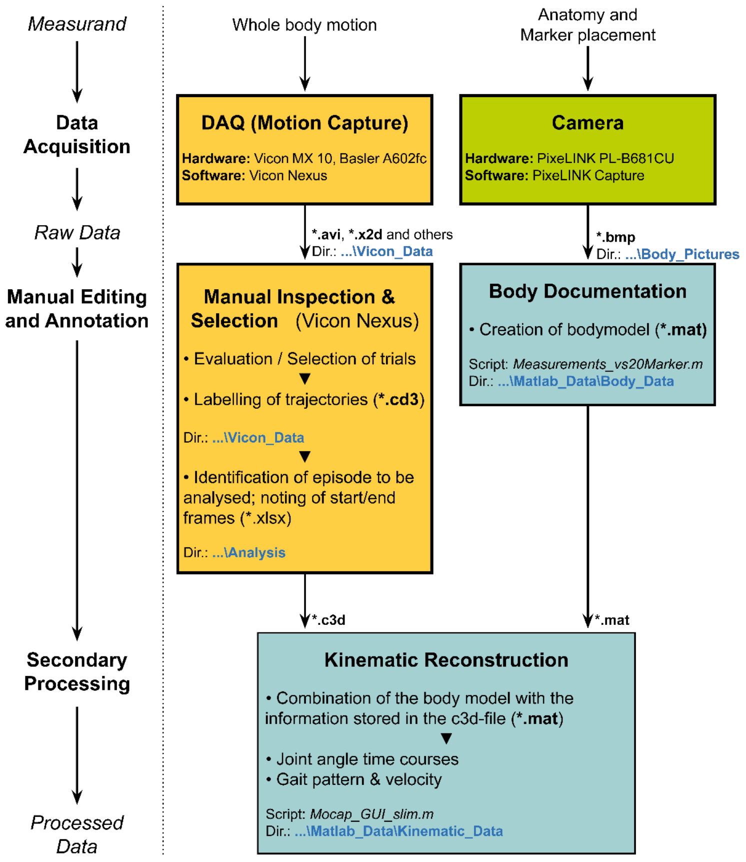

Measurand
Data Acquisition
Raw Data
Manual Editing and Annotation
Secondary Processing
Processed Data
Whole body motion
Anatomy and Marker placement
DAQ (Motion Capture)
Hardware: Vicon MX 10, Basler A602fc
Software: Vicon Nexus
Camera
Hardware: PixeLINK PL-B681CU
Software: PixeLINK Capture
*.avi, *.x2d and others
Dir.: ...\Vicon_Data
*.bmp
Dir.: ...\Body_Pictures
Manual Inspection & Selection (Vicon Nexus)
• Evaluation / Selection of trials
• Labelling of trajectories (*.cd3)
Dir.: ...\Vicon_Data
• Identification of episode to be analysed; noting of start/end frames (*.xlsx)
Dir.: ...\Analysis
Body Documentation
• Creation of bodymodel (*.mat)
Script: Measurements_vs20Marker.m
Dir.: ...\Matlab_Data\Body_Data
*.c3d
*.mat
Kinematic Reconstruction
• Combination of the body model with the information stored in the c3d-file (*.mat)
• Joint angle time courses
• Gait pattern & velocity
Script: Mocap_GUI_slim.m
Dir.: ...\Matlab_Data\Kinematic_Data

Figure 3.1: **Research data acquisition and processing pipeline.** For raw data acquisition, whole body motions were recorded with a marker-based motion capture system (Vicon) and an additional digital video camera. Furthermore, the anatomy of the animal, along with the marker positions on different body segments were recorded with a microscope camera. In a first step of manual editing and annotation, marker trajectories of selected episodes were labelled and, potentially, connected in case of recording gaps. This step resulted in a *.c3d*-file, a file format described in section 3.3.1. The body pictures were used to generate a body model containing, for example, segment lengths and information about marker position in a body-centred coordinate system. The model is stored in a MATLAB *.mat*-file. Finally, the kinematic reconstruction was achieved in MATLAB by combining marker trajectories with the body documentation. The resulting processed data, i.e., joint angle time courses, gait pattern, and velocity were saved as another MATLAB file.

was 1.5 mm. Markers were glued to the cuticle by use of transparent nail polish. Two markers were attached to each leg, one to the distal femur and one to the distal tibia (Fig. 3.2, right panel). Additionally, five markers were attached to thorax and head, with three markers defining the body-fixed coordinate system of the metathorax and one additional marker on the prothorax and head (Fig. 3.2 B, left panel). In most animals, a further marker was placed on the rostral mesothorax. Care was taken that neither the nail polish nor the markers constrained the movement of any joint. Segment dimensions and the positions of all markers on their respective body segment were measured from high-resolution photographs (0.02 mm per pixel) taken under a stereo lens (Olympus SZ61T, equipped with a digital camera (Pixelink PL-B681CU), controlled by *μScope* software (top right green box in Fig. 3.1).

A Vicon MX10 motion capture system with eight *T10 cameras* (Vicon, Oxford, UK) was used for data acquisition of marker positions (top yellow box in Fig. 2). Temporal resolution of the motion capture system was 200 Hz; spatial resolution was approximately 0.1 mm. The time of entry of the animal into the capture volume was used as starting frame of the recording. The recording was stopped when the animal reached the far end of the setup. Trials were discarded if the animal climbed the side walls of the setup instead of the stairs, or stopped walking before the first stair. In this case, the same trial condition was repeated.

An additional digital video camera (Basler A602fc, Ahrensburg, Germany) equipped with a near range zoom lens (Edmund Optics, Barrington, NJ, USA) was used to record a complementary image sequence for visual inspection, e.g., for validation of the kinematic analysis. The video showed a side view of the climbing sequence of the first stair, with a temporal resolution of 50 Hz (synchronized with the Vicon system) and a spatial resolution of approximately 0.14 mm per pixel. The software Nexus 1.4.1 (Vicon, Oxford, UK) was used for controlling the motion capture process and for subsequent offline analysis.

3.2.3 Manual editing and annotation: trajectory labelling and body model

Within Nexus, each of the markers was identified and labelled at least once per trial by hand (second yellow box in Fig. 3.1). Markers were then tracked automatically, provided that each marker was recorded by at least two cameras. The resulting trajectories of spatial coordinates of all markers were inspected for filling of small trajectory gaps. Generally, marker detection was very robust. On average, less than 5 gaps per 60 s occurred in single marker trajectories in case of *C. morosus* trials, with mean trial durations of 11.29 ± 4.8 s, equivalent to 2258 ± 964 frames (mean ± standard deviation). Gaps shorter than 200 ms (40 frames) were filled by use of an interpolation algorithm of the software Nexus.

A body model was established for each animal, using a custom-written Graphical User Interface in MATLAB (top right blue box in 3.1) that loaded all available photos and prompted the user to click on segment limits and marker locations (middle blue box in 3.1). The body model consisted of a branched kinematic chain (Fig. 3.2B) with a four-segmented body axis and six three-segmented limbs. The corresponding body model file contains information about body and leg segment dimensions, attachment locations of side chains on the main chain (i.e., the locations of the thorax-coxa joints), the marker coordinates relative to the base of their carrying segment, and bias rotations of the marker-fixed coordinate system defined by the three makers on the root segment (Fig. 3.2B, left panel) relative to the body-centred coordinate system defining the sagittal, horizontal and transverse planes of the body.

3.2.4 Secondary processing: Whole-body kinematics

Whole-body kinematics yielded the joint angle time courses associated with 42 degrees of freedom (DoF) of the body model. All calculations were done in MATLAB (lower blue box in 3.1), using the toolbox C3Dserver (Motion Analysis Laboratory, Erie, PA, USA), for importing C3D data from Vicon Nexus (subsection 3.3.1).

Scaling and filtering: Joint angles were calculated by use of two data sets coming from (i) the segment lengths and marker positions on the animal, as calibrated under the stereo lens, and (ii) the marker trajectories, as obtained from motion-capturing.

Since the body model measurements were more precise than the Vicon calibration, the marker trajectories were scaled by the factor lBM/lMC, where lBM is the distance of two markers in the body model with fixed distance (e.g., two markers on the metathorax), and lMC is the corresponding mean distance of the same markers in the motion capture data. lBM/lMC ranged from 0.94 to 1.00, mainly depending on the calibration quality of the Vicon system. After scaling of marker trajectories, the time courses of all marker coordinates were low-pass filtered in MATLAB, using a 4th-order Butterworth filter with a cut-off frequency of 20 Hz.

The motion capture data yielded information about the animal's position and posture in each frame in a right-handed, world-fixed coordinate system (CS) with the x- and y-axes aligned with the long and traverse axes of the setup walkway, respectively, and the z-axis pointing upwards. The centre of the segment border between the 1st and 2nd abdominal segment (note that, in stick insects, the 1st abdominal segment is fused to the metathorax) was taken as origin for a thorax-fixed root CS. With regard to this root CS, all positions of the other thorax segments and the coxae were expressed in right-handed Cartesian coordinates, with the x-axis pointing rostrad within the sagittal plane, i.e., from the origin towards the head, the horizontal y-axis pointing towards the left within the horizontal body plane, and the z-axis pointing dorsad within the sagittal plane.

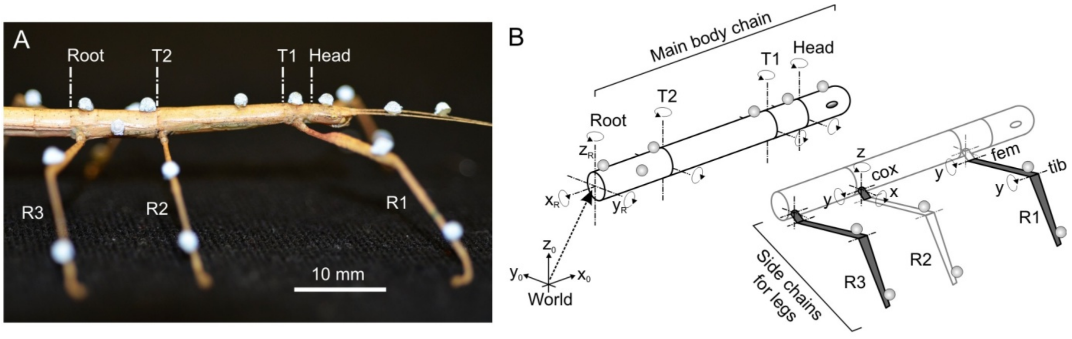


Figure 3.2: **A marker-based motion capture and whole-body kinematics calculations. A:** Insects were labelled with reflective markers. **B:** For kinematic analysis, the body was modelled by a branched kinematic chain. The main body chain (left) consists of the three thorax segments (Root, T2, T1) and the head. Six side chains (right) model the legs, with the segments coxa, femur and tibia (cox, fem, tib; only right legs are shown, labelled R1 to R3). All rotation axes (DoF) are indicated (3 for the root segment, 2 for thorax/head segments, and 5 per leg). DoF are denoted according to the subsequent segment and the axis of the local coordinate system around which the rotation is executed. Leg DoF are: cox.x, cox.y, cox.z (labelled for R2 in right panel), fem.y and tib.y (labeled for R1 in right panel). [Fig. 1 A, B of [2]]

Calculating the main body chain: The main kinematic chain included the three thorax segments and the head. The root segment (metathorax, including the fused 1st abdominal segment) had six DoF: three translational DoF indicating the position of the body in the external coordinate frame [x0, y0, z0] and three rotational DoF, indicating roll, pitch and yaw rotation around the x0-, y0-, and z0-axis, respectively. The other three segment joints of the main body chain had two rotational DoF each, indicating pitch and yaw rotation around the segments y- and z-axes, respectively. This resulted in twelve DoF for the main chain. In four animals with 17 markers (without second mesothorax marker), the metathorax-mesothorax joint was assumed to be immobile.

The rotation of the root segment with respect to the world coordinate system was determined from the axis orientations of a body-fixed root coordinate system ([xR, yR, zR] in Fig. 3.2B). The latter was defined by the three markers on the root segment, such that xR pointed in the direction of the main chain and zR was orthogonal to the plane defined by the three markers. The calibration images of the asymmetric side marker on the root segment yielded a bias rotation angle. Back-rotating the marker-fixed root coordinate system by this angle yielded alignment [xR, yR, zR] with the sagittal, horizontal and frontal body planes. Measures taken from calibration images were then used to calculate the origins of all connecting segments. In case of the root segment, these were the mesothorax (T2) and the hind leg coxae (R3.cox, L3.cox). Next, the vector connecting the root-T2 joint with the marker on T2 was calculated. After back-rotating this vector by its bias rotation with respect to [xR, yR, zR], as determined from calibration images, its polar coordinate angles yielded the joint rotation angles around the axes T2.z and T2.y. The resulting T2-fixed coordinate system was used to calculate the origins of the prothorax (T1) and of the middle leg coxae. The rotation angles of the T2-T1 joint and T1-head joint, along with the remaining segment origins of the main body chain were calculated in analogy to the calculation steps taken for T2.

Calculating the six side chains: Each thorax segment was connected to two kinematic side chains, modelling the left and right legs (see Fig. 3.2, right panel, where R1 to R3 label the right front to hind legs). The side chains consisted of a coxa with three rotational DoF in the thorax-coxa joint (ThC-joint [protraction/retraction, levation/depression, supination/pronation]), the trochantero-femur (subsequently called femur) with one DoF in the coxa-trochanter joint (CTr-joint, [levation/ depression]), and the tibia with one DoF in the femur-tibia joint (FTi-joint, [extension/flexion]). For calculation of the leg joint angles, the first step was to determine the „leg plane“ spanned by the two leg markers and the origin of the corresponding side chain. If the normal vector of this plane was expressed within the coordinate system of its connecting thorax segment, its polar coordinate angles gave the protraction/rectraction and supination/ pronation of the ThC-joint, along with the rotated z- and x-axes defining the leg plane. The sum of levation/depression in the ThC- and CTr-joints was then calculated by expressing the vector connecting the ThC-joint to the femur marker within the xz-coordinate system of the leg plane. From the known segment lengths of coxa and femur, along with the exact marker position on the femur, the relative contribution of the ThC- and CTr-joint to femoral levation could be determined by triangulation. Finally, the known femur length was used to determine the location of the FTi-joint, and the vector connecting the latter to the tibia marker was used to calculate the extension/flexion of the FTi-joint (with consideration of the bias rotation caused by the misalignment of the tibial marker and the tibial axis).

3.3 Analytical Reproducibility

All data files and MATLAB scripts for analysis as listed in Fig. 3.1 were made available by the Biological Cybernetics group. As a result, the data and scripts are available at `https://gitlab.ub.uni-bielefeld.de/conquaire/biological-cybernetics`. Data that were not part of the reproducibility check (e.g., raw video files, fotos and data files used by the proprietary software Nexus only) will not be discussed.

3.3.1 Analysis pipeline, data formats and software tools

As described in section 3.2 the research group used MATLAB for all computational data analysis and creation of plots. Accordingly, the original codebase is fully written in MATLAB. The motion data was recorded with a Vicon motion capture system, operated by the proprietary software Nexus. The reproducibility check thus started with the labelled marker trajectory data that was exported from Nexus in the C3D format. The *.c3d*-files were loaded into MATLAB with the help of C3Dserver. Specific versions of MATLAB need to be installed for processing the loaded data from the C3Dserver.

C3Dserver and file formats

The C3Dserver is a 32/64-bit C3D Software Development Kit (SDK) for Microsoft Windows® platforms only. It simplifies C3D file programming and data access by providing the users with high-level commands to create, modify and process data. The C3Dserver can be freely downloaded and installed on all 64-bit and 32-bit versions of Microsoft Windows from XP through Windows 10 using the standard Microsoft user environment. Data saved from the Vicon motion tracker has to be loaded into MATLAB with the help of the C3Dserver. While the server is available as 32-bit and 64-bit versions with identical C3D access functions, one can only run 32-bit applications on a 32-bit installation as the 64-bit C3Dserver DLL will not be installed on a 32-bit server. On the other hand, if the C3Dserver is installed on a computer with a 64-bit operating system, then we can install distinct 32-bit and 64-bit DLLs, making it easier to use the

C3Dserver with both 32-bit and 64-bit applications. The 64-bit DLL will be installed as `C:\ProgramFiles\CommonFiles\MotionLabSystems\C3Dserver\c3dserver64.dll`. The 32-bit DLL will be installed in `C:\ProgramFiles(x86)\CommonFiles\MotionLabSystems\C3Dserver\c3dserver.dll`.

3.3.2 Technical Challenges and Issues

Scientific research groups use a variety of file formats with various machines using standard formats to read in data and output it. Here, the captured data is stored in a *.c3d*-file that can be exchanged and accessed via the C3Dserver, but it is predominantly supported to run on the Windows platform only. The C3D file format is a public domain file format for storing motion and other 3D data recorded in various laboratory settings. The C3Dserver, whose server features include several MATLAB supporting functions that allow files to be analysed with additional MATLAB functions being written to perform operations on the data in *.c3d*-file.

The biggest challenge we thus faced was the requirement of the proprietary C3Dserver for data processing, analysis and visualisation that was only available for machines running the Windows operating systems. Since there was no software support for Linux to read in the motion tracking data to MATLAB, we could not recreate the full pipeline on a Linux machine. The Library is maintaining the infrastructure for research data management (RDM), hence, they would have the additional work of installing, both MATLAB and the Windows server, patching and updating them regularly, including maintaining licensed version upgrades which can get expensive over time. The kinematic reconstruction was achieved in MATLAB by combining marker trajectories with the body documentation. The resulting processed data, i.e., joint angle time courses, gait pattern, and velocity, were saved as another *.mat*-file.

Another problem was related to the backslash used in PATHS on the Windows machine. All relative paths in the code supported Windows, which uses a backslash instead of (forward)slash on *nix machines. While analysing the MATLAB data with C3Dserver and MATLAB on Windows, this is not an issue. However, a user trying to use the MATLAB code on a *nix machine would have to replace and correct all the paths before running the code to reproduce the figures from that point onwards. For example: For Figure 3.3B the *nix user can type these code commands from the terminal after they loaded the data beforehand:

```
figure; hold on
%    Trajectory of the tibia-tarsus joint of the left front leg
plot3(data.L1.tar.pos(:,1), data.L1.tar.pos(:,2), data.L1.tar.pos(:,3),'r', 'LineWidth', 2)
%    Trajectory of the tibia-tarsus joint of the right front leg
plot3(data.R1.tar.pos(:,1), data.R1.tar.pos(:,2), data.R1.tar.pos(:,3),'g', 'LineWidth', 2)
%    Trajectory of the head
plot3(data.Hd.pos(:,1), data.Hd.pos(:,2), data.Hd.pos(:,3),'k', 'LineWidth', 2)
%    Equal aspect ratio
axis equal
```

Furthermore, the most severe limitation was due to the use of proprietary software tools, like Windows-only SDK. As there was no free and open source software (FOSS) support for the SDK, it was impossible to recreate or plug into the analysis pipeline with a Linux machine. Since MATLAB uses Gnuplot as the plotting engine, we could pipe-in (read) the data with Octave2, an open source MATLAB clone, and plot the data. As the plotting engine (Gnuplot) is the same for Octave and MATLAB the figure rendering is similar to the published paper. Thus, the three figures in the paper can be reproduced using FOSS toolkits in a Linux environment if the data was created beforehand with the help of the C3Dserver and MATLAB on Windows.

As a result of our reproduction experiment, we could reproduce the walking and climbing behaviour for those experimental runs that were committed into the corresponding GIT repository. Figure 3.3 shows on the left the original panel from the paper published by Theunissen et al. [1] for *C. morosus*. On the right, our reproduction of the same trial is depicted. As the figure shows, asides from the rendering of the obstacle and the colouring, we could successfully reproduce the plots from the original paper.

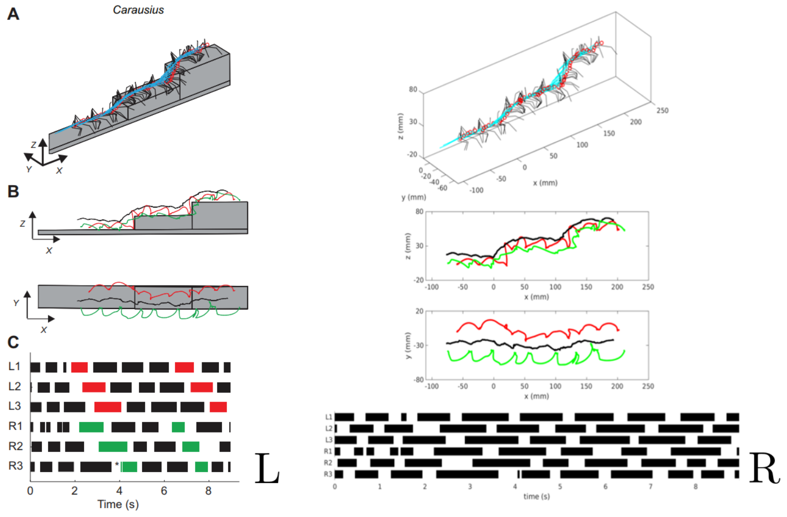


Figure 3.3: **Representative trial of unrestrained walking and climbing behaviour of *C. morosus* as one of the three species investigated in the original paper published by Theunissen et al. [1] (Figure 3).** The left panel *L* shows the original figure section. The right panel *R* shows the movement as reproduced in the reproduction study conducted in the context of this chapter. The A, B and C subcomponents of the diagram show the following: **A**: Movement of the body axis (cyan lines), head (red circles) and front legs (black lines), illustrated by superimposed stick figures every 100 ms. **B**: Trajectories of the tibia-tarsus joint of left (red) and right (green) front legs, and of the head (black line) super-imposed on the setup in side and top view. Note that the caption of the original publication says metathorax instead of head at this place. This mislabelling was discovered during the replicability study. The authors apologise for this error. The mislabelling has no effect on any claims made in the original paper. **C**: Podogram of the gait pattern, i.e., time sequences of the alternating swing-stance-phases of all six walking legs, where each black line depicts the duration of a stance phase of one of the legs. Red and green lines mark the first stance phases on the next stair in left and right legs, respectively. **L1 to L3**: left front, middle and hind leg; **R1 to R3**: corresponding right legs.

3.4 Conclusion

We have described a reproducibility case study in the field of biology. We have in particular attempted to represent the main results of a study in whole-body movement analysis of three species of stick insects. The main objective of the study was to relate inter-species differences in kinematics to differences in overall morphology, including features such as leg-to-body-length ratio, which were not an obvious result of phylogenetic or ecological divergence. We have shown that we could successfully reproduce a main figure of the paper *"Comparative whole-body kinematics of closely related insect species with different body morphology"* by Theunissen et al. [1]. We classify this case as one of *limited analytical reproducibility.* While we could reproduce the whole-body movements for a number of experimental runs that the authors provided in a GIT repository, this has only been possible by direct guidance of the authors. Further, the reproduction relies on use of commercial software, in particular MATLAB as well as the C3Dserver running on Windows only.

Acknowledgements

We would like to thank Florian Paul Schmidt for uploading the files to the *biological-cybernetics* repo in the Gitlab *Conquaire* group. We would like to thank Lukas Biermann for helping with the reproduction of the analyses in MATLAB.

References

[1] Theunissen LM, Bekemeier HH, and Dürr V. Comparative whole-body kinematics of closely related insect species with different body morphology. *J Exp Biol*, 218:340-352, 2015.

[2] Theunissen LM and Dürr V. Insects use two distinct classes of steps during unrestrained locomotion. *PLOS one*, 8:e85321, 2013.

[3] Dallmann CJ, Dürr V, and Schmitz J. Joint torques in a freely walking insect reveal distinct functions of leg joints in propulsion and posture control. *Proc R Soc*, B 283:20151708, 2016.

[4] Dallmann CJ, Hoinville T, Dürr V, and Schmitz J. A load-based mechanism for inter-leg coordination in insects. *Proc R Soc Lond B Biol Sci*, 284:20171755, 2017.

[5] Ritzmann RE and Büschges A. Adaptive motor behavior in insects. *Curr Opin Neurobiol*, 17:629-636, 2007.

[6] Dürr V, Theunissen LM, Dallmann CJ, Hoinville T, and Schmitz J. Motor flexibility in insects: Adaptive coordination of limbs in locomotion and near-range exploration. *Behav Ecol Sociobiol*, 72:15, 2018.

[7] Cruse H, Dürr V, Schilling M, and Schmitz J. *Principles of insect locomotion. In: Arena P, Patanè L (eds) Spatial temporal patterns for action-oriented perception in roving robots.*, volume pp 43-96. Springer, Berlin, 2009.

[8] Full RJ, Blickhan R, and Ting LH. Leg design in hexapedal runners. *J Exp Biol*, 158:369-390, 1991.

[9] Burrows M. *The Neurobiology of an Insect Brain.* Oxford University Press, Oxford, 1996.

[10] Büschges A. Lessons for circuit function from large insects: towards understanding the neural basis of motor flexibility. *Curr Opin Neurobiol 22:602-608*, 2012.

4 | Reproducing Trajectory Analysis of Bumblebee Exploration Flights

Vineet Sharma[1], Olivier Bertrand[2], Jens Lindemann[2], Cord Wiljes[1], Martin Egelhaaf[2], Philipp Cimiano[1]

1 – Semantic Computing Group, Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University
2 – Faculty of Biology, Bielefeld University

Abstract

This chapter describes a case study in using a combination of virtualization technology, Git as well as a continuous integration (CI) server to support sustainability of analytical pipelines. The case study was designed to reproduce one processing step in the analytical pipeline described in the paper *"Taking a goal-centered dynamic snapshot as a possibility for local homing in initially naive bumblebees"* [1]. In this paper, the researchers report their findings regarding the exploratory flights of bumblebees in unknown territories. Trajectories were recorded using two cameras and triangulated, yielding 3D trajectories of the flights. The original analytical workflow was implemented in MATLAB. As a result of Conquaire, the analytical workflow could be reproduced using Python, yielding trajectories that faithfully match the original trajectories. In Conquaire, we implemented an analytical workflow that relies on virtualization as well as on a continuous integration server. The main function of the virtualization is to preserve the computational environment so that it can be easily executed by third parties without the need to reproduce the exact computational environment nor to install any libraries. A continuous integration server was used to implement basic mechanisms for quality control over the data, leading to the discovery of some minor mistakes that could be directly corrected. The case study has demonstrated the usefulness of using a combination of virtualization and continuous integration to support analytical reproducibility in the natural sciences, neuroethology in particular.

Keywords

Insect spatial locomotion, bumblebee flights, analytical reproducibility, virtualization, continuous integration

4.1 Introduction

Animals move in their environment in a quest for food, a mating partner, or a place to raise their offspring. The animals, therefore, need to solve spatial tasks, viz. orientating themselves, identifying and reaching a target (such as a mating partner or a food source), following habitual routes (for e.g., between their home and food sources). Even in cluttered environments, animals manage to solve these complex spatial tasks without collisions with obstacles in their path. These abilities are not only observed in vertebrates but also in insects with small brains. Indeed, flying insects can chase their partner [2], learn the surroundings of their nest [3, 1], cross cluttered environments [4, 5], and follow routes [6, 7]. Given the small number of nerve cells in insect brains and the limited reliability of neurons in general, extracting information required to solve navigational tasks needs to rely on extremely efficient neural mechanisms. As a consequence of millions of years of evolution, these mechanisms are tightly linked to the sophisticated locomotion and gaze strategies of insects.

The research focus of the Neurobiology group at Bielefeld University is to elucidate the computational principles, down to the level of neurons and neural networks that generate and control visually guided behaviour in complex and cluttered environments. Understanding the computational principles involved in visually guided behaviour requires, first, monitoring the behaviour of the animal over long periods, and second, reconstructing the visual perception of the environment from the animal's perspective.

The visual processing and behaviour of insects is extremely fast, and hence monitoring their behaviour and reconstructing it requires high frequency and precision recording techniques to obtain the position and orientation of the animal. The position of an animal in an environment can be accurately derived via triangulation or 3D reconstruction of high-frequency data from video recordings of the animal taken with several synchronized cameras. This method requires a precise orientation and positioning of the camera, as well as a correction of potential distortions due to the lens, and an accurate detection of the position and orientation of the insect on the camera (obtained by feature extraction). However, no tracking software is error-free, and thus, the recording even after manual reviewing may contain errors (especially for extended recordings, e.g. of several 10,000 frames) that need to be automatically post-processed by a later processing stage.

Lobecke et al. [1] recorded the behaviour of naive bumblebees exiting their nest for the first time. This behaviour can last for several minutes, and the monitoring of the animal's behaviour resulted in the collection of several thousand images on which the bumblebees' positions were automatically tracked and manually reviewed. The orientations of the bumblebees during their learning flights were obtained from the recorded positions using the Camera Calibration Toolbox from MATLAB [8]).

In this chapter, we discuss a case study in applying a combination of continuous integration principles, virtualization and Git to support reproducibility of one computational step in the experimental pipeline described by Lobecke et al. [1]. Our main motivation for this case study is to develop best practices that support the execution of the original analytical workflow by third parties. For this reason, we explore how virtualization technology can be used to create a reproducible computational environment that can be directly executed without the need to install software. An approach based on virtualization prevents problems related to broken dependencies due to later non-availability of the required version of software and packages. In addition to using virtualization, we make use of an integration server to specify and execute a number of integrity tests that ensure validity of the data.

The structure of this chapter is as follows: in the following section 4.2, we describe how the data in the original study by Lobecke et al. was collected. In section 4.3, we describe the technical environment we have set up to preserve the computational environment and thus ensure executability of the analytical workflow. We also describe how we have used continuous integration (CI) principles to implement a set of quality checks and integrity tests that ensure the validity of the data.

4.2 Experiment settings and data acquisition pipeline

The behaviour of naive bumblebees was recorded with two cameras (Falcon2 3M, Teldyne Dalsa, Inc) at 148fps, an exposure time of 1/1000s and a spatial resolution of 2048x2048 px. The focal lens of the cameras was 8mm, and the physical pixel size was 6 μm. The behaviour of bumblebees was continuously monitored for several hours on a hard disk array using the software Marathon Pro (GS Vitec, Germany). Relevant sequences of learning flights were stored as 8-bit jpeg images for the flight analyses. From the series of images, the position of the bumblebee on the image was obtained by segmenting the image into background and foreground and fitting an ellipse around the foreground (the bumblebee) by using the software ivTrace[9][1]).

After this automated procedure, the position and orientation of the bumblebee on the images were manually reviewed and potential errors were corrected by watching the video frame by frame and using the software ivTrace. In a paral-

[1]https://opensource.cit-ec.de/projects/ivtools

lel step, the Camera Calibration Toolbox for MATLAB by Jean-Yves Bouguet[2] was used for the camera calibration and the 3D stereo triangulation. A checkerboard pattern (5 cm per square) was used for the calibration and the difference between checkerboard points recorded by the camera and checkerboard points reprojected to the images from their triangulated 3D positions was determined. The average position error for the top and the side camera were 0.11 and 0.09 px, respectively.

Lobecke et al. [1] reported that the first learning flights of bumblebees are highly variable and depend on the recorded individual. The learning flight was recorded along a prolonged time-span and at a high spatio-temporal resolution. The bumblebees' flight positions and orientations were then reconstructed by using triangulation from two synchronized cameras. Fig. 4.1 depicts the computational workflow of the calibration to triangulation process. Fig. 4.3 depicts an example of a trajectory of a bumblebee flight. For more detailed depiction, see [1].

All data files, MATLAB and Python scripts for analysis as listed in Fig. 4.1 were made available by the Neurobiology group. The XML-file (Fig. 4.7) contains parameters of the camera that were used for recording the bee flight movement. They are used in the triangulation process to calculate trajectories using two *tra format* files. The dataset is the basis for a publication by Lobecke et al. (2018) [1]. The tra files (Fig. 4.8) contain the trajectory values in 2D format from two cameras, one located on top and the other located on the side of the bee. The MATLAB file format contains resulting trajectory information in 3D format.

[2]`http://www.vision.caltech.edu/bouguetj/calib_doc/`

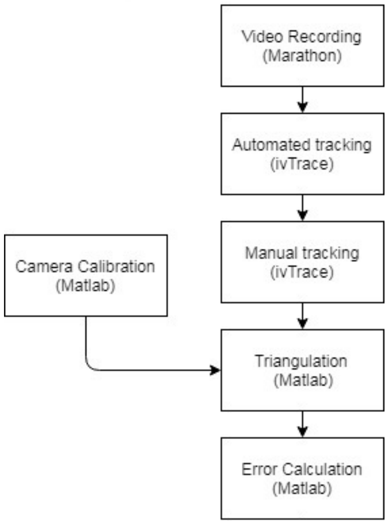

Video Recording (Marathon)
Automated tracking (ivTrace)
Manual tracking (ivTrace)
Camera Calibration (Matlab)
Triangulation (Matlab)
Error Calculation (Matlab)

Figure 4.1: Procedure to calculate the trajectories of bumblebee flights, original procedure as described in Lobecke et al. [1]

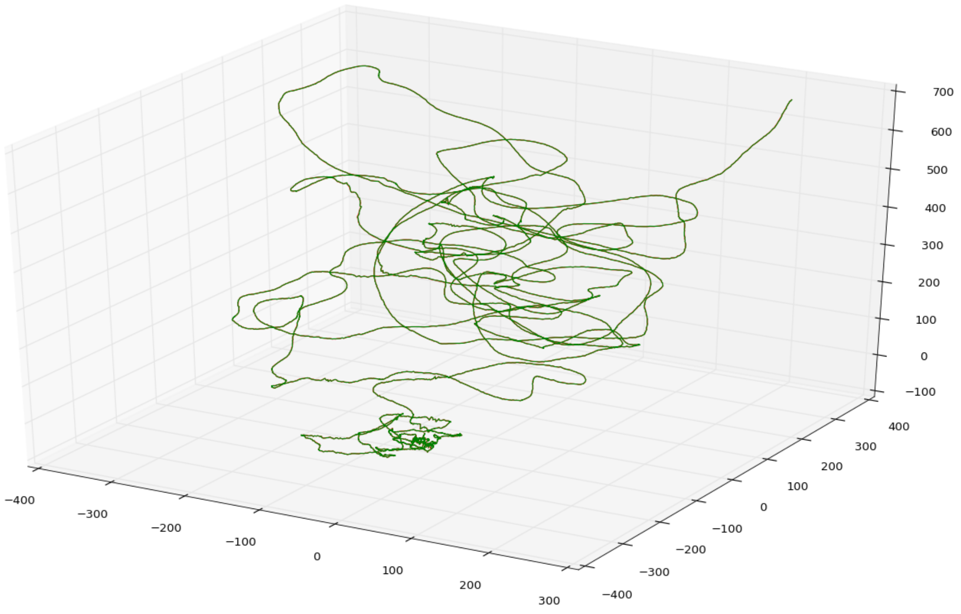


Figure 4.2: Example trajectory of a bumblebee flight, seen in 3D (cf. Lobecke et al. [1])

4.3 Computational Environment for Reproducibility

In this case study, we set up a computational environment that builds on three key components to support 3rd party execution of the analytical pipeline for computing the 3D trajectories:

- Git Repository: The original data and the scripts to compute 3D trajectories from the 2D data of the two cameras were uploaded to a Git repository. The benefit of using Git is that data and scripts are stored in a versioned fashion so that particular versions of data and scripts can be referenced. Further, the data is backuped.
- Virtualization: We rely on virtualization technology to create a virtual image of the computational environment that can be shared and executed on any machine that runs the same virtualization software. In our case, we rely on VMWare.
- Continuous Integration: We deploy a continuous integration server that pulls the data and scripts from the Git repository, builds the analytical pipeline, and executes a number of integrity tests on the data.

In the following, we describe the virtualization and continuous integration approach in more detail. Before, however, we briefly describe how the original MATLAB code that was used in the original experiment was migrated to an open source programming language, Python in particular.

4.3.1 Software Migration

The original code used in the study carried out by Lobecke et al. was written using the commercial software MATLAB. As part of Conquaire, the scripts were ported to the open source programming language Python. Some data files remained in MATLAB format, which did not constitute a problem as Python's scipy library can be used to read in MATLAB files. The resulting Python code is available in a shared GitLab repository[3]. The Python script reads the position of the bees from the two cameras, performs the triangulation for the two camera images and produces the 3D trajectories as output. Note, that the reconstruction of the camera calibration from the data as depicted in Fig. 4.3 was only necessary for reproduction purposes. For future data, the calibration parameters for the python scripts would also be generated from a checkerboard calibration processes.

Using this Python script, we could successfully reproduce the 3D trajectories from the original experiment. Fig. 4.4 plots the 2D projection of the 3D

[3]`https://GitLab.ub.uni-bielefeld.de/olivier.bertrand/tra3dpy`

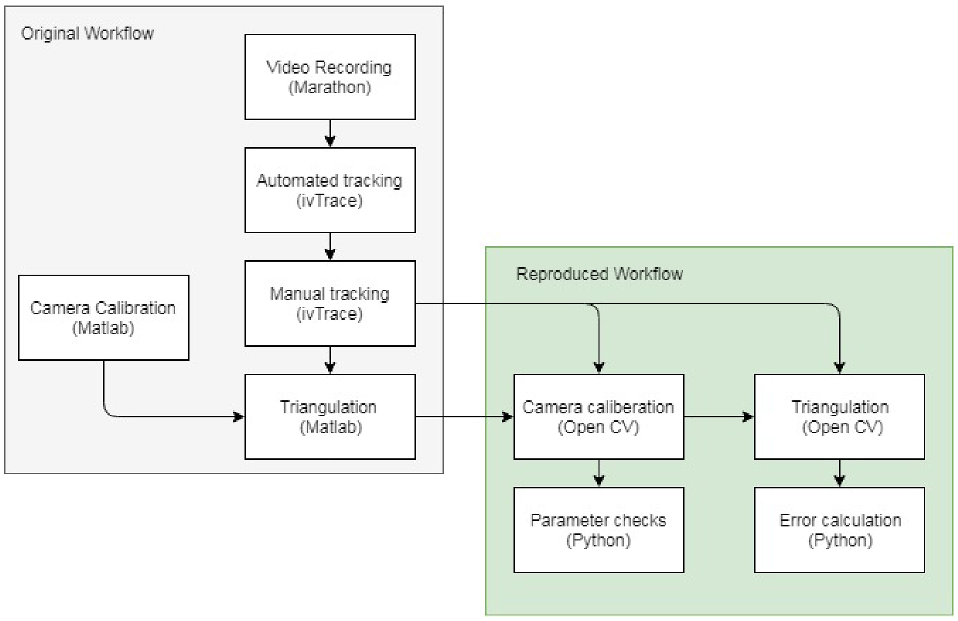

Original Workflow
Video Recording (Marathon)
Automated tracking (ivTrace)
Manual tracking (ivTrace)
Camera Calibration (Matlab)
Triangulation (Matlab)
Reproduced Workflow
Camera caliberation (Open CV)
Triangulation (Open CV)
Parameter checks (Python)
Error calculation (Python)

Figure 4.3: Procedure to obtain the trajectories of bumblebees. In shaded gray: The original procedure followed in Lobecke et al. [1]. In shaded green: the reproduced and adapted procedure. In parenthesis, the software/tools used to accomplish the task.

trajectories computed by the original MATLAB workflow in comparison to the Python-based workflow. One can appreciate that the deviations are minor and barely visible. A statistical analysis of the differences for all 18 investigated flight experiments is shown in Fig. 4.5. The average error along x- and y-axis is a maximum of 0.024 mm and is much smaller than the maximum error of measurement and therefore negligible. In contrast, the average error along the z-axis is larger (0.3 mm). However, the differences are clearly small and within an acceptable range.

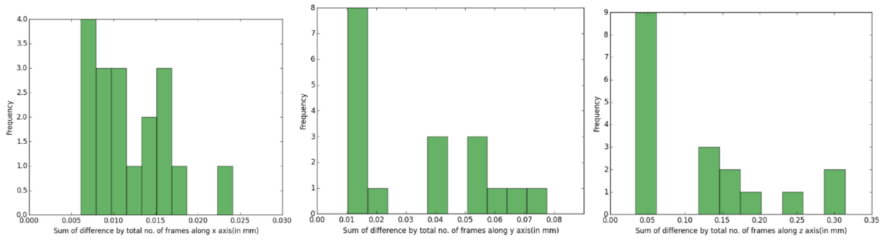


Figure 4.5: Distribution of differences between original MATLAB and new Python calculation for the three dimensions, x, y and z respectively

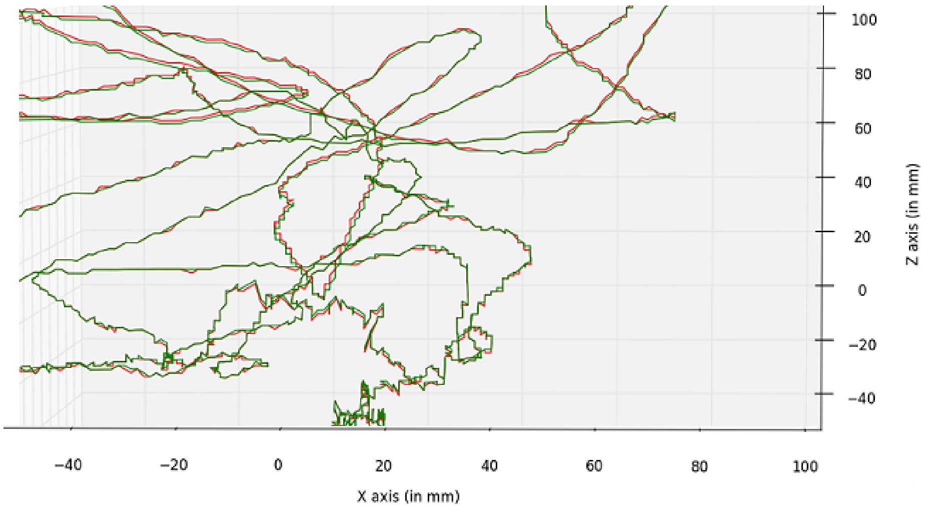


Figure 4.4: A close magnified snapshot displaying comparing the 2D projections to x- and z-axes of the 3D trajectories computed by the MATLAB analytical workflow (red) and the Python-based workflow (green).

4.3.2 Virtualization

A virtual machine was set up with the necessary libraries and dependencies required to run the toolbox. A linux-based virtual environment was created using VMWare. The virtual machine was provided with 2GB RAM and 50GB of storage. The CI server Jenkins was installed and the Python environment needed to execute the Python tool mentioned above was setup. In particular, Python version 3.4 was installed. The benefit of the virtualization is that the computational workflow can be executed by a third party without any need for installing operating systems, software nor libraries except for setting up a machine that runs VMWare and that supports execution of the virtual image. Thus, the party interested in running the computational workflow does not have to take care of installing any packages with the correct version. Further, the workflow can be executed in spite of the specific version of the libraries on which the script depends not being available anymore.

4.3.3 Continuous Integration supporting quality control

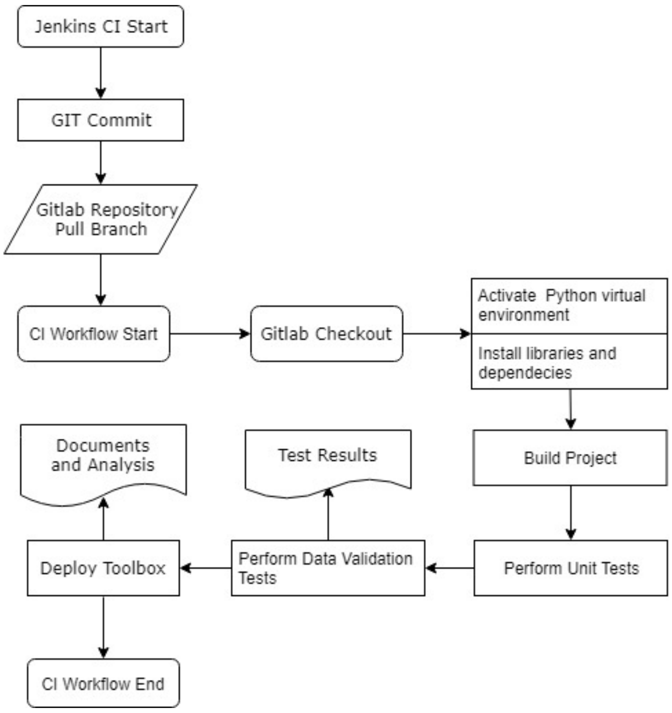
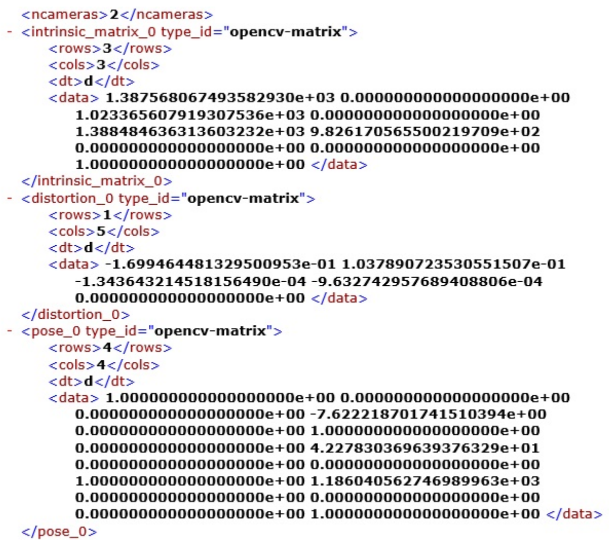


Figure 4.6: Flow Chart of Jenkins Continuous Integration pipeline.

As mentioned above, a Jenkins server was installed and deployed on the virtual machine. The Jenkins server is used to automate the process of checking out the toolbox from the Git repository and deploying the analytical pipeline in the local (virtual) machine. It allows to deploy the toolbox in a repeatable and reliable way involving automated testing. The CI workflow has been implemented in such a way that it continuously checks the Git repository for new changes and executes the whole pipeline every time the data and/or scripts have been updated. The workflow also installs all the necessary Python libraries using the pip package manager. The whole pipeline is depicted in Figure 4.6. After starting the Jenkins Server and starting the workflow, the data and scripts are checked out from the Git repository. Then, the necessary Python libraries are installed on the virtual machine and the project is build. A number of unit tests are performed on the software. Then, a number of data validation tests are executed and the test results are stored in a log. When all tests are passed, the toolbox is run on the data and the results of the analysis are stored.

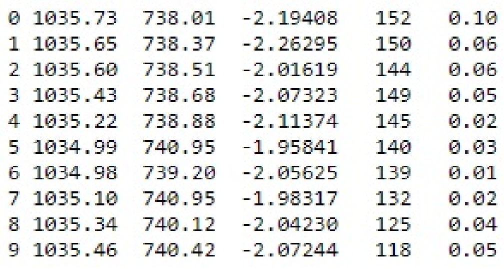
Data validation tests were written for the three types of files:

- XML file: The XML file describes parameters of the camera used when



```
  <ncameras>2</ncameras>
- <intrinsic_matrix_0 type_id="opencv-matrix">
      <rows>3</rows>
      <cols>3</cols>
      <dt>d</dt>
      <data> 1.387568067493582930e+03 0.000000000000000000e+00
         1.023365607919307536e+03 0.000000000000000000e+00
         1.388484636313603232e+03 9.826170565500219709e+02
         0.000000000000000000e+00 0.000000000000000000e+00
         1.000000000000000000e+00 </data>
  </intrinsic_matrix_0>
- <distortion_0 type_id="opencv-matrix">
      <rows>1</rows>
      <cols>5</cols>
      <dt>d</dt>
      <data> -1.699464481329500953e-01 1.037890723530551507e-01
         -1.343643214518156490e-04 -9.632742957689408806e-04
         0.000000000000000000e+00 </data>
  </distortion_0>
- <pose_0 type_id="opencv-matrix">
      <rows>4</rows>
      <cols>4</cols>
      <dt>d</dt>
      <data> 1.000000000000000000e+00 0.000000000000000000e+00
         0.000000000000000000e+00 -7.622218701741510394e+00
         0.000000000000000000e+00 1.000000000000000000e+00
         0.000000000000000000e+00 4.227830369639376329e+01
         0.000000000000000000e+00 0.000000000000000000e+00
         1.000000000000000000e+00 1.186040562746989963e+03
         0.000000000000000000e+00 0.000000000000000000e+00
         0.000000000000000000e+00 1.000000000000000000e+00 </data>
  </pose_0>
```

Figure 4.7: Camera calibration data in XML format



```
0 1035.73  738.01  -2.19408   152   0.10
1 1035.65  738.37  -2.26295   150   0.06
2 1035.60  738.51  -2.01619   144   0.06
3 1035.43  738.68  -2.07323   149   0.05
4 1035.22  738.88  -2.11374   145   0.02
5 1034.99  740.95  -1.95841   140   0.03
6 1034.98  739.20  -2.05625   139   0.01
7 1035.10  740.95  -1.98317   132   0.02
8 1035.34  740.12  -2.04230   125   0.04
9 1035.46  740.42  -2.07244   118   0.05
```

Figure 4.8: The tra format. The rows are in the following format: frame number, x, y, orientation, roundness, size

recording the bees' flight movements (see 4.7 for a sample). We implemented a parser that checks the syntactic well-formedness of the XML file. In addition, we implemented a set of basic tests checking that the x- and y-position of the center of the camera is within acceptable ranges. The test succeeds if the center of both cameras is less than half of the size of the camera. Finally, we wrote a test to check that the focal length parameter of the camera is within acceptable ranges.

- tra files: The tra files contain 2D trajectory values of the bees' flights as recorded by the two cameras. A set of unit tests was implemented to check that there are no empty values for any row/column as well as that each value is of numeric type. In addition, we implemented checks to verify that the values are within acceptable ranges as specified for each column.

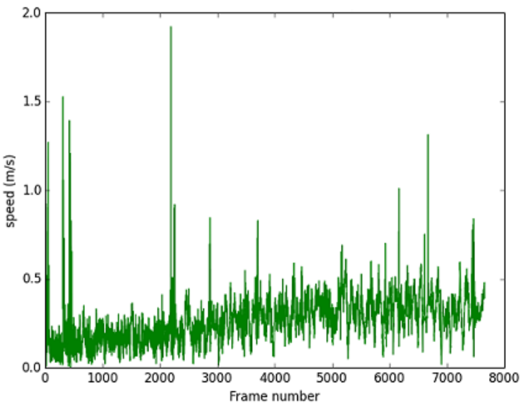


Figure 4.9: Flight speed for a bee which is well within the range as defined by the researcher (below 10m/s).

A sample of the data is shown in Figure 4.8.

- MATLAB files: The MATLAB files contain the 3D trajectories as calculated from the 2D files using a triangulation mechanism described by Lobecke et al. [1]. We implemented a test that computes the distance between any subsequent 3D data points and computes the bumblebee's speed from the distance and frames per second as recorded by the cameras. The test is passed if the speed is below the maximum of 10m/s.

These tests were intended to validate the data by discovering potential errors. The XML file with the camera parameters passed all the tests. Our validation scripts highlighted that some rows in the tra files had missing values and that some rows had 11 (instead of 6) values. In the case of the MATLAB files, some tests were not passed as for a number of data points the bumblebee's flight speed was observed to be out of the possible range (Fig. 4.9 and Fig. 4.10). Overall, this validation helped the researchers to discover small errors in the data and correct them.

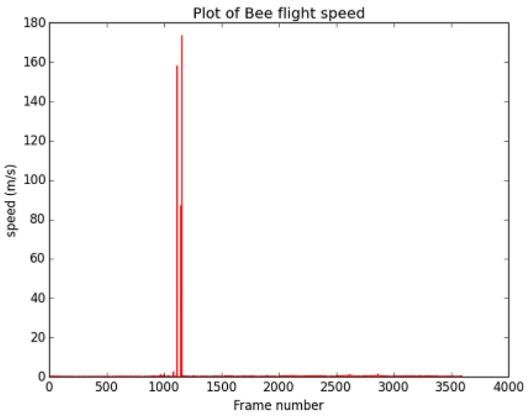


Figure 4.10: Flight speed for a bee in which an error in the data was found. The erroneous speed was above 150 m/s, which is far outside the acceptable range.

4.4 Conclusion

We have described a case study in applying a combination of continuous integration principles, virtualization and Git to support reproducibility of one computational step within an experiment in neurobiology studying the first flights of bumblebees. Git supports the versioned storage of data and scripts so that we can refer back to any version of the data if needed. Virtualization technology allows to preserve the computational environment in order to avoid a situation in which the software can not run any more due to broken dependencies, non-availability of the particular version of a required software, etc. Third party researchers can re-run the computational procedure by merely installing the image of the virtual machine, without having to install any further software or having to built it. A continuous integration server has been deployed on the virtual machine to automatically pull the most recent version of the data on the repository, build the computational pipeline and run a number of tests that check the well-formedness of the data.

In the specific use case considered, the use of virtualization and continuous integration might be considered an overkill as the processing scripts in Python that calculate the 3D trajectories have a limited complexity. The quality tests implemented are also rather simple. Yet, our goal has been to understand the potential of using virtualization and continuous integration, also with respect to more complex cases and experimental environments in which more complex software artifacts and analytical pipelines are involved. In the specific case study considered, we could successfully re-run one computational step from the experimental settings described in the paper *"Taking a goal-centered dynamic snapshot as a possibility for local homing in initially naive bumblebees"* [1]. In particular, we could rerun the step that calculates and visualizes the trajectories of bumblebees. In this sense we could reproduce a key step in the analysis of the recorded flights.

A drawback of our proposed architecture and combination of virtualization, continuous integration and Git is that the data resides on a Git repository and is pulled every time the computational pipeline is deployed and tested by the continuous integration server. While this allows to pull the most recent version of data and scripts, in our experience once the data and scripts are final, they are typically not modified so that a static inclusion of the data and scripts in the virtual machine would be sufficient. The dependency on a Git repository introduces a dependency that can potentially break if the Git server is not hosted anymore. In future work, the potential and benefits of using virtualization in combination with a continuous integration server should be further investigated on additional use cases. Especially, using the CI pipeline for continuous quality control on newly recorded data in follow-up projects would be highly beneficial for neuroethological research.

References

[1] Anne Lobecke, Roland Kern, and Martin Egelhaaf. Taking a goal-centred dynamic snapshot as a possibility for local homing in initially naïve bumblebees. *The Journal of experimental biology*, 221(Pt 2):jeb.168674, jan 2018.

[2] Norbert Boeddeker, Roland Kern, and Martin Egelhaaf. Chasing a dummy target: smooth pursuit and velocity control in male blowflies. *Proceedings. Biological sciences*, 270(1513):393–9, feb 2003.

[3] Théo Robert, Elisa Frasnelli, Natalie Hempel De Ibarra, and Thomas S Collett. Variations on a theme: Bumblebee learning flights from the nest and from flowers. *Journal of Experimental Biology*, 2018.

[4] J. D. Crall, S. Ravi, A. M. Mountcastle, and S. A. Combes. Bumblebee flight performance in cluttered environments: effects of obstacle orientation, body size and acceleration. *Journal of Experimental Biology*, 218(17):2728–2737, sep 2015.

[5] Roland Kern, Norbert Boeddeker, Laura Dittmar, and Martin Egelhaaf. Blowfly flight characteristics are shaped by environmental features and controlled by optic flow information. *The Journal of experimental biology*, 215(Pt 14):2501–2514, jul 2012.

[6] Joseph L Woodgate, James C Makinson, Ka S Lim, Andrew M Reynolds, and Lars Chittka. Life-Long Radar Tracking of Bumblebees. *PloS one*, 11(8):e0160333, 2016.

[7] Mathieu Lihoreau, Lars Chittka, and Nigel E Raine. Travel optimization by foraging bumblebees through readjustments of traplines after discovery of new feeding locations. *The American naturalist*, 176(6):744–57, dec 2010.

[8] J.Y. Bouguet. Matlab camera calibration toolbox. 2000.

[9] Jens Peter Lindemann. *Visual navigation of a virtual blowfly*. PhD thesis, Bielefeld University, Bielefeld, Germany, 2005.

5 | Reproducing experiments of ice nucleation in atmospheric chemistry

Fabian Herrmann[1], Evelyn Jantsch[2], Philipp Cimiano[1], Thomas Koop[2]

1 – Semantic Computing Group, Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University
2 – Faculty of Chemistry, Bielefeld University

Abstract

This chapter describes a case study in reproducing results in the area of atmospheric chemistry. The specific result reproduced is described in the paper *'BINARY: an optical freezing array for assessing temperature and time dependence of heterogeneous ice nucleation'* by Budke and Koop [1]. The study investigated the conditions under which ice nucleation occurs using Snomax®, a commercial ice inducer containing freeze-dried nonviable bacterial cells from *Pseudomonas syringae*, as a test substance for the investigation of heterogeneous ice nucleation processes. The ice inducing bacterial cell agents are known to be active at high temperature and are used in snow cannons. The study considered a temperature range between 0°C and -12°C. The main result was the finding that two classes of nucleations occur at a number ratio of about 1 to 1000 in the chemical samples, based on the difference of 3 orders of magnitude of the temperature plateau values. As a result of the Conquaire project, we reimplemented the original workflow relying on OriginPro in Python and could reproduce the central figure of the above mentioned paper by Budke and Koop using free and open software. This thus counts as a case of full analytical reproducibility. The data and scripts for the paper by Budke and Koop are available at `https://gitlab.ub.uni-bielefeld.de/conquaire/atmospheric_chemistry`.

Keywords

Atmospheric Chemistry, Ice Nucleation, Pseudomonas syringae, Snomax

5.1 Introduction

The study of ice formation is an active research area in the atmospheric sciences [2]. For example, ice crystals occur in high altitude clouds and they are responsible for initiating most precipitation above continents [2, 3, 4]. From a thermodynamic point of view, crystalline ice is the stable phase of water below the melting temperature T_m, which is 0°C at ambient pressure, see Figure 5.1. In many cases, the formation of ice crystals is kinetically inhibited and can occur at lower temperature either via homogeneous or via heterogeneous nucleation, see Figure 5.1.

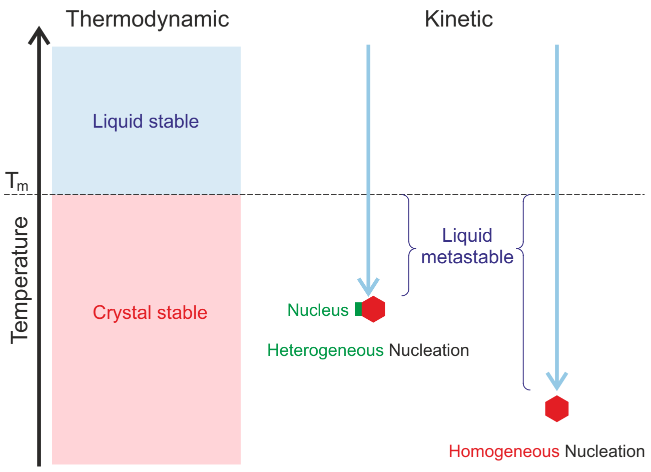


Figure 5.1: Schematic depiction of different nucleation mechanisms for the freezing of water. T_m is the melting temperature of the crystalline phase ice; adapted with changes from Koop, 2004 [5].

For homogeneous freezing, a number of water molecules have to arrange themselves into an ice-like cluster, termed critical ice embryo, in order to trigger ice formation. The size of this critical embryo is temperature dependent and decreases with decreasing temperature, thus making ice nucleation more likely at lower temperature. For example, micrometer-sized pure water droplets freeze homogeneously at approximately -38°C (homogeneous nucleation temperature) [5]. In contrast, heterogeneous ice nucleation can occur at higher temperatures – even close to the melting temperature of ice – depending upon the presence and activity of so-called ice nuclei (IN), see Figure 5.1 [5, 3, 6, 7]. Laboratory experiments can be employed to help understanding the processes that lead to ice nucleation in the atmosphere. By investigating ice nucleation temperatures of different IN, we can quantify different IN activities, which can be used for parametrizations of ice formation in atmospheric cloud models [8].

The activity of an IN material suspended in a water droplet can be obtained from the measured number of active sites per dry mass $n_m(T)$ as a function of temperature. Equation 5.1 presents a definition for $n_m(T)$, where T is temperature, $K(T)$ is the experimentally observed cumulative number of active sites per volume of water, and C_m is the mass concentration of IN in the water.

$$n_m(T) = \frac{K(T)}{C_m} \tag{5.1}$$

$K(T)$ can be obtained from equation 5.2.

$$f_{ice}(T) = \frac{n_{ice}(T)}{n_{tot}} = 1 - e^{-K(T)\cdot V_{drop}} \tag{5.2}$$

Here, $f_{ice}(T)$ represents the experimentally observed cumulative ice fraction, which is defined by the ratio of the number of droplets frozen at temperature T, $n_{ice}(T)$, and the total number of investigated droplets, n_{tot}. V_{drop} is the droplet volume.

Established methods for the determination of heterogeneous ice nucleation temperatures have different advantages and disadvantages. For instance, larger droplet volumes encounter a higher probability of impurities. In contrast, smaller volumes are often realized through emulsions and, therefore, an oil phase is in contact with the water droplet, which may influence results for those IN (e.g. pollen and fungal spores) which have an affinity to hydrophobic phases, i.e. the concentration of suspended IN would be overestimated in such cases [9]. Many experimental techniques are droplet arrays based on a method originally developed by Vali and Stansbury, where small volume droplets are placed on a cooling stage [10, 11]. However, since no oil phase is used to enclose the droplets, frozen droplets grow by water vapor transport from the remaining supercooled liquid droplets, according to the Wegener-Bergeron-Findeisen process. Moreover, sometimes frost halos form around frozen droplets. These ice rings tend to grow on the surface below the droplets and can cause ice nucleation in adjacent supercooled droplets. Budke and Koop [1] took these potential shortcomings into account when they developed a new device to investigate ice nucleation termed BINARY (short for Bielefeld Ice Nucleation ARraY), which was used in the present study. The different droplets in BINARY are separated in individual compartments thus preventing water vapor transfer between neighboring droplets.

Snomax® is a well-studied IN material and, therefore, a good reference substance for testing new methods [12, 13, 14]. Snomax® is a commercial product containing freeze-dried cells from *Pseudomonas syringae*, a rod-shaped bacterium living on a variety of plants. *P. syringae* bacteria are known to induce heterogeneous ice nucleation at very high temperatures of approximately -2°C (class A) and also in a temperature range of about -7 to -10°C (class C) [15]. The latter study was a multi-group intercomparison project and also included data from the BINARY setup. Using this setup Budke and Koop determined $n_m(T)$ in a temperature range between 0°C and -12°C [1].

5.2 Methods

In this section, we describe the experimental settings and methods as well as the main results described in the paper by Budke and Koop [1].

5.2.1 Experiment settings and Data acquisition pipeline

In the study under investigation [1], the BINARY technique was used to determine heterogeneous ice nucleation temperatures of Snomax®. Therefore a certain dry mass of Snomax® (m) was suspended in freshly double-distilled water of volume V_{H_2O} to obtain the desired mass concentration $C_m = m/V_{H_20}$ of Snomax® in water. 36 droplets (each $V_{drop} = 1\ \mu$L) of such a suspension were pipetted into the compartments of a polydimethylsiloxane (PDMS) lattice placed on a hydrophobic glass surface, resulting in a 6 x 6 droplet array as shown in Figure 5.2a. The droplet compartments are sealed with another glass slide on top of the PDMS lattice to prevent droplet evaporation (see Figure 5.2b).

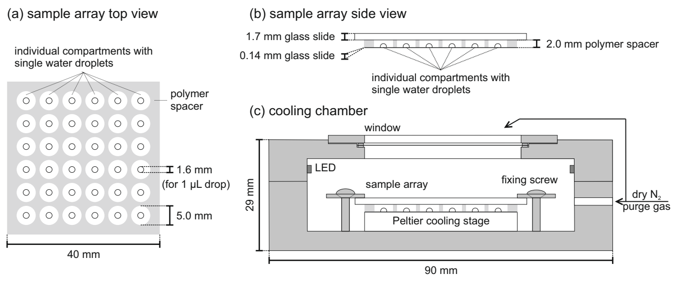


Figure 5.2: Schematic picture of the Bielefeld Ice Nucleation ARraY (BINARY) setup. **(a)** Top view of the 6 x 6 droplet array. The droplets are separated from each other by a polymer lattice creating individual compartments. **(b)** Side view showing the sealing of the compartments by top and bottom glass slides. **(c)** Position of the sample array on the Peltier cooling stage inside the cooling chamber. Figure is taken from Budke and Koop, 2015 [1].

This sample array is positioned on a Peltier stage within a cooling chamber (Linkam LTS120) as shown in Figure 5.2c. A metal frame presses the whole array onto the Peltier cooling stage with the help of fixing screws to assure a homogeneous and efficient heat transfer. The Peltier stage is connected to a heat sink bath at 5°C and its top side can be cooled to -40°C at cooling rates between 0.1 and 10°C min^{-1}. All experiments described below were measured at a cooling rate of 1°C min^{-1}. Small cold-light white LED stripes are fixed at the top edges inside the cooling chamber aiding the visualization of phase changes through light scattering (liquid droplets appear darker whereas ice crystals appear brighter due to the backscattered light). A CCD camera (QImaging MicroPublisher 5.0 RTV) is mounted above the whole setup to observe the droplets through a 40 x 40 mm window in the top ceiling of the cooling chamber. Both the interior of the cooling chamber and the surface of the top window are purged with dry nitrogen during the experiment to prevent water condensation. A LabVIEW™ virtual instrument is used to detect ice nucleation and melting events from the digital images obtained by the CCD camera. In detail, for each compartment the average gray value of all pixels within a predefined area is obtained. These gray values range between 0 for black pixels and 255 for white pixels.

Figure 5.3b and c show a representative behavior of the gray values and their changes for the compartment marked by a yellow box in panel (a). Starting with the red curve at 4°C, the average gray value in Figure 5.3b is almost constant until the droplet freezes at -3.9°C, as indicated by a sharp jump. This steep increase is also shown as the derivative in Figure 5.3c. After a temperature of -10°C is reached, heating is started (green curves) and ice melting begins at 0°C, again indicated by a gray value change. The thresholds for defining the occurrence of nucleation and melting events are gray value changes larger than 1 and -1, respectively (dotted red and green lines in Figure 5.3c).

5.2.2 Methods applied to analyze the experimental data

For each individual droplet, the uncalibrated heterogeneous ice nucleation temperatures T_{nuc} are obtained and saved in a text file for offline calibration and further analysis. The temperature calibration function and how it was developed from experiments is discussed in detail in the paper [1]. Briefly, the calibrated nucleation temperature T_{cal} can be obtained using equation 5.3, where r is the cooling rate of 1°C min^{-1}.

$$\begin{aligned} T_{cal} = &-((-6.03165) + 0.02113 \cdot (273.15 + T_{nuc}) - (3.59774 + (-0.02956) \\ &(273.15 + T_{nuc}) + 6.10156 \cdot 10^{-5} \cdot (273.15 + T_{nuc})^2) \cdot (-r) + T_{nuc} \end{aligned} \tag{5.3}$$

Each T_{cal} value is then binned into temperature intervals of 0.1°C width, i.e. all T_{cal} values within the interval $X_{low} \leq T_{cal} < X_{up}$ get sorted into the bin X_{low}. Thereafter, T_{cal} will only be used as the binned value T. Now the number of individual T_{cal} data are counted to gain $n_{ice}(T)$ and n_{tot} for determining f_{ice} using equation 5.2. This counting is done for all droplets with the same Snomax® concentration, so each measured concentration has one cumulative ice fraction ranging from 0 to 1. Using equation 5.4 (derived from equation 5.1 and 5.2) $n_m(T)$ is obtained for each concentration and can be plotted for all investigated temperatures.

$$n_m(T) = \frac{-\ln(1 - f_{ice}(T))}{C_m \cdot V_{drop}} \tag{5.4}$$

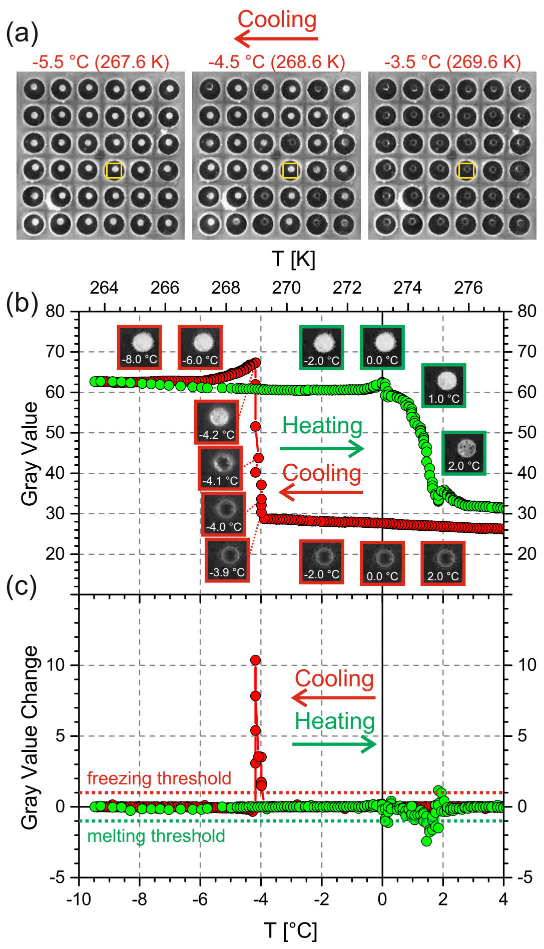


Figure 5.3: Typical experiment with Snomax®-containing droplets (0.1 μg μL^{-1}) showing the automatic detection of ice nucleation events by the change in brightness during freezing. **(a)** Image series of the 6 x 6 droplet array during cooling. **(b)** Measured gray value of the droplet compartment indicated by the yellow box in panel (a) during cooling (red) and heating (green). Freezing and melting start at -3.9°C and 0.0°C, respectively. **(c)** Plot of the change in gray value between successive images showing peaks at the phase transition temperatures. Threshold values of ±1 for the automatic attribution of freezing and melting are indicated by the dashed lines. Figure is taken from Budke and Koop, 2015 [1].

5.2.3 Main Results

Figure 5.4 presents the main result of the paper in form of a combined curve of $n_m(T)$ values from several Snomax® suspensions with different concentrations (see color code).

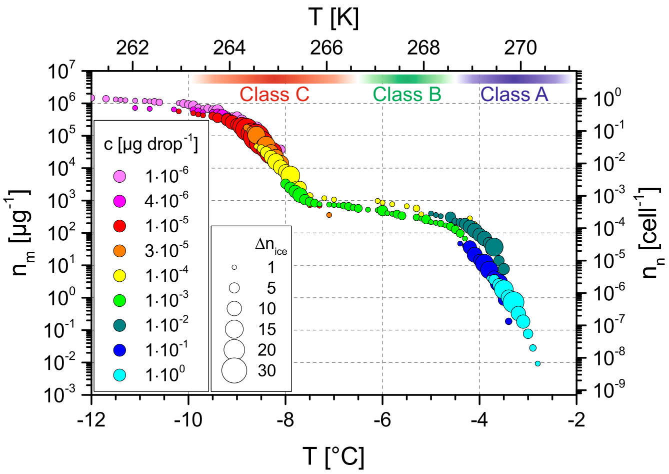


Figure 5.4: Experimentally determined active site density per unit mass of Snomax® $n_m(T)$ versus temperature. Symbol colors indicate data from droplets with different Snomax® concentrations; symbol size indicates the number of nucleating droplets per temperature interval (0.1°C). The temperature range for different classes of IN is also indicated by the colored bars. Figure is taken from Budke and Koop, 2015 [1].

Two steep increases can be seen, which represent two different types of IN activity at different temperature regimes. Plateaus in a $n_m(T)$ plot, e.g. between -4.5°C and -7.5°C, can be interpreted as temperatures where no IN is active. It should be noted that data points below -12°C down to -35°C were obtained, but are not shown since they did not reveal any other IN (purple symbols). Also indicated in Figure 5.4 are the temperature ranges for different IN classes as defined in the literature [16]. Two different classes of IN in Snomax® were identified, inducing ice nucleation at about -3.5°C (class A) and at about -8.5°C (class C). For class A IN in Snomax® $n_m(T)$ ranges from about 10^{-2} μg^{-1} up to almost 10^3 μg^{-1}. However, the number of active sites is much larger for class C IN as the increase starts at 10^3 μg^{-1} rising to almost 10^6 μg^{-1}, indicating that class C IN are about a factor of 10^3 more abundant than class A IN. The number of active sites can also be expressed as a number of active sites per cell (i.e., $n_n(T)$ on the right axis in Figure 5.4). Hence, there is about one class C active site per *P. syringae* cell.

5.3 Analytical Reproducibility

As a main objective of this study, we defined the goal of being able to independently reproduce the plot shown here in Figure 5.4 as main result of the work described by Budke and Koop [1]. The validation was done by calculating the calibrated temperatures from the temperature for a given cooling rate and Snomax® concentration; for each concentration bin, the $f_{ice}(T)$ was calculated. The calculations had been done originally using OriginPro for the original paper, while we reproduced these calculations using a custom Python program.

5.3.1 Research Data - Primary

The data was read off the BINARY experiment setup, then processed entirely by OriginPro, a proprietary computer software from OriginLab Corporation, that is mainly used for interactive scientific graphing and data analysis on the Microsoft Windows platform only. It is a GUI software with a spreadsheet-like front end which uses a column-oriented data processing approach for calculations. It has its own file format, **.OPJ**, for project files which are directly processed by the system for statistics, data analysis and visualization.

The group uses OriginPro along with a scripting language known as **LabTalk** that allows finer control, by writing small macros that run over the data analysis process for the experiment data. With LabTalk the group programs routine operations, including batch operations, with customizable graph templates and analysis dialog box themes. Various features exist to save a collection of operations within the workbook, viz., saving a suite of operations, auto recalculation on changes to data or analysis parameters, and different analysis templates.

5.3.2 Research Data - Analyzed and Processed

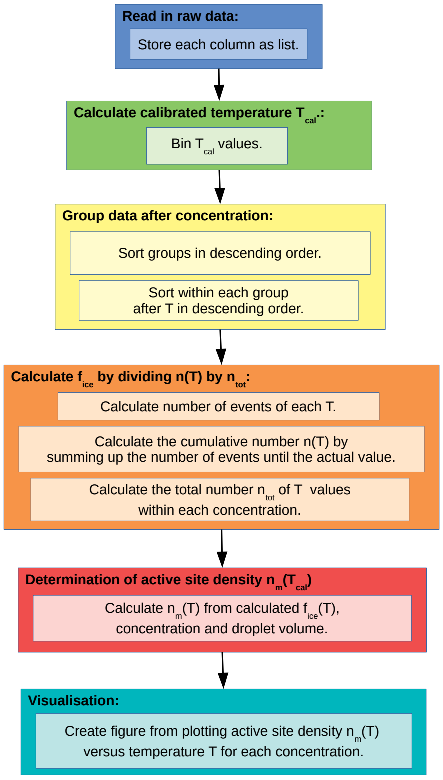
The Snomax® data file contains data from the OPJ data file that is read into the Origin software system. The data was exported into tab-separated files with OriginPro as *.txt files with six TAB delimited columns. The calibration data numbers start from line four with the headers confined to the first three lines; viz. the first line has the column names, while line 2 contains the data description or unit, and the third line contained information about the substance.

For the computational reproducibility experiment, we used Python to process these text files for data analysis and visualization based on the validated raw data. After calibrating the temperature, the python script binned the data, then grouped the data for all columns by concentration (decreasing) into different bins and then within each concentration bin the data is sorted by (decreasing) calibrated temperature T_{cal}. Afterwards, $f_{ice}(T)$ was determined for each temperature value in each bin. In the last step the mass concentration of Snomax® and the volume of the droplets are converted into the active site density per unit mass, $n_m(T)$.

After tabulating $f_{ice}(T)$ and $n_m(T)$ for each concentration bin, the resulting data is stored in a csv file that became the input data to reproduce the plot from the original paper shown in Figure 5.4.

5.3.3 Data Workflow Lifecycle

In order to reproduce the mentioned plot, the functionality implemented originally in the OriginPro frameworks was reproduced using a Python program. The resulting workflow implemented in Python reproduces the workflow implemented in OriginPro and schematically represented in Figure 5.5.



Read in raw data:
Store each column as list.
Calculate calibrated temperature T_cal:
Bin T_cal values.
Group data after concentration:
Sort groups in descending order.
Sort within each group after T in descending order.
Calculate f_ice by dividing n(T) by n_tot:
Calculate number of events of each T.
Calculate the cumulative number n(T) by summing up the number of events until the actual value.
Calculate the total number n_tot of T values within each concentration.
Determination of active site density n_m(T_cal)
Calculate n_m(T) from calculated f_ice(T), concentration and droplet volume.
Visualisation:
Create figure from plotting active site density n_m(T) versus temperature T for each concentration.

Figure 5.5: Schematic representation of analytical workflow as implemented in Python program.

First, the given raw data is read in and each column is saved as a list. In the second step, the calibrated temperature T_{cal} is calculated from the measured temperature T_{nuc} and the cooling rate r with formula 5.3. In the third step, the data is grouped by the concentration C_m into different bins and is sorted in descending order. Within each bin the data is sorted by the temperature T in descending order. In the fourth step, for each bin a new table is generated. The bin is grouped by the temperature T and a second row is introduced showing the number of occurrences of each different binned temperature. A third row is used to summarize the occurrences including the current temperature. It shows the number of droplets up to the current temperature. This value and the total number of all droplets in this bin are used to calculate the frozen fraction $f_{ice}(T)$ with the given formula 5.2. Then it is appended to the table. In the fifth step, the active site density per unit mass, $n_m(T)$ is calculated from $f_{ice}(T)$, the concentration C_m and the droplets volume V_{drop} with the formula 5.4 and is appended as fifth column to the new table. Thereafter, this table is saved as a **.CSV** file, a common data format used by researchers with many tools for file input-output operations. As a second result, the generated table is used to reproduce and plot the graph in Figure 5.6 which displays our graph and the graph from the original paper for comparison. With the given raw data the results from the original experiment could be successfully reproduced using Python, an open source programming language.

5.3.4 Summary of Reproducibility Experiment

We reproduced the results from the analyses from the original paper as shown in the visualization graph Figure 5.6 by plotting $n_m(T)$, the cumulative number of IN per dry mass of Snomax® as a function of calibrated temperature. Origin software is a proprietary analysis toolbox with no equivalent libre software alternative. Hence, the original OPJ data file format can only read data into the Origin software system. The system allows data to be exported into tab-separated files with delimited columns. Due to the complexity and time associated with learning to use a new system like Origin, we opted to use Python to code the formulae and run the data files to be analyzed. In addition, Python is open source and is supported by many platforms.

Two particularly strong increases in $n_m(T)$ are observed, one at about $-3.5\,°\text{C}$ $(269.6K) \pm 0.5K$ and one at $-8.5\,°\text{C}(264.6K) \pm 0.5K$, indicating the presence of two distinct classes of ice nucleators with different activation temperatures.

The two plateaus at temperatures just below each increase of $n_m(T)$ in Figure 5.6 arise when no INs are active at these temperatures in the investigated suspensions. The $n_m(T)$ values of the plateaus differ by about 3 orders of magnitude, from which it can be inferred that the two classes of Snomax® ice nulceations occur at a number ratio of about 1 to 1000 in the samples. The active site densities per cell $n_n(T)$, shown in Figure 5.6 on the right axis, were calculated using the specific particle number of cells in Snomax®.

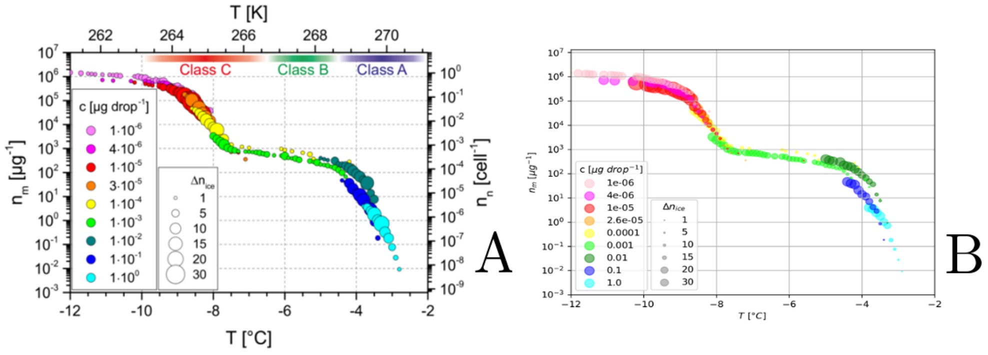


Figure 5.6: Experimentally determined active site density per unit mass of Snomax® $nm(T)$ versus temperature. A: Original version of diagram as published by Budke and Koop [1]; B: diagram resulting from reproducing the computational workflows of Budke and Koop as described in this paper. Symbol colors indicate data from droplets with different Snomax® concentrations; symbol size indicates the number of nucleating droplets per temperature interval (0.1°C). The temperature range for different classes of IN is also indicated by the colored bars.

5.4 Conclusion

In this work we could successfully reproduce the main results of the paper by Budke and Koop [1], reproducing the original analytical workflow using OriginPro by using free and open software, in this case a Python program implemented as part of the Conquaire project. Here, we thus have a case of limited reproducibility as the direct reproduction would have required obtaining a commercial license for OriginPro and re-creating the GUI interactions used in the original work. Instead, we have opted for a re-implementation of the original analysis in Python. We have thus not directly reproduced the original workflow, but developed a workflow that can be regarded as functionally equivalent. As we did not reproduce the original workflow exactly, we have a case of limited analytical reproducibility as defined in chapter 1 of this book. The data has been uploaded to the DFG FOR1525 project website (https://www.ice-nuclei.de/), where it is available upon request. Moreover, the data has been verified by an intercomparison paper by Wex et al. [15]. As a result of Conquaire, both the data and the script are available in a Git repository for further re-use and verification. While there is not yet a DOI for the dataset, the dataset and script are referenceable via a GIT repository, even down to a particular version.

Acknowledgments

We thank Carsten Budke for providing the data and technical discussions during the computational reproducibility process.

References

[1] C. Budke and T. Koop. BINARY: an optical freezing array for assessing temperature and time dependence of heterogeneous ice nucleation. *Atmospheric Measurement Techniques*, 8(2):689–703, 2015.

[2] Hans R Pruppacher and James D Klett. *Microphysics of Clouds and Precipitation.* Kluwer Academic Publishers, New York, 2 edition, 1997.

[3] Will Cantrell and Andrew Heymsfield. Production of ice in tropospheric clouds: A review. *Bulletin of the American Meteorological Society*, 86(6):795–807, 2005.

[4] Dennis Lamb and Johannes Verlinde. *Physics and Chemistry of Clouds.* Cambridge University Press, Cambridge, 2011.

[5] T Koop. Homogeneous Ice Nucleation in Water and Aqueous Solutions. *Z. Phys. Chem*, 218:1231–1258, 2004.

[6] P J DeMott, A J Prenni, X Liu, S M Kreidenweis, M D Petters, C H Twohy, M S Richardson, T Eidhammer, and D C Rogers. Predicting global atmospheric ice nuclei distributions and their impacts on climate. *Proc. Natl. Acad. Sci. U. S. A.*, 107(25):11217–11222, 2010.

[7] B J Murray, D O'Sullivan, J D Atkinson, and M E Webb. Ice nucleation by particles immersed in supercooled cloud droplets. *Chem. Soc. Rev.*, 41(19):6519–6554, 2012.

[8] C Hoose and O Möhler. Heterogeneous ice nucleation on atmospheric aerosols: a review of results from laboratory experiments. *Atmos. Chem. Phys.*, 12(20):9817–9854, 2012.

[9] B G Pummer, H Bauer, J Bernardi, S Bleicher, and H Grothe. Suspendable macromolecules are responsible for ice nucleation activity of birch and conifer pollen. *Atmos. Chem. Phys.*, 12(5):2541–2550, 2012.

[10] Gabor Vali and E J Stansbury. Time-dependent Characterstics of the Heterogeneous Nucleation of Ice. *Canadian Journal of Physics*, 44(3):477–502, mar 1966.

[11] Gabor Vali. Supercooling of Water and Nucleation of Ice (Drop Freezer). *Am. J. Phys.*, 39(10):1125, 1971.

[12] L R Maki, E L Galyan, M M Chang-Chien, and D R Caldwell. Ice nucleation induced by pseudomonas syringae. *Appl. Microbiol.*, 28(3):456–459, sep 1974.

[13] G Vali, M Christensen, R W Fresh, E L Galyan, L R Maki, and R C Schnell. Biogenic Ice Nuclei. Part II: Bacterial Sources. *J. Atmos. Sci.*, 33(8):1565–1570, aug 1976.

[14] O Möhler, D G Georgakopoulos, C E Morris, S Benz, V Ebert, S Hunsmann, H Saathoff, M Schnaiter, and R Wagner. Heterogeneous ice nucleation activity of bacteria: new laboratory experiments at simulated cloud conditions. *Biogeosciences*, 5(5):1425–1435, 2008.

[15] Heike Wex, S Augustin-Bauditz, Yvonne Boose, Carsten Budke, Joachim Curtius, Karoline Diehl, Axel Dreyer, Fabian Frank, Susan Hartmann, Naruki Hiranuma, Evelyn Jantsch, Zamin a. Kanji, Alexei Kiselev, Thomas Koop, Ottmar Möhler, Dennis Niedermeier, Björn Nillius, Michael Rösch, Diana Rose, C Schmidt, Isabelle Steinke, and Frank Stratmann. Intercomparing different devices for the investigation of ice nucleating particles using Snomax as test substance. *Atmospheric Chemistry and Physics*, 15(3):1463–1485, feb 2015.

[16] M A Turner, F Arellano, and L M Kozloff. Three separate classes of bacterial ice nucleation structures. *J. Bacteriol.*, 172(5):2521–2526, 1990.

6 | Visualization of economic agent-based simulations: introducing the FLAViz toolbox

Sander van der Hoog[1], Philipp Cimiano[2]

1 – Faculty of Business Administration and Economics, Bielefeld University
2 – Semantic Computing Group, Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University

Abstract

We describe the result of a collaboration between the Economic Theory and Computational Economics (ETACE) group at Bielefeld University and the Conquaire project. The Economic Theory and Computational Economics (ETACE) group, with a project led by Prof. van der Hoog, applies agent-based modeling approaches to study dynamic equilibrium models resulting from the interaction of heterogeneous rational agents. This allows insights into the application of different industrial policy measures in different regions, the existence of varying spatial frictions on goods and labour markets, the spatial dynamics of industrial activity, technical change and growth, the micro- and macro-prudential regulations and their effects on micro-fragility and macro-financial stability, as well as financialisation of the real sector and the need for productive credit for economic development. In this paper, we describe the implementation of the FLAViz library that realizes a data analytic processing pipeline supporting the computational analysis and visualization of simulation data generated in the FLAME environment. This library is a key step towards ensuring computational reproducibility of the analyses of the available simulation data.

Keywords

Computational Economics, Python, Pandas, simulation data, High-Performance Computing.

6.1 Introduction

The research group on Economic Theory and Computational Economics (ETACE) is concerned with the analysis of different aspects of economic dynamics and strategic interaction. It employs and extends both analytical methods, in particular dynamic optimization and dynamic game theory, and computational approaches, where the latter include numerical methods for the solution of (dynamic) equilibrium models as well as agent-based simulations.

Research at ETACE is based on the conviction that a thorough examination of (dynamic) economic phenomena should be based on a combination of (i) dynamic equilibrium analysis, providing benchmark results under full rationality (and foresight) of decision makers, and (ii) the explicit consideration of the economic dynamics unfolding under the interaction of rationally bounded heterogeneous agents. The aim of the work undertaken by the ETACE group is to extend the toolbox of economists and policy makers, and to apply these tools to relevant research questions, mainly in the areas of Industrial Economics, Labour Economics and Macroeconomic Dynamics.

Ongoing research at ETACE can be broadly categorized in the following research topics:

- Agent-based Modelling for Economic Policy Analysis
- Economics of Innovation and Industrial Dynamics
- Network Formation and Spatial Dynamics
- Labour Economics and Search Theory

Within the Conquaire project, we have addressed work in the first topic area, that is *Agent-based Modelling for Economic Policy Analysis*. In particular, we have identified the Eurace@Unibi Model, a specific agent-based simulation model, as a case study to test the notions of *analytical reproducibility* and *continuous integration of research data*, which are two key aspects of data management for the Conquaire project.

In recent years, it has been widely acknowledged by economic scholars that the explanatory power of standard representative agent models is in many cases limited. This has led to a surging interest in the empirical exploration of bounded rationality in economic decision making, mainly by means of laboratory experiments and attempts to incorporate heterogeneity in endowments or behavior into economic models. A particularly natural and promising approach to account for economic phenomena that result from the (bounded) rational interaction of heterogeneous economic agents is the use of agent-based computer simulation models. Phenomena of such types are abundant (the avalanche-like dynamics in the network of connected commercial banks inducing the current economic crisis is just one prominent example in that respect), and a large amount of insightful agent-based research has addressed a wide range of relevant economic issues (see e.g. the Handbook of Computational Economics Volume II edited by Tesfatsion and Judd [1] for an overview, and the more recent Handbook of Computational Economics Volume IV edited by Hommes and LeBaron [2] for applications).

A main research topic at ETACE is the development of micro-founded macroeconomic heterogeneous agent-based models that can be used as an integrated framework for policy analysis in different economic policy areas. Based on work carried out in the EU-funded Eurace Project, the Eurace@Unibi model has been developed and used as a tool for the analysis of various economic policy questions related to issues of technological change and economic growth, labor market policies, social cohesion and convergence, and to study banking and credit market regulations. See [3] and [4] for a more detailed description of the model. The Eurace@Unibi model is among the most sophisticated and well-documented models in this domain of economic research. It has strong empirical micro-foundations and reproduces a large set of empirical stylized facts. Ongoing work focuses on the analysis of policy effects considering spatial factors and knowledge and information flows. In particular, the goal of the model is to allow to study the effects of:

- the application of different industrial policy measures in different regions,
- the existence of varying spatial frictions on goods and labour markets,
- the spatial dynamics of industrial activity, technical change and growth,
- microprudential and macroprudential regulations and their effects on micro-fragility and macro-financial stability, and the
- financialization of the real sector and the need for productive credit for economic development.

The Eurace@Unibi model is adapted and extended on a regular basis to address concrete research questions in economic policy. Finally, members of the ETACE group develop and apply statistical methods and concepts to systematically and rigorously analyse computational policy experiments using agent-based simulation models. The data being generated by such simulation models can be quite complex, not just in terms of data volumes but also in terms of its dimensions, heterogeneity, and variety. This is especially true when large-scale agent-based models with large agent populations are simulated. To analyse such high-dimensional data, new data visualization techniques must be developed, and this was one of the main tasks to be accomplished by the ETACE group in the context of the Conquaire project. Since the Eurace@Unibi model, which was selected as our use-case for the Conquaire project, has been implemented in the simulation environment FLAME, we give a brief description of this simulation platform below. In the following subsection 6.2, we describe how the

FLAME environment is used to generate the simulation executable and describe a library called FLAViz that has been developed in cooperation with Conquaire and supports the analysis of simulation data generated by a FLAME-generated model.

6.2 Methods

In this section, we describe the FLAME environment.

6.2.1 The FLAME Environment

The Flexible Large-scale Agent Modelling Environment (FLAME) is a generic agent-based modelling platform, which can be used to generate agent-based models in a wide range of applications, such as biology, crowd simulations, and economic analyses (see the FLAME website for examples and code).[1] The software components *XParser* and *Libmboard* can be downloaded from the GitHub repository of FLAME-HPC.[2]

In principle, FLAME is not a simulator, but a *simulator generator* since it creates a simulation executable that can be run on any hardware platform from laptops or servers, to HPC clusters. Currently, there exist different versions of FLAME for use with CPUs or GPUs, and efforts are underway to create a single, uniform environment that addresses all hardware architectures. The CPU version is called FLAME-HPC and is currently the most mature version (see [5, 6, 7, 8, 9, 10] for a more detailed description of FLAME).

Several features make FLAME particularly appealing as a framework to develop and analyse large-scale agent-based models since the framework has been specifically designed for use on high-performance computing clusters. It provides a very transparent and clean way to model information flows between agents using messages, both internal inside the conceptual model and outside of it through the use of a Message Passing Interface (MPI), provided by the Libmboard library. The only means to communicate private data between agents is through the exchange of messages, where the data an agent can transmit consists of a list of values of its own state variables (e.g. wealth, income, skills, profits, expectations about certain variables). Messages are added to a centralized message board and the sender determines which agents can read the message. Agents check the message boards in every iteration in order to collect all the information they are supposed to receive. An agent can use the collected information as input to its decision rules or as the basis for updating some of its own state variables.

Since high-performance computing clusters are involved, and computational resources on such clusters are still a scarce resource, the data generation and

[1]See the FLAME website `<http://www.flame.ac.uk>`.

[2]See the GitHub repository `<https://github.com/FLAME-HPC>`.

data analysis stages are a multi-stage process in which considerations of computational time and data storage play an important role. These two steps are separated in time, with the data first being generated and stored to disk, and afterwards the data is again loaded for analysis.

At the simulation design stage (before simulations are actually run), the model analyst can select to output either a complete snapshot of all variables of all agents (this is very data intensive), or select a subset of agents for which all variables will be stored. In addition, it is also possible to select a certain frequency at which the data is output, say every n iterations, or to select only a subset of variables (a much less data intensive mode of simulation).

6.2.2 Simulation Data

FLAME uses the XML format for data input and output files. In order to design a simulation model in FLAME, three types of XML files are typically required:

- **Model XML files**: This file follows a DTD (see FLAME User Manual, [5, pp.43-44]). It specifies the model's data structures and variable types, with XML tags for the environment, models, agents, messages, ADTs, and time units. The environment-tag contains static constants (model parameters) and file names for the C function files (user-created). The xagent-tag contains memory variables and functions. Messages and ADTs contain attributes, which are the variables contained in these data containers.

- **Data input XML files**: This file is an input argument to the simulator executable (see FLAME User Manual, [5, pp.30-31]). It contains all initial values for the model constants and agent variables. Usually the input file is called *0.xml*, and the default file size is now about 25 MB for our standard economic model.

- **Data output XML files**: These are the output files generated by the simulator executable (which itself is generated by FLAME by compiling the user-created and template C code). This type of file only contains the values for all the agents' variables. The environment constants have no output (except when the output file is a snapshot, see below), since the constants are static and are already contained in the input file.

In order to understand the structure and data content of the output XML files, a brief discussion about the notion of *agents* might be helpful. In research at the ETACE group, we deal with different economic *agent types*, such as *Eurostat, Bank, Firm, Household, Central bank*, etc. Each agent type has a different set of variables, since this depends on what activities the agent performs in the model. For example, the agent type *Bank* might contain variables such as cash, total credit, deposits, mean interest rate, etc. Another agent of type *Eurostat* might contain variables like: *unemployment rate, total debt, monthly output, average wage*, etc.

Also, each agent type is an archetype, and many instances of each agent type may actually exist in the simulation. In this sense, the agent types are similar to an object class, and the individual agents are similar to object instances. Depending on the particular type of economic analysis, we have different requirements for the simulation output. For example, a particular simulation might contain only the agent type *Eurostat*, while for another analysis we might need more than one agent type, for example all *Eurostat, Firm and Bank* agents. Therefore, the agent types and their variable lists can be filtered before they are output to disk, saving on simulation time and storage requirements. This is one reason why the output XML files may vary in size. Some common file sizes (per iteration) are: 105 bytes (store only Eurostat, 1 variable), 2 MB (store multiple agent types, multiple instances of each type, and many variables per agent instance), 25 MB (store a population snapshot, containing all agents, and all variables per agent). If a certain analysis requires millions of runs for millions of iterations (for molecular dynamics for instance), it makes sense to filter out some of the data before it is output to disk.

The population snapshot file of 25 MB also contains the model constants/parameters in addition to the agent variables. These static constants are usually not part of the output file, as this would be redundant since they are already contained in the input XML file (0.xml). As the purpose of the snapshot file is to be used again as an input file to the simulator, the model constants must also be contained in this file.

The output XML files are named with the iteration numbers. Basically, a file named *1.xml* contains all the values at the end of the first iteration; similarly the file *2.xml* contains all the values at the end of the second iteration, and so on.

Visualizing Simulation Data

In order to generate the simulation data we have adopted the following ontology:

- **Sets**: a set reflects a model parameter setting. Each set differs from another set only in the parameter setting of the model.
- **Runs**: a run is a replication for a fixed parameter setting. Each run differs from other runs only by the random seed. The other initial conditions are kept exactly the same across runs.

Thus, parameter variations are captured in *settings* or *sets*. Each set reflects a different parametrization of the simulation model. In case the model contains random variables and stochasticity, the statistical properties of the model can be explored using different random seeds and a Random Number Generator (RNG). By default, we use the RNG from the open source *GNU Statistical Library*

(GSL), which is based on a Mersenne Twister (`mt19937`). For each data set, multiple runs are performed using different random seeds, producing different simulation output for each run. These runs can be called Monte Carlo replication runs since the random seeds are themselves varied in a random fashion. The seed is set randomly based on the system time at simulation launch time, and stored for later replication of the data, if required.

6.3 Analytical Reproducibility

In this section, we describe the implementation of the Flexible Large-scale Agent Visualization Library (FLAViz), which is a software library specifically designed for the analysis and visualization of data generated by Agent-Based Models (ABMs). Agent-based simulation models typically generate data across multiple dimensions, e.g. parameter sets, Monte Carlo replication runs, different agent types, multiple agent instances per type, many variables per agent, and time periods (iterations). This implies the data is structured as time series panel data sets. FLAViz has been developed in cooperation with the Conquaire project and has been specifically designed for FLAME-generated data, but in principle data from any ABM can be used, as long as the data adheres to the file specifications. FLAViz builds on the Python pandas library to deal with such high-dimensional time series panel data sets. The data is stored as structured data using multiple hierarchical levels in the HDF5 file format. This allows for proper data aggregation, filtering, selection, slicing, transformation, and visualization. The toolbox is setup in a modular way as a flexible set of tools that can be integrated into an automated work-flow for analysing the time series data generated by any computational model. The software code for the visualization library FLAViz is open-source and available for download from the GitHub repository.[3] The installation instructions and dependencies are documented in the readme file of the repository, as well as tutorials and example data.

6.3.1 Data Analysis Pipeline

FLAViz is an addition to the FLAME set of tools used for the simulation and analysis of large-scale agent-based models. FLAME natively outputs data in XML format. In FLAViz this gets processed using Python scripts and transformed into HDF5 files for final storage. Building on the pandas and matplotlib libraries, various plots can be specified, e.g., time series, box plots, scatter plots, histograms, and delay plots.

FLAViz version 0.1.0 (beta) is written in Python (ver- 3.6) and other package dependencies include:

- Pandas (ver-0.21.0)

[3] `<https://github.com/svdhoog/FLAViz>`

- YAML files for easy configuration management
- Matplotlib for data visualization
- HDF5, and
- PyTables

FLAViz uses two important inbuilt features of the pandas library, viz.:

- **Hierarchical indexing**: this allows a high dimensional data frame (the ndarray format)
- **Bygroup**: this allows to re-order the hierarchical index, to reshape the data dimensions

At the outset, the original simulation data are stored in XML files and are then converted to a more data-processing friendly format, viz. the HDF5 format. This is needed because the XML files that FLAME simulations generate are a fully tagged data format and is therefore very verbose. For large scale simulations this is prohibitive in terms of the sheer size of the data volumes generated. The storage and parsing of large data volumes generate a complex data structure. In order to reduce this storage footprint, yet retain the structured data format, the HDF5 standard was chosen for its hierarchical data storage structure. The Pandas library can easily read large *.h5 files and store the data internally into one of its native data formats (either *pandas.dataframe* or *numpy.ndarray*).

The data hierarchy is as follows:

1. **Agent types**: $a = 1, ..., A$ - Classes, groups of agent sub-populations
2. **Sets**: $s = 1, ..., S$ - Parameter settings (model calibrations)
3. **Runs**: $r = 1, ..., R$ - Monte Carlo replication runs (random seeds)
4. **Iterations**: $t = 1, .., T$ - Time periods
5. **Agents**: $i = 1, ..., n\ a$ - Individual agents (per type)
6. **Variables**: $j = 1, ..., m$ - Scalars, Arrays, Composites

Due to this large data heterogeneity, the file sizes may vary across simulations with the same model, even when using exactly the same input file, due to stochasticity. The data for each *agent-type* is stored in a single HDF5 file without any file-size limitations. The data is heterogeneous across several dimensions:

- **agent types**: there can be many different agent types (e.g., household agents, firm agents, bank agents, etc.)

- **agent instances**: there can be a different number of agent instances per agent type
- **agent memory variables**: there can be a different number of memory variables per agent type (but all agents of the same type have the same set of memory variables, specified a priori, in the *model.xml* file that fully specifies the model's structure)

HDF5 File Format

HDF5 has a simplified file structure that includes only two major types of objects:

1. **Datasets**: which are multidimensional arrays of a homogeneous type; and
2. **Groups**: which are container structures which can hold datasets and other groups.

The main restrictions of the HDF5 file standard are:

1. the HDF5 file format requires that the atomic data set at the lowest hierarchical level is a homogeneous data format (no ragged edges). This means that the choice of the 6 dimensions (Sets, Runs, Iterations, Agent types, Agent instances per type and Variables) requires us to choose those dimensions that remain invariant across all model simulations as the ones contained in this homogeneous data structure. These dimensions are: *Agent instances*, *Iterations* and *Variables*. These dimensions are invariant because we simulate the same model many times, and we do not change the model structure across simulations. Therefore the number of variables per agent remains the same, the number of agent instances per agent type is constant, and the total number of iterations also remains constant across simulation runs. Another reason for choosing those 3 dimensions is that the sets and runs form a unit of analysis, so it makes sense to choose those for the higher level in the hierarchy. Also, the simulation output for the sets and runs can be generated on a cluster in a massively distributed fashion, by distributing the compute load across many nodes. Logically, this implies storing the output in separate files according to the set/run combinations first, and only at the very end combining all these files according to the agent types.
2. The 3D Panel format in Python pandas has 3 axes (item, major and minor) and is specified as *row-major*. This means that the data structure requires the largest dimension to be on the major axis. In our case, the largest dimension is the number of iterations, typically 1000 or higher. The other dimensions are the number of agent instances (on the order of 100), and the number of variables (also on the order of 100).

Given the above constraints, we specify the 3D Panel data structure as follows:

1. **item axis**: agent instances
2. **major axis (table rows)**: iterations
3. **minor axis (table columns)**: variables

To deal with the remaining 3 dimensions (agent types, sets and runs), we proceed as follows. We generate a separate HDF5 file per agent type, using the naming convention `AgentType.h5`. To account for the two remaining data dimensions of sets and runs, we specify the data groups inside the HDF5 file using set/run combinations as follows: `set_s_run_r` ($s = 1, ..., S$ and $r = 1, ..., R$).

Summarizing, the simulation data is stored in a **HDF5** container file (*.h5, *.hdf5) using a hierarchical data format. Currently, these **HDF5** files are structured as follows:

- Each agent type is contained in a separate HDF5 file, with the same name as the agent type (e.g., *Firm.h5*, *Bank.h5*, etc.).
- Inside each HDF5 file there is a **Group** (similar to a folder structure) for each combination of *set* and *run*, using the naming convention `set_s_run_r`.
- Inside each **Group** there is a **Dataset** which contains a *pandas Panel*, which is a datastructure that consists of *items*, *major* and *minor* axes.
- the **Python pandas Panel** is written to the HDF5 file with the *PyTables* module of Python, which uses a write-once policy (no appending).

The HDF5 file structure described above can be created from SQLite database files that contain the results from set/run combinations by using the data processing scripts that are included in the FLAViz package. Alternatively, the HDF5 file could be created from the XML files directly, but a big disadvantage of this method is that the entire collection of XML files has to be available on disk in uncompressed form (very bulky), which could be prohibitive for large-scale applications. It is also not very resource-friendly, due to its lack in parallelism. Another option would be to stream the data into the final database file as it becomes available from the simulations. Unfortunately, however, streaming the data into an HDF5 file is not possible, due to the write-once feature of the *PyTables* module that we have chosen to adopt in the library to write to the HDF5 file. The reason for this choice is that appending data to an HDF5 file would require a different write method using the *h5py* module, which is less performant than doing it write-once.

6.3.2 Plotting with FLAViz

To adhere to the general principle that all results of a published paper should be computationally reproducible given the data from computational experiments, we should be able to reproduce the plots using various permutations and combinations of the data. FLAViz uses three configuration files, through which the necessary conditions can be set. The configuration files follow the hierarchical `yaml` format for clarity and functionality with specific indentation for the input to be interpreted correctly. For the general plot settings, the yaml file `config.yaml` contains settings for selecting the desired **sets** and **runs**, or to specify ranges for the **iterations** and **variables** along the **major** and **minor** axes, respectively. It is also possible to perform data transformations of agent variables, and to select data based on data slicing. For example, select all data at iteration $t = x$, or select all data for agent $ID = i$. Data filtering can also be performed, in which case the data is filtered based on agent conditions or variable conditions. For example, filter the selected data on the condition that the agent variable X has value v. For selecting the plotting styles, the yaml file `plot_config.yaml` contains settings to select what kind of features the plot should contain. Everything related to axes, legends, colours, etc, can be set in this file, which follows the basic features of `matplotlib`, which is the standard plotting library used by Python pandas. Currently, if the user specifies multiple plots, these are processed one by one. To speed up this process and parallelize the plotting routine, each plot could be run as a separate sub-process that retrieves data from the main data set once it has been read into main memory. This is left for future development of the FLAViz library.

Example config files As an example, the plot in Fig. 6.1 shows a visualization of data for the agent type "Firm", the variable "price", and is based on data for 4 sets (selected sets: 10, 13, 16, 17). Each set consists of 20 runs. The plotting style is specified as using a time series multiple-batch plot, showing the 20^{th} and 80^{th} percentiles. The construction and generation of this plot is specified in the following settings in the configuration files.

config.yaml:

```
plot1:
    timeseries:
        agent: Firm
        analysis: multiple_batch
        variables:
            var1: [price]
        set: [10,13,16,17]
        run: [range,[1,20]]
        major: [range,[6020,12500,20]]
        minor: [range,[1,80]]
        summary: custom_quantile
        quantile_values:
            lower_percentile : 0.20
            upper_percentile : 0.80
```

plot_config.yaml:

```
plot1:
    number_plots: one
    plot_legend: yes
    legend_location: best
    xaxis_label: Time
    yaxis_label: price
    linestyle: solid
    marker: None
    fill_between: yes
    fillcolor: darkgreen
```

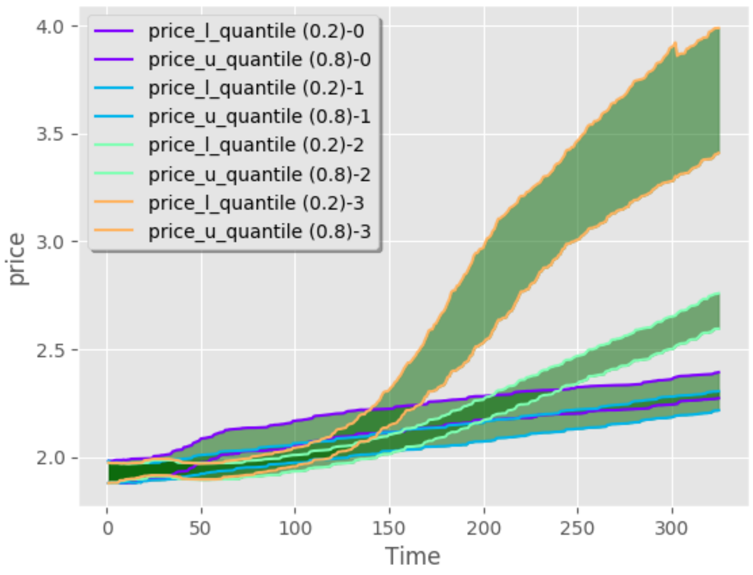


Figure 6.1: Plotting the price time series for data covering 4 sets, each consisting of 20 runs of 6.500 iterations, and for 80 Firm agents.

6.4 Summary and limitations

FLAME simulations with large-scale economic simulation models require high-performance super-computing (HPC) facilities and generate large datasets that typically represent a bottleneck from the perspective of both computational resources and storage requirements.

BigData storage using the HDF5 format (a hierarchical filesystem-like data format) works well for economic data that consists predominately of time series data (i.e., numerical, not text data). If needed, more complex storage APIs representing images and tables can be built using datasets, groups, attributes, types, dataspaces and property lists. Because the bulk of the data is transformed into straightforward arrays (the table objects) for processing, the data can be accessed in a much faster way than with more traditional row-based processing in an SQL database.

The time performance of the FLAViz Library could probably be reconfigured/refactored to work in a more distributed fashion by optimizing the order in which the data is post-processed. We can probably speed-up several operations that now are taking place sequentially. This is a matter of figuring out what the main loops on the various dimensions of the data (agents, variables, iterations, etc.) are, and then determine the optimal order in which these loops should be executed. The items in the loops can then be executed in different threads, and it should be investigated whether there are any information dependencies that need to be resolved between those threads.

Also the storage performance could be optimized. Not all data need to remain in memory at all times. Currently, we first read-in all the data into main memory, then filter it based on conditions, and then process it further.

In simulation science, we deal with very complex data objects with a wide variety of metadata that require a portable file format without any limits on the number or size of data objects in the collection. The HDF5 format is a versatile data model that makes it easier to manage extremely large and complex data collections with time and storage space optimizations. It also runs on a range of computational platforms. Some advantages of using the HDF5 format are:

- **HDF5 is a Self Describing Format**: Each file, group and dataset can have associated metadata that describes exactly what the data is, viz., data types, description, documentation of data ontologies, information about how the data in the dataset were collected, etc.

- **Compressed & Efficient subsetting**: The HDF5 format is a compressed format and data size optimization makes the overall file size smaller. The data slicing feature allows subsets of a dataset to be extracted for processing in order to avoid storing the whole dataset in main memory.

- **Heterogeneous Data Storage**: HDF5 files can store multiple types of data within the same file as sets of datasets containing heterogeneous data types (e.g., both text and numeric data in one dataset)

- **Open Format**: HDF5 has technical support in many programming languages and tools, like ’R’, ’Python’ and ’Julia’ due to its open format.

6.5 Conclusion

This paper has described a case study in the area of computational economics on the computational reproducibility of simulation results. In contrast to other chapters, we have not aimed at reproducing a particular result published by the ETACE group. Instead, Conquaire has cooperated with the ETACE group to implement a generic visualization library called FLAViz, to support the exploration and visualization of simulation data. The pipeline implemented in FLAViz makes use of the HDF5 format, which has turned out to be a very flexible and versatile data format.

FLAViz supports the analytical reproducibility of research data in two ways, both ex-ante and ex-post publication. Firstly, if researchers store their simulation data on an ongoing basis during a research project, and FLAViz configuration files are also available, then an automatic plot generation tool can be used in the sense of Continuous Integration of research data. This helps a lot in increasing the trustworthiness and credibility in the results, as well as giving us the ability to track how the results are changing over time as the research project progresses. Secondly, if pre-generated simulation data is available from a published paper from the original authors, then the FLAViz toolbox could be directly applied to this dataset to reproduce the plots of the published paper. These can then be used to check the validity of the claims made by the original authors in their paper. In these two important ways, toolboxes such as FLAViz can be regarded as helping us to ensure the analytical reproducibility of research data.

Acknowledgements

We would like to thank Krishna Devkota for implementing the FLAViz library and Fabian Hermann for documentation and bug fixing.

References

[1] Leigh Tesfatsion and Kenneth Judd, editors. *Handbook on Agent-Based Computational Economics*, volume 2. North-Holland: Elsevier, Amsterdam, 2006.

[2] Cars H. Hommes and Blake LeBaron, editors. *Handbook on Agent-Based Computational Economics*, volume 4. North-Holland: Elsevier, Amsterdam, 2018.

[3] Herbert Dawid, Simon Gemkow, Philipp Harting, Sander van der Hoog, and Michael Neugart. Agent-Based Macroeconomic Modeling and Policy Analysis: The Eurace@Unibi Model. In S-H Chen, Kaboudan M., and Y.-R. Du, editors, *The Oxford Handbook of Computational Economics and Finance*, chapter 17, pages 490–519. Oxford University Press, 2018.

[4] Herbert Dawid, Philipp Harting, Sander van der Hoog, and Michael Neugart. A Heterogeneous Agent Macroeconomic Model for Policy Evaluation: Improving Transparency and Reproducibility. *Journal of Evolutionary Economics*, 29:467–538, 2019.

[5] Mariam Kiran. *FLAME Flexible Large-sale Agent-based Modelling Environment User Manual.* University of Sheffield, 2010.

[6] Simon Coakley and Mariam Kiran. *FLAME User Manual.* University of Sheffield and Rutherford Appleton Laboratories, STFC, 2012.

[7] Simon Coakley, Marian Gheorghe, Mike Holcombe, Shawn-Lee Chin, David Worth, and Christopher Greenough. Exploitation of high performance computing in the FLAME agent-based simulation framework. In *Proceedings of the 14th International Conference on High Performance Computing and Communications*, pages 538–545, 2012.

[8] Paul Richmond. FLAME GPU Technical Report and User Guide. Technical Report CS-11-03, 2011.

[9] Simon Coakley, Paul Richmond, Marian Gheorghe, Shawn-Lee Chin, David Worth, Mike Holcombe, and Christopher Greenough. Large-Scale Simulations with FLAME. In Joanna Kołodziej, Luís Correia, and José Manuel Molina, editors, *Intelligent Agents in Data-intensive Computing*, Studies in Big Data Series, pages 123–142, 2016.

[10] Mariam Kiran. *X-Machines for Agent-Based Modeling: FLAME Perspectives.* Computer and Information Science Series. Chapman & Hall/CRC Press, Boca Raton, Fla., 2017.

7 | Reproducing experiments on early verb understanding in infants

Vidya Ayer[1], Christian Witte[1], Philipp Cimiano[1], Katharina J. Rohlfing[2], Iris Nomikou[3]

1 – Semantic Computing Group Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University
2 – Psycholinguistics, Faculty of Arts and Technology, Paderborn University
3 – Department of Psychology, University of Portsmouth

Abstract

In this chapter, we describe an effort to reproduce the main result of the paper *"Evidence for early comprehension of action verbs"* by Nomikou et al. [1]. The study aimed at investigating the ability of 9-and-10-month old infants in understanding verbs. The study in question followed a so called *preferential looking paradigm* consisting in investigating the ability to understand the meaning of words by testing whether infants look longer at a related stimulus compared to an unrelated stimulus. As a method to track looking time proportions to the target stimulus, an eye tracker was used. Data were collected from 9- to 10-month old infants who were presented with paired-picture trials while listening to corresponding verbs. The infants saw two images on the screen side by side, each one from a different context category (CARE or PLAY). One of the pictures was related to the verb in question, while the other image was a confounder. The percentage of time that infants looked at the matching picture, before and after having heard the verb, was recorded across participants, computing a difference score. In case the difference was positive, this was taken as evidence of understanding the meaning of the verbs. The study could only find a positive difference for 10-month olds but not for 9-month olds, showing that the ability to understand the verb in question emerges between 9 and 10 months. In close interaction with the authors of the original paper we rewrote the analysis scripts which were used by the authors to refine the results during a second iteration of reviewing in response to requests by reviewers. Overall, we could reproduce the central results of the study. This case represents a case of *full analytical reproducibility.*

The data and scripts for the paper described above can be found at `https://gitlab.ub.uni-bielefeld.de/conquaire/psycholinguistics`.

Keywords

Psycholinguistics, Language learning, Verb Understanding, Infants, Eye Tracking

7.1 Introduction

The Psycholinguistics research group at Paderborn University is concerned with investigating language development in young children. Its main research interest is how children acquire the meaning of words to reveal links between language and cognitive development and to analyze early meanings as building blocks for conceptual and linguistic thinking.

There is a debate on the question whether nouns are acquired before verbs. In contrast to nouns, which can be easily singled out by holding an object or pointing to it, verbs are relational since they combine agents and their actions with some objects. In consequence, verbs have a more complex semantic structure. However, it is also possible that early use of nouns is cumulative as it also binds together situational elements. For example, an infant might say *ball* but relate this noun to the action of rolling [2].

In the study reproduced as part of this work titled *'Evidence for early comprehension of action verbs'* [1], the research group studied 9- and 10-month-old infants' understanding of verbs using a technique similar to the one used by Bergelson et al. [3], except that verbs were used in place of nouns. Following the conceptual development approach proposed by Mandler [4], the hypothesis was that infants must conceptualize situated actions early in their development to get concepts about objects off the ground. Early concepts, thus, will entail the role of objects, i.e., what the objects do and what is done to them [4], which provides a solid basis for the acquisition of verbs. Thus, the hypothesis was that children at a younger age, as found so far, will understand verbs that are drawn from their everyday life contexts. Instead of using dynamic pictures that refer to verbs, static object pairs were used and parents were asked to utter the relevant verbs.

In this work, we aim at reproducing the main result of the paper mentioned above, i.e. that demonstrated early verb understanding by showing that infants tend to look longer at the correct target picture once their parent uttered the corresponding verb. The study found a developmental difference between 9- and 10-month olds though: 9-month-olds were not able to reliably demonstrate verb understanding. With respect to early semantic development, as visualized by the target looking times, the data suggests that on hearing a verb, the infants can associate it to object stimuli related to the verb. This is in line with the researchers argument claiming that action concepts can be evoked in object perception. Furthermore, the results complement research proposing that children learn language by building relations and drawing from rich visual concepts [5].

7.2 Methods

Here, we describe the methods used for the experimental settings in the original experiment.

7.2.1 Experimental settings and data acquisition pipeline

The study in question followed a so called *preferential looking paradigm* consisting in investigating the ability to understand the meaning of words by testing whether infants look longer at a related stimulus compared to an unrelated stimulus. As method to measure target looking times, an eye tracker was used. Data were collected from 9- to 10-month old infants who were presented with paired-picture trials. The infants saw two images on the screen side by side, each one from a different context category (CARE or PLAY). One of the pictures was related to the verb in question, while the other image was a confounder. These images were shown for a total of 9.5 seconds. Within the first 3s of each trial, parents heard a beep before they heard a sentence that they were asked to reproduce. Then, a second beep prompting them to begin repeating the sentence. While the parent was saying the target verb, the experimenter pressed a key on a wireless keyboard to mark the precise moment at which the verb was perceivable to the infant. This mark was logged into the data. An attention getter, i.e. a 3s clip featuring colorful animated shapes accompanied by different sounds, appeared after each trial. The experiment lasted 5 minutes. The entire visit of the infants to the lab lasted 45 minutes.

Because of individual differences in the production of the target phrase by the parent, the post-target analysis window extended from 367 to 4.500 ms after the onset of the spoken target word. To calculate the onset of the target word, the recorded time-stamp of the keyboard key press was used. A Python script was used to split the looking times into two periods: before and after the uttered verb. The dependent variable, namely, word comprehension, was thus operationalized by a difference between the proportion of target looking upon hearing the target word (367 to 4.500 ms post keyboard keypress) minus the proportion of target looking before hearing the word (from when pictures were displayed until just before the keyboard keypress). This way, a difference value was obtained that could be positive or negative. If the value was positive, it indicated increased looking at the target object by the infants after hearing the verb, thus demonstrating their understanding of the target word.

7.2.2 Methods applied to analyze the data

The raw eye-tracking data were filtered using python scripts according to predefined areas of interest (AOIs). Then total gaze durations at the AOIs were calculated and subsequently the script took into account a specific timestamp generated by a key press of the keyboard and calculated the gaze durations before and after the keypress as well as the proportions of gaze at the target or distractor AOIs before and/or after the keypress. These calculations were formatted in a table and used for further calculations. These included before-after difference scores for each of the two presented instances of each pair of stimuli, with the two difference scores being subsequently averaged. These difference scores were then used in a series of statistical tests: t-tests, ANOVAs and binomial tests. For details, the reader is referred to the original publication [1].

In a subsequent review round of the submitted manuscript, various versions of the initial script were produced in collaboration with the Conquaire project to repeat the analysis using a fixed time window for the inclusion/exclusion of data points. This was requested by the paper reviewers. To address this comment, three new versions of the scripts were created with varying window durations, the changes incurred were assessed by comparing the results of a sample of data files and the usage of the script with a 4500ms time-window was selected to re-run the analysis and all the statistical tests.

During the creation and implementation of the scripts, both initially and in the second round of analysis, there was intensive collaboration between members from the psycholinguistics group and the Conquaire team. This was necessary to check for errors in the scripts. For this, random manual calculations were performed on the raw data and then compared with the results produced by the scripts to test for accuracy. In some cases, multiple iterations were needed until the systematicity in the discrepancy between script and manually calculated results was discovered and corrected.

7.2.3 Main Results

Using the process detailed above, the scripts produced tables of variables ready for statistical analysis. A mixed, between, and within-subjects ANOVA was used with AGE (9 months vs. 10 months old) as the between-subjects variable, and TIME (before vs. after the word was spoken) as the within-subjects variable. There was a significant AGE x TIME interaction effect $F(1, 46) = 5.687, p < .021, \eta = .107$. Since an independent-samples t-test indicated significant differences between the 9 and 10 months olds, the data were treated in separate groups. Additionally, a linear regression was calculated with the increase in looking times at the target as the dependent variable and infants' age in days as an independent variable. The regression model did not attain significance, suggesting that the change in performance was not linear, $F(1, 46) = 2.23, p = 0.142$.

7.3 Analytical Reproducibility

Computational reproducibility experiments were conducted with the Psycholinguistics research group at Paderborn University at the paper publishing stage to modify the data analysis scripts and produce results, then implement visualizations with Pandas and matplotlib that was later stored in GitLab under continuous integration. To facilitate team-collaboration on porting and refactoring the code, the python scripts and extracted (TSV format) files for data analysis are available at the following Git repository: `https://gitlab.ub.uni-bielefeld.de/conquaire/psycholinguistics`.

Primary Data

The data in the git repository include the images seen by the infants on the screen, the recordings heard by the parents, the eye-tracking data and the 3s attention getter clip featuring colorful animated shapes moving to different sounds that appeared after each trial. Excel sheets with information identifying participants were not uploaded to the GIT repository due to privacy protection issues.

Analysis Data

The python scripts and extracted data (TSV) files for analysis are stored in the **data_output** folder on gitlab. The research data structure (in the TSV and Excel) files are described below: The TSV files are stored in the "data_output" folder within the subdirectory folders, viz. "tables_3500", "tables_4000" and "tables_4500" for the three time windows. For example, to protect the identity and ensure the infant participants' privacy, filenames are anonymized and named as "VP20_output.tsv" etc.. In each file, the various columns such as "Left_before", "Left_After", "Right_Before", "Right_After", "Fixation_Direction", etc.., contain the measurements for each participant (VP). Within the same TSV document, starting from approximately line 26, another header line contains a new set of measurements titled: Before, After, Bef_Aft_Tar, Target, Dis (before), Dis (after), Bef_Aft_Dis, T-D, T-D(B-A).

7.3.1 Data Workflow Lifecycle

The research data workflow lifecycle diagram in Figure 7.1 explains the sequence of the research data processing and tasks for this project. The research project used Free & Open Source Software (FOSS), which increased the prospect of cross-platform availability of processing tools as Python programming language and visualization packages (like Pandas, Matplotlib) are freely available for multiple platforms.

The old data analysis scripts, written in Python version 2.x, were ported to version 3.6 for program maintenance due to end-of-life for Python version

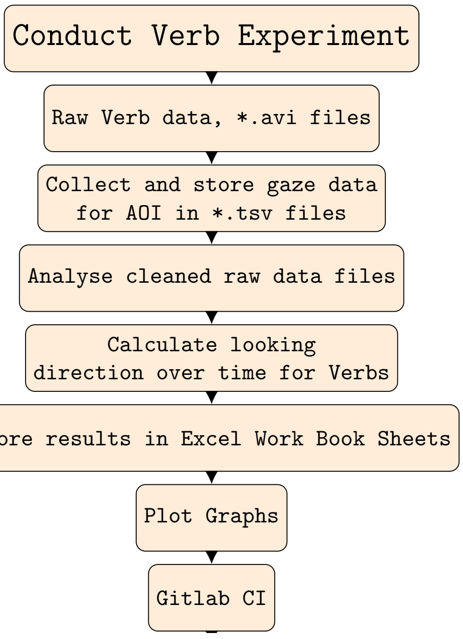

Conduct Verb Experiment
Raw Verb data, *.avi files
Collect and store gaze data for AOI in *.tsv files
Analyse cleaned raw data files
Calculate looking direction over time for Verbs
Store results in Excel Work Book Sheets
Plot Graphs
Gitlab CI

Figure 7.1: Data Workflow

2.x. Refactoring the old scripts from a complex mass of conditional loops, into a simplified modular callable program, was undertaken to ease program maintenance.

The main restructuring changes that were introduced are:

- Most conditional loops were refactored into modular methods. Breaking the code apart into more logical components creates semantic units that are clear and reusable.

- A dict to store the vertical area of interest for each avi file.

- Introduced a class that acts as a wrapper for the dict (which stores the result of one avi file (AOI)) and other methods that can handle the logical componentization.

- A sliding time window to compensate for missing data points - this short time window allows searching for the next fixation data. Three time windows: 3.5ms, 4.0ms and 4.5ms (experimentLength = 3500/4000/4500) were used.

Two Excel sheets stored the analysis results **results_simple_difference_score.xlsx** and **results_simple_target_distractor.xlsx** while the analysis data is stored in tab-separated value (TSV) files.

7.3.2 Reproducibility Results

Once the analysis script was ported to Python-3.6, it was possible to analyze the data and reproduce the results described in the paper as described in section 7.2.3 above. Figure 7.2 shows the percentage of looking times to the matching image for the different verbs, averaged across all subjects including all ages (both 9-month and 10-month olds). The verbs in question were: *'bauen'* (engl. build), *'fahren'* (engl. ride), *'lesen'* (engl. read), *'sitzen'* (engl. sit), *'anziehen'* (engl. dress), *'baden'* (engl. bathe), *essen* (engl. eat), *'schlafen'* (engl. sleep). Figure 7.3 shows the percentage of looking times to the matching image for the different verbs, averaged across all subjects for 9-month old infants only; Figure 7.4 shows the corresponding average looking times for 10-month old infants. Finally, Figure 7.5 shows the percentage of looking times averaged over all verbs and subjects, comparing the average for 9-month old infants vs. 10-month old infants. Within the 9-month-old infant group, on average, the infants spent 51.1% (SD = .056, MIN = 36.7%, MAX = 63%) of their looking time on the target object before the target word was spoken and 49.6% (SD = .063, MIN = 35.4%, MAX = 62%) of their looking time on the target object after the word had been spoken. Within the 10-month-old infant group, on average, these infants spent 43.4% (SD = .095, MIN = 25.9%, MAX = 60.1%) of their looking time on the target object before the target word was spoken and 49.6% (SD = .081, MIN = 31%, MAX = 64.1%) of their looking time on the target object after the word had been spoken. We could thus reproduce the main results of the original paper, showing a positive difference between percentage of looking time to target image after the corresponding verb was spoken minus the proportional looking time to the target before the verb was spoken for 10-month olds. For 9-month olds, this difference was on average negative, showing a lack of verb understanding.

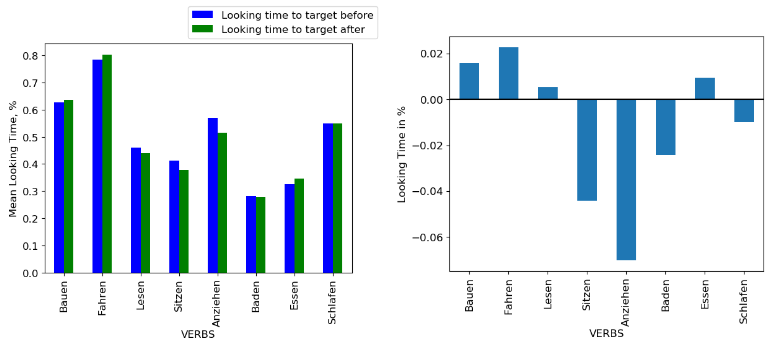


Figure 7.2: Looking times in percentage at matching image before and after utterance for all eight verbs averaged over all subjects (both 9-month and 10-month olds); Right: Difference in looking times for both 9 and 10-month olds

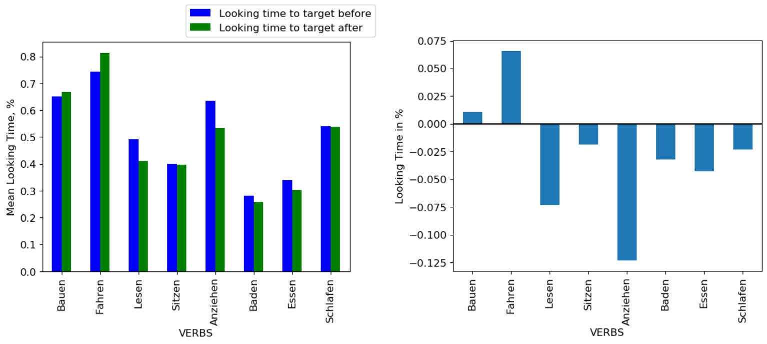


Figure 7.3: Left: Looking times in percentage at matching image before and after utterance for all eight verbs averaged over all subjects (9-month olds); Right: Difference in looking times (After-Before) for 9-month olds

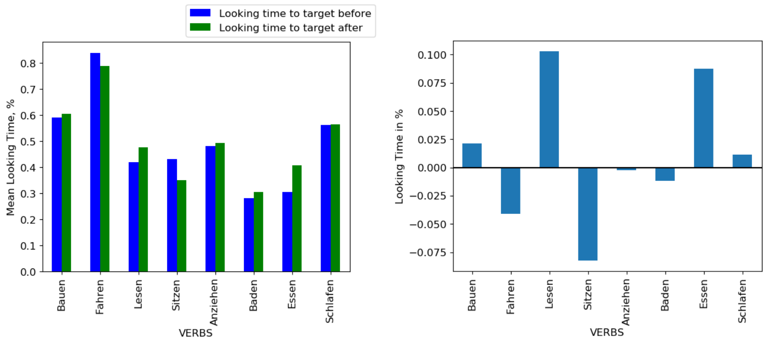


Figure 7.4: Left: Looking times in percentage at matching image before and after utterance for all eight verbs averaged over all subjects (10-month olds); Right: Difference in looking times (After-Before) for 10-month olds

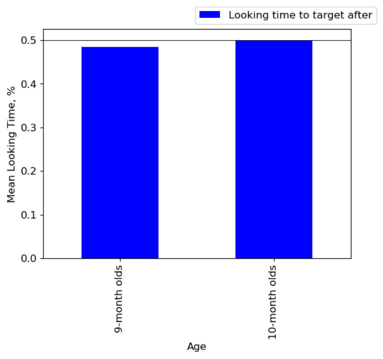


Figure 7.5: Average of percentages of looking times to target averaged over all verbs and subjects comparing 9-month and 10-month infants

7.4 Summary of computational reproduction experiment

In this reproducibility experiment, we were able to reproduce the main result of the study published by Nomikou et al. [1]. This was possible as the data and Pyhton scrips used to analyze the data were available. We engaged in this reproducibility experiment while the paper was in a second round of reviewing and considered the comments of the reviewers to adapt the Python program to allow for different time windows in the analysis. Overall, the results could be reproduced independently. The data and the Python script are available in a git repository for re-use and validation by third parties. This represents a case of *full analytical reproducibility.* Both the derived data capturing the looking times of each subject as well as the script for analysing the data are available in the Git repository, therefore supporting reproduction.

7.5 Conclusion

In this paper, we describe the successful reproduction of the computational analysis phase of a study investigating the early understanding of verbs by 9-month and 10-month-old infants. The reproduced study adopted a preferential looking time paradigm and conducted a so called paired-picture trial in which a verb under investigation was semantically associated to one of two pictures shown, the target picture, and another picture acting as a so called confounder. Using an eye tracker, the difference between proportion of looking times at the matching image before the verb was spoken compared to looking times after the verb was spoken was measured. As a result, the study showed positive differences for 10-month olds, which was operationalized as a measure of early understanding of the verbs. For 9-month olds, in contrast, the study was not able to reliably demonstrate verb understanding. The analytical pipeline that was used to generate results for publication was developed jointly between researchers of the Psycholinguistics group in Paderborn and researchers working in the Conquaire project. The derived data from the experiments (looking times) as well as the Python script are available for further re-use and correspond exactly to the version that was used to generate the published results. In this case, we thus have an example of *full analytical reproducibility*, with the analyses being repeatable by others as a result of the Conquaire project.

Acknowledgments

We would like to acknowledge the support of Lukas Biermann and Fabian Herrmann for helping with implementation of the scripts and data analysis.

References

[1] Iris Nomikou, Katharina J. Rohlfing, Philipp Cimiano, and Jean M. Mandler. Evidence for early comprehension of action verbs. *Language Learning and Development*, pages 64–74, 9 2018.

[2] Katherine Nelson. Concept, word, and sentence: Interrelations in acquisition and development. *Psychological review*, 81(4):267–285, 1974.

[3] Elika Bergelson and Daniel Swingley. At 6–9 months, human infants know the meanings of many common nouns. *Proceedings of the National Academy of Sciences*, 109(9):3253–3258, 2012.

[4] Jean M Mandler. On the spatial foundations of the conceptual system and its enrichment. *Cognitive science*, 36(3):421–451, 2012.

[5] Iris Nomikou, Malte Schilling, Vivien Heller, and Katharina J. Rohlfing. Language-at all times. *Interaction Studies*, 17(1):120–145, 2016.

8 | Reproducing an experiment in automatic disfluency detection

Frank Grimm[1], David Schlangen[2], Julian Hough[2], Philipp Cimiano[1]

1 – Semantic Computing Group, Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University
2 – Dialogue Systems Group, Faculty of Linguistics & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University

Abstract

In this chapter, we describe an effort to reproduce the main results of the published paper *"Joint, Incremental Disfluency Detection and Utterance Segmentation from Speech"* [1], published as part of the proceedings of the "European Chapter of the Association for Computational Linguistics" (EACL) in 2017. The paper focuses on the task of disfluency detection and utterance segmentation and proposes a simple deep learning system that processes dialogue transcriptions and Automatic Speech Recognition (ASR) output. For this purpose, the Dialogue Systems Group (DSG) at Bielefeld University developed a library that relies on a data model for live ASR data that combines timing and textual information. It utilizes a refined text corpus of open data to demonstrate the feasibility of the system for simultaneously detecting disfluencies and segmenting the individual utterances for use in conversational systems and similar speech related tasks. The code and data for this reproducibility experiment are available at `https://gitlab.ub.uni-bielefeld.de/conquaire/deep_disfluency`.

Keywords

Linguistics, Speech Recognition, Python, Machine learning, LSTM, HMM, RNN, NLTK, Theano, Keras

8.1 Introduction

The Dialogue Systems Group at Bielefeld University, located at the Faculty of Linguistics and Literary Studies and the Cluster of Excellence Cognitive Interaction Technology (CITEC), studies artificial conversational systems. The deep learning based disfluency detection system presented in their paper at the International Conference of the European Chapter of the Association for Computational Linguistics (EACL) aims to improve existing solutions in the field of psychiatric health care delivery by introducing the capability to work on live data. This can facilitate the detection of word repairs for human conversational partners and improve turn taking during dialogues. While currently established systems might make use of disfluency markers in text and segment dialogues into individual utterances already, this is often restricted to processing data offline. As such, a new artificial dialogue system could for example be employed during interview sessions in order to ensure that protocols are followed. They can also augment and assist the human interviewer, since artificial conversational agents have been shown to exhibit many different markers that can be interpreted as psychological distress, such as filled pause or speech rates, as well as other temporal, utterance, and turn-related interactional features [2]. In offline processes, analysing transcripts of such sessions today is often costly and frequently relies on a disconnected utterance segmentation process. In the paper *'Joint, Incremental Disfluency Detection and Utterance Segmentation from Speech'* [1], a more cost-effective process is developed, commencing with directly processing speech data and working with online data as it incrementally becomes available during a conversation. The authors evaluate the full process through multiple metrics to capture how each subtask performs as joint or separate models, in online or offline settings. The specific research objective was to investigate how well a joint deep learning model for incremental disfluency detection and utterance segmentation performs on transcripts and ASR output. The former extends existing work on the pre-segmented utterances of the *Switchboard* (SwDA) corpus [1]. The latter uses an external ASR system (IBM Watson) to incrementally process acoustic data and, thus far, could not achieve comparable performance. While recent advances, particularly regarding lowered Word Error Rates (WER), make hypotheses generation through ASR much more reliable, they traditionally lacked similarly fine-grained annotations on different disfluency types as they were applied to transcripts. The paper in question defines the tasks of (incremental) disfluency detection and utterance segmentation, as well as the joint model. The authors discuss reasonable constraints and develop two tagsets *a)* simple and *b)* complex for different complexities of disfluency types. Three explicit research questions are subsequently developed:

- **Q1**: Given the interaction between the two tasks, can a system which performs both jointly help to improve equivalent systems doing the individual tasks?

- **Q2**: Given the incremental availability of word timings from state-of-the-art ASR, to what extent can word timing data increase performance of

[1]`https://github.com/julianhough/swda`

either task?

- **Q3**: To what extent is it possible to achieve a good online accuracy vs. final accuracy trade-off in a live, incremental, system?

In order to address these questions, the authors of the paper present two deep learning architectures for the technical task of incremental decoding for live predictions, namely Elman Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) based networks. Their experimental protocol evaluates both models in separate and joint task settings to assess whether they can exploit common constraints. The system demonstrates competitive results on different subtasks, verifying its suitability and potential to be used within conversational agents in the domain of psychiatric health.

The research experiment utilizes several machine and deep learning techniques to implement an incremental disfluency detection and utterance segmentation pipeline. Since the live audio recordings used for parts of the original experiment were not available to the Conquaire reproducibility experiment due to licensing, we focused on their tagging system in general and worked with the data that were readily available to reevaluate the published models. All considerations within this chapter refer to the *Deep Disfluency* framework as presented in the git commit identified by the hashcode **4c57a19** [2].

8.2 Methods

Speech recognition (SR), also known as *automatic speech recognition* (ASR), *computer speech recognition* or *speech to text* (STT), is a sub-field within computational linguistics that develops methods and technologies to automate the recognition and translation of spoken language into text by machines.

Human speech patterns vary between individuals and contain complex signals such as nuances and diversity in vocal patterns, aspects which adult humans take into account almost automatically. A machine on the other hand has to explicitly mitigate these aspects in order to gain a more thorough understanding of speech signals, even more so in a conversational context.

In order to suitably train a modern, reliable, speech recognition system, many machine learning algorithms and techniques work in tandem. Since fully training a STT system end-to-end would require large amounts of raw and annotated audio data in various environments, the authors rely on a suitable external system to incrementally generate textual input sequences from audio recordings and focus on the specific aspects of disfluent terms as discussed above. Here, we describe the methods used for these experiments in the Deep Disfluency library and subsequently discuss the reproduction of their results.

[2]available at `https://github.com/d<sg-bielefeld/deep_disfluency/` (4c57a19)

Model Training

The speech models are trained on millions of pre-translated words and phrases from corpora against a live ASR system. For incremental ASR, a free trial version of IBM's *Watson*[3] Speech-To-Text (STT) service was used, which according to the authors works well on noisy input data and also retains some useful artifacts such as disfluency markers (e.g. filler terms like 'uh').

The Deep Disfluency system uses the following input features:

- Words in a backwards window from the most recent word (for transcribed and ASR data, the lack of lookahead capabilities simulates the live influx of speech information).
- Durations of words in the current window, either from manually transcribed data or automatically generated by the ASR system.
- Part-Of-Speech (POS) tags for words in current window. These are either extracted from the transcribed corpus or generated through a Conditional Random Field (CRF) based tagger that was optimized on a domain specific training corpus.

The models of the Deep Disfluency system extract these features in two main experimental settings: *a)* on data generated for Switchboard audio recordings through an external ASR system and *b)* on manually transcribed data from the commonly used and well-annotated Switchboard corpus (SWdA).

For regular usage, the models trained on these corpora can be loaded and are subsequently used to apply the full tagging pipeline to arbitrary input sequences. The pipeline, described in more detail below, consists of the extraction of features as listed above, sequence to sequence tagging through one of the deep neural network architectures and consolidating their output with timing information through a Hidden Markov Model (HMM) to produce a final set of tags for each token in the sequence.

Taggers

The Deep Disfluency tagger accepts input sequences (and optionally, external POS tags and word timings) word-by-word and outputs XML-style tags for each word, symbolising disfluencies in terms of complex repairs or edit terms. The full tagset consists of:

`'<e/>'`	an edit term word, not necessarily inside a repair structure
`'<rms id=N/>'`	reparandum start word for repair with ID number N

[3]`https://www.ibm.com/watson/developercloud/speech-to-text.html`Watson

`‘<rm id=N/>‘`	mid-reparandum word for repair N
`‘<i id=N/>‘`	interregnum word for repair N
`‘<rps id=N/>‘`	repair onset word for repair N (where N is normally the 0-indexed position in the sequence)
`‘<rp id=N/>‘`	mid-repair word for repair N
`‘<rpn id=N/>‘`	repair end word for substitution or repetition repair N
`‘<rpndel id=N/>‘`	repair end word for a delete repair N

Every detected repair (and gold standard entry) will exhibit at least the `rms`, `rpS` and `rpn/rpndel` tags, others might be omitted.

Two example outputs on Switchboard utterances are shown below, where <f/> is the default tag for a fluent word:

```
4617:A:15:h    1   uh          UH     <e/>
               2   i           PRP    <f/>
               3   dont        VBPRB  <f/>
               4   know        VB     <f/>

4617:A:16:sd   1   the         DT     <rms id="1"/>
               2   the         DT     <rps id="1"/><rpn id="1"/>
               3   things      NNS    <f/>
               4   they        PRP    <f/>
               5   asked       VBD    <f/>
               6   to          TO     <f/>
               7   talk        VB     <f/>
               8   about       IN     <f/>
               9   were        VBD    <f/>
               10  whether     IN     <rms id="12"/>
               11  the         DT     <rm id="12"/>
               12  uh          UH     <i id="12"/><e/>
               13  whether     IN     <rps id="12"/>
               14  the         DT     <rpn id="12"/>
               15  judge       NN     <f/>
               16  should      MD     <f/>
               17  be          VB     <f/>
               18  the         DT     <f/>
               19  one         NN     <f/>
               20  that        WDT    <f/>
               21  does        VBZ    <f/>
               22  the         DT     <f/>
               23  uh          UH     <e/>
               24  sentencing  NN     <f/>
```

The authors compare Elman Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) network architectures to train the essential disfluency detection and prediction component of the system. The machine learning library *Theano* is used to implement both networks.

While both architectures are recurrent in nature, LSTMs present a special case of the RNN architecture. Both often exhibit distinct behaviour on different tasks and it is apriori unclear which model would outperform the other. Albeit their similar foundation, the main difference lies in the scope of previously seen information each architecture can take into consideration when predicting the next output. RNNs perform best when the information they require for a prediction is available relatively close to the current input, whereas LSTMs can learn to take information into account that is potentially further away.

All neural network models in the disfluency experiments are trained on identical training sets (within the given experiment), either up to a maximum of 50 epochs or until their parameters converge.

For POS tagging, the system uses the NLTK CRF tagger, which in turn utilizes the *crfsuite* package[4], trained with Limited-memory BFGS (L-BFGS) gradient descent optimization on the training set of the SWdA corpus and evaluated in terms of accuracy against its test split.

8.3 Analytical Reproducibility

In this section, we describe the implementation of the Deep Disfluency library, a software library that is designed for the analysis of voice and textual data. The DSG group were aware of our efforts and interested in the computational reproducibility of their research publications. They made efforts to open up their research by using open toolkits and publishing code and data to a public git repository. The authors publish the library under the permissive free MIT license through an organisational account on GitHub. The project repository contains both the source code for their, Python-based libre, software stack as well as a great deal of raw and intermittent data to reproduce various experiments.

Deep Disfluency

The Deep Disfluency code is packaged and available on a GitHub repository with documentation outlining data availability and the installation process. End users can install the system as a regular Python package via *pip*, the Python installer. The package can be obtained from the public *Python Package Index* (PyPI, using `pip install deep_disfluency`) or locally installed from source using *setuptools* (`pip install setuptools`). At the time of writing, the project depends on Python 2.7 and conveniently documents all packages that were used for

[4]`https://pypi.python.org/pypi/python-crfsuite`

development in a *virtualenv* and *pip* compatible 'requirements.txt' file. This mechanism is used to manage dependencies and retains the exact version of all external library dependencies (`pip install −r requirements.txt`). Some of the *Theano* related requirements, especially for GPU enabled computation, are easier installed through the alternative repositories offered by *conda*, which is part of the *anaconda* platform for data science with Python[5]. Corpus data are either bundled with the package or can automatically be downloaded via an installation script that pulls the data and stores it locally. Internally, the structure of the disfluency system is straightforward and clearly separated into different sections:

- *ASR*, for interacting with the ASR system,
- *Corpus*, for handling different corpora,
- *data*, containing raw data (if allowed by the respective license), as well as intermittent results,
- *experiments*, for reproducing individual experiments and analysis,
- *tagger*, containing the main deep learning model implementation in 'tagger/deep_tagger.py', and
- *decoder*, where a Hidden Markov Model (HMM) is implemented that combines timing information and outputs from the network model in addition to enforcing some model constraints on the output sequence.

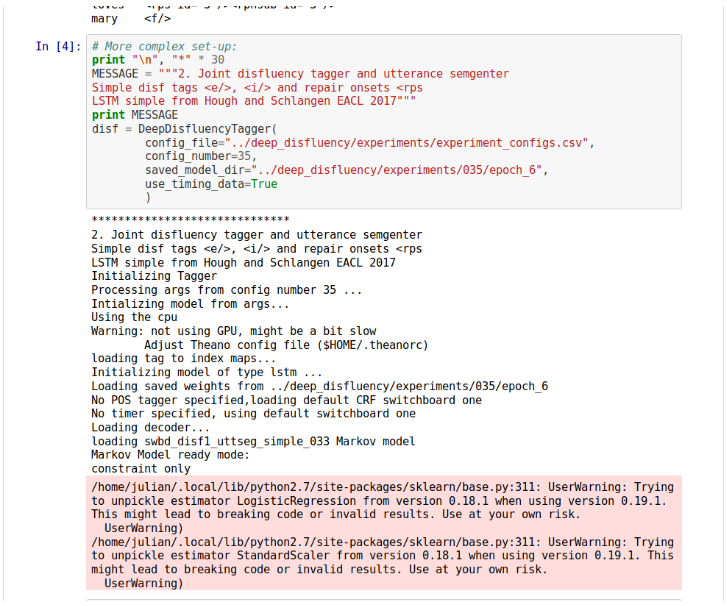
Other auxiliary parts of the system are analogouslyresponsible for one specific subtask only. After the package and its requirements have been installed, the documentation within the repository leads users to either try out a demonstration code or follow the instructions to reproduce individual experiments the system was previously used for. The latter can be used to reevaluate the experiments using RNN and LSTM models, either utilizing the pretrained models provided by the authors or training the full system from scratch. As outlined in figure 3 of the original paper, the system uses Viterbi decoding on a HMM to enforce some constraints on the final output sequence and include timing information from the transcribed corpus and ASR systems. As a crucial input feature, a part-of-speech (POS) tagger for the system was trained on in-domain Switchboard data. The implementation is based on NLTK and the resulting Conditional Random Field (CRF) model is packaged alongside the library.

Many parts of the system are modular and provide sensible defaults, e.g. if no POS tagger is specified, the library will load the default CRF tagger trained on Switchboard data. This allows end users to easily apply the disfluency detection on their own input sequences. The general pipeline that is exposed through the library of the Deep Disfluency system follows figure 3 of the original paper and

[5] `https://www.anaconda.com/`

consists of *a)* input of word embeddings and timing information, *b)* feature extraction (e.g. through POS tagging), *c)* decoding the input through a deep neural network and, optionally, *d)* combining timing information with the output of the neural network in a Hidden Markov model (HMM).

The demonstration code, located in the Jupyter [6] notebook *'demo/demo.ipynb'*, contains a set of concise examples and offers instructions on how to initialize the tagger with different configurations and pretrained models. The code also demonstrates how to tag arbitrary text sequences with the library. Figures 8.1 and 8.2 show the notebook output when the tagger and utterance segmentation system creates repair tags using RNN and LSTM configurations.



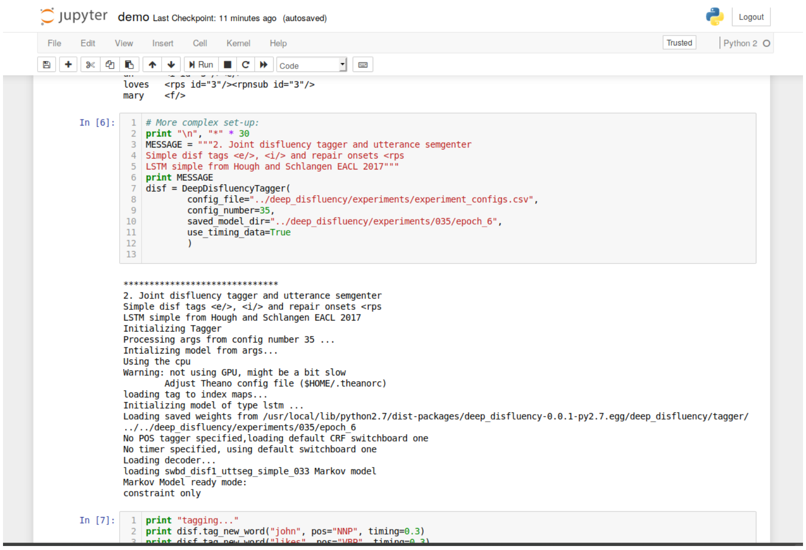
```
mary    <f/>

In [4]: # More complex set-up:
        print "\n", "*" * 30
        MESSAGE = """2. Joint disfluency tagger and utterance semgenter
        Simple disf tags <e/>, <i/> and repair onsets <rps
        LSTM simple from Hough and Schlangen EACL 2017"""
        print MESSAGE
        disf = DeepDisfluencyTagger(
                config_file="../deep_disfluency/experiments/experiment_configs.csv",
                config_number=35,
                saved_model_dir="../deep_disfluency/experiments/035/epoch_6",
                use_timing_data=True
                )

        ******************************
        2. Joint disfluency tagger and utterance semgenter
        Simple disf tags <e/>, <i/> and repair onsets <rps
        LSTM simple from Hough and Schlangen EACL 2017
        Initializing Tagger
        Processing args from config number 35 ...
        Intializing model from args...
        Using the cpu
        Warning: not using GPU, might be a bit slow
                Adjust Theano config file ($HOME/.theanorc)
        loading tag to index maps...
        Initializing model of type lstm ...
        Loading saved weights from ../deep_disfluency/experiments/035/epoch_6
        No POS tagger specified,loading default CRF switchboard one
        No timer specified, using default switchboard one
        Loading decoder...
        loading swbd_disf1_uttseg_simple_033 Markov model
        Markov Model ready mode:
        constraint only
        /home/julian/.local/lib/python2.7/site-packages/sklearn/base.py:311: UserWarning: Trying
        to unpickle estimator LogisticRegression from version 0.18.1 when using version 0.19.1.
        This might lead to breaking code or invalid results. Use at your own risk.
          UserWarning)
        /home/julian/.local/lib/python2.7/site-packages/sklearn/base.py:311: UserWarning: Trying
        to unpickle estimator StandardScaler from version 0.18.1 when using version 0.19.1. This
        might lead to breaking code or invalid results. Use at your own risk.
          UserWarning)
```

Figure 8.1: Tagger output in the Deep Disfluency demo.ipynb file

Software Toolkit and File Formats

The authors make use of a Free and Open Source Software (FOSS) based Python stack for development consisting of different NLP libraries, like NLTK; with machine learning libraries like Theano (now defunct) for deep learning. The library is currently implemented as a Python package targeting Python 2.7 environments, although a Python 3 port seems to be available. Proper packaging provides some metadata for the code itself and allows end users to install the full library, along with all dependencies, through convenient and well accepted mechanisms.

[6] https://jupyter.org/



```
loves    <rps id="3"/><rpnsub id="3"/>
mary     <f/>
```

In [6]:
```
# More complex set-up:
print "\n", "*" * 30
MESSAGE = """2. Joint disfluency tagger and utterance semgenter
Simple disf tags <e/>, <i/> and repair onsets <rps
LSTM simple from Hough and Schlangen EACL 2017"""
print MESSAGE
disf = DeepDisfluencyTagger(
        config_file="../deep_disfluency/experiments/experiment_configs.csv",
        config_number=35,
        saved_model_dir="../deep_disfluency/experiments/035/epoch_6",
        use_timing_data=True
        )

```

```
******************************
2. Joint disfluency tagger and utterance semgenter
Simple disf tags <e/>, <i/> and repair onsets <rps
LSTM simple from Hough and Schlangen EACL 2017
Initializing Tagger
Processing args from config number 35 ...
Intializing model from args...
Using the cpu
Warning: not using GPU, might be a bit slow
        Adjust Theano config file ($HOME/.theanorc)
loading tag to index maps...
Initializing model of type lstm ...
Loading saved weights from /usr/local/lib/python2.7/dist-packages/deep_disfluency-0.0.1-py2.7.egg/deep_disfluency/tagger/
../../deep_disfluency/experiments/035/epoch_6
No POS tagger specified,loading default CRF switchboard one
No timer specified, using default switchboard one
Loading decoder...
loading swbd_disf1_uttseg_simple_033 Markov model
Markov Model ready mode:
constraint only
```

In [7]:
```
print "tagging..."
print disf.tag_new_word("john", pos="NNP", timing=0.3)
```

Figure 8.2: Tagger output from the local demo.ipynb file

The Deep Disfluency package collectively specifies 85 direct dependencies, most of which are standard libraries commonly used in the NLP space. These dependencies should be readily accessible to all end users. Some relevant examples are listed below, we address the defunct *Theano* dependency in more detail as part of the following section.

- **NLTK**: natural language processing for the SwDA corpus readers and CRF implementations
- **gensim**: vector space modeling and topic modeling toolkit
- **Jupyter**: Jupyter notebook used to house parts of the analysis and visualisations
- **Keras**: a neural network library that seems to be used as an initial alternative for the Theano LSTM implementation
- **matplotlib**: a visualisation library
- **numpy**: common data structures and optimized algorithms for mathematical computing
- **pandas**: Library for working with complex data representations such as time series; also includes facilities for data manipulation and analysis
- **scikit-learn**: a machine learning library

- **scipy**: scientific and technical computing algorithms such as optimization, linear algebra, FFT
- **Theano**: (Defunct) Optimization and evaluation of mathematical expressions (including GPU computation); used for the main neural network implementations of the paper

The system makes use of a number of file formats, all of them well documented and accessible through open source frameworks. POS tagged corpora, ASR outputs and Switchboard transcriptions are stored as structured text files, comma-separated values (CSV) in the latter case. Most data artifacts created during experiments, e.g. model weights for the neural network, are serialized using the underlying libraries to create reusable *numpy* matrices. The decoder component forms a notable exception in using the Python package *pickle* for serialization. This format is specific to Python and guaranteed to offer backwards compatibility, enabling portable models between different versions. When used as a library, a convenient Python interface makes all internal file formats transparent and allows users to submit their own pre-segmented tokens for predictions through code instead.

Technical Challenges and Issues

When reproducing the main results of the paper with the Deep Disfluency library, we faced the following problems and challenges:

Dependencies: While most of the dependencies of the project are still under active development and maintenance, two minor issues were noteworthy for future reproductions:

Theano: The neural network component of the Deep Disfluency library is based on *Theano* which has been declared defunct as of 2017, when support ceased following the 1.0 release. The machine learning library originated from the Montreal Institute for Learning Algorithms (MILA), University of Montreal, who ended development and ceased implementing new features. The library shifted to low-maintenance mode, i.e. one should not rely on security bug fixes or patches being implemented at this point. At the time of this writing, a few maintainers seem to still actively commit and merge pull requests (PR) on the GitHub repository. While the deprecation of *Theano* does not, at present, hinder executing the code, it presents a potential danger which affects sustainability and makes it costlier to maintain a dependency to the library. Subsequent work should possibly make an effort to replace the affected parts of the system. Another downside that became apparent when installing the library in an environment where the precompiled dependencies of the *anaconda* repositories were unavailable is that some *Theano* dependencies require rather complicated manual setup routines and compilation on the target architecture.

Python 2.7: The Deep Disfluency library is written for Python 2.7 which has a planned end of life in the year 2020. The library will have to migrate to Python 3.x and potentially be restructured to accommodate a replacement for the machine learning library Theano. While such migrations are no trivial task in terms of time and effort, an open pull request on the GitHub repository indicates that a port to Python 3 is either under active development or already completed.

Original Data: The primary raw data used in the disfluency research project for ASR of live voice recordings was unavailable for the Conquaire reproducibility experiment. Interested parties could acquire the raw audio dataset through a subscription to the LDC Catalog[7]. Since the project retained their output of the ASR component (in 'data/asr_results/'), this does not pose a problem to reproducibility. The process on retraining the system with the original dataset is also preserved and well documented. Furthermore, the authors bundled the corpus of manually transcribed Switchboard data. We focus on this transcription based corpus since it is more readily available and can reproduce the main claims of the original paper.

8.4 Summary of reproducibility experiment

The library was installed from source in an environment equipped with hardware for computation on graphical processing units (GPUs), since the neural networking components within the Deep Disfluency system are capable of taking advantage of such hardware. The setup process through the Python packaging mechanisms did not present any major difficulties and, aside from the environment specific Theano dependency problems described earlier, could be performed just as detailed in the project documentation.

Since it is an isolated compontent that has large effect on data quality within the system, we initially verified the reported performance of the CRF used for POS tagging. The claimed accuracies of 0.915 (overall) and 0.959 (for the *UH* label) on the Switchboard test set could be easily and exactly reproduced. Invoking the feature extraction code[8], with the *TEST* flag set to *True*, loads the pretrained model that was used in the original experiments and evaluates it automatically.

Other parts of the original experiments were then repeated. The authors fortunately aggregate most of the code for the described experiment in *a)* a

[7]`https://catalog.ldc.upenn.edu/`

[8]located at 'deep_disfluency/feature_extraction/POS_Tagging.py '

Python program for training and generating evaluation data on the test sets and *b)* a Jupyter notebook for evaluation of the data generated by the different experiments.

All experiments come with a configuration entry of hyper parameters in the ‘experiment_configs.csv‘ file. This file not only controls the neural network architecture used in an experiment, it also documents important details such as hidden layer sizes and learning rates. This level of documentation and parametrization is vastly conducive to replaying experiments the way they were originally performed.

Since the authors included the best performing epochs of their original training, we opted to rerun the evaluation on the test set of the SWdA transcription corpus. Both programs involved in this worked out-of-the-box since the whole codebase makes an effort to use relative paths when referring to data files or cached models. This made switching the Jupyter Notebook used for analysis a matter of pointing a single directory from the original repository data to that of our new run. We then investigated parts of this output in regards to the original outcome.

System	F_{rps}(per word)	F_e(per word)	F_{uttSeg}(per word)	NIST SU
LSTM +timing	**0.693**	0.864	0.654	58.401
LSTM	0.665	0.862	0.666	59.714
LSTM (complex) +timing	0.655	**0.909**	0.680	**56.544**
LSTM (complex)	0.655	0.907	**0.683**	58.231
RNN +timing	0.660	0.839	0.602	68.064
RNN	0.639	0.835	0.607	70.160
RNN (complex) +timing	0.633	0.904	0.653	59.254
RNN (complex)	0.627	0.903	0.662	60.072

Table 8.4: Reproduction of results in table 2 from the original paper.

While our run of the evaluation did not produce the exact results from the original paper, they seem to be close and lead to mostly the same conclusions. The LSTM generally outperforms the RNN architecture as evident in table 8.4, which reproduces parts of table 2 in the original paper. The reported best values on the transcript corpus were $F_e = 0.918$ (LSTM) for repair onsets and $F_{rps} = 0.720$ (LSTM+timing) for editing terms, the reproduced ones reach marginally lower results ($\Delta F_e = -0.09, \Delta F_{rps} = -0.027$). Notable differences are that *a)* the reproduction yields the best F_{rps} score on the LSTM+timing model with complex tags, whereas the original analysis seems to prefer the simple tagset with timings, and *b)* the reproduction seems to exhibit consistently raised utterance segmentation error rates (NIST SU) when compared to the original.

The reproduced results on joint vs. separate tasks are similarly close to the original, see table 8.5 (consistently higher NIST SU error rates remain visible here). These data do not necessarily match the conclusions of the original paper, since the joint task formulation seems to only outperform others in terms of repair onset detection accuracy (F_{rps}) but fails to do so in terms of NIST SU rate, accuracy of edit term words (F_e), and utterance boundary detection (F_{uttSeg}). This might indicate a difference in computing environments rather than wrong results since the variance of results in our reevaluation seems generally higher. This could stem from differences in dependencies that we had to setup manually, or even differences in hardware, especially since GPU acceleration was involved in the reproduction. We executed the evaluation on a node equipped with *nVidia GeForce GTX 1080 Ti* graphic cards, invoked in a cluster environment.

System	F_{rps} (per word)	F_e (per word)	F_{uttSeg} (per word)	NIST-SU
LSTM (uttSeg only)	-	-	**0.720**	**50.222**
LSTM (disf only)	0.658	**0.912**	-	-
LSTM (joint task)	**0.693**	0.864	0.654	58.401

Table 8.5: Reproduction of results in table 3 from the original paper.

Similar small deviations can be observed regarding the re-evaluated data in table 8.6, this corresponds to table 4 of the original paper and presents the performance of incremental results over the transcript corpus. Repair onset detection in terms of words follows the findings of the original publication, with the simple LSTM model outperforming the complex ones. TTD_{rps} measured over time shows more variance than the original data, after corresponding with the authors we suspect this is likely to be an error in how the evaluation scripts aggregate the results. Even with slightly different values, the clear winner in this metric remains the simple LSTM model. In terms of edit overhead (EO) measure, the new evaluation follows the same trends between systems as originally reported. Here, the complex LSTM model that incorporates timing information clearly outperforms the simpler approaches.

The library and data for this project were generally very accessible. The researchers managed to provide an intuitive abstraction layer around their complex system of underlying data models. By bundling not only their final models but also the data used to produce them, they enable other researchers to reproduce results and adapt the system for their own corpora. Free and Open Source Software (FOSS) plays a significant role in reproducing the above results since it enables others to closely match the original environment in which an experiment was performed. We discuss how these aspects affected the reproduction and facilitates data-sharing initiatives in the following section.

System	TTD_{rps} (word)	TTD_{rps} (time in s)	Edit Overhead (word)
LSTM + timing	**0.001**	1.151	10.282
LSTM	**0.001**	**0.763**	10.735
LSTM (complex) +timing	0.104	1.093	**8.577**
LSTM (complex)	0.123	0.855	9.972

Table 8.6: Partial reproduction of the results in table 4 (incremental results for transcript level systems) from the original paper.

Discussion of the reproducibility experiment

Through the public GitHub repository and requirements documentation within the Python ecosystem we were able to reproduce most of the software environment that was used in the original experiments. Some details, such as GPU acceleration and other hardware dependent factors are subject to continuous improvement and cannot be reliably reproduced. By using compatible versions, a best effort was made to get as close as possible to the original setup within the reproduction setting. All major parts of the analytical pipeline were well documented and the authors made visible efforts to comply with many principles of good scientific data management: Findability, Accessibility, Interoperability, and Reusability (FAIR) [9] [3]. The system can be found in a public GitHub repository that presents an aggregation of all the necessary source code, documentation and most of the underlying research data that allows others to use and analyse the system. By packaging their resulting models and exposing a concise Application Programming Interface (API) to their library, the project facilitates re-use of the system as a whole in follow-up and related tasks. The project bundles sufficient instructions and programs to download all external data researchers might need in the context of the original experiments. Much of the raw data that forms the basis of the experiments is widely available. While licensing prevents the project from including the raw voice recordings used to create the ASR models, the dataset is obtainable through reliable sources and the extensive research that has already been performed on it indicates that it will likely remain accessible in the foreseeable future. The authors also provided trained models and the intermittent results they used at the time of publishing, which - in terms of reproducibility - might even be preferable over the raw data due to possible changes in the external ASR system that was used at the time. Additional research is encouraged by maintaining a copy of the Switchboard SWdA corpus itself in a separate repository[10], without having to incorporate the full disfluency system as a dependency. The system allowed us to setup a devel-

[9]`https://www.go-fair.org/fair-principles/`

[10]`https://github.com/julianhough/swda`

opment environment in short time and enabled us to independently reevaluate the models that were generated in the original experiments. The documentation, along with the scientific paper itself, provide sufficient information to gain familiarity with the codebase. While the project does not currently include an explicit description of semantic metadata, the library provides enough of an abstraction to be interoperable with any external data source. This enabled us to exactly verify parts of the original results, namely the performance of the CRF used for domain-optimized POS tagging. The full and more complex experimental settings could so far be partially reproduced through the Deep Disfluency system and original evaluation scripts, which the authors helpfully retained and separated by publication. We have been able to recreate similar results on some of the models, whereas differences in other parts of the results remain open for further investigation. Since the software used for reproduction was almost identical to the original, the reproduction did not have any major problems to re-use even the intermittent data packaged in the repository. Possible explanations for these deviations might include differences in hardware and subsequently different behaviour in terms of numerical processing or similar incompatibilities.

8.5 Conclusion

This chapter showcased a case study from the field of speech recognition and computational linguistics. The particular task was to detect disfluency markers and edit terms in spoken language (or transcriptions). This is used to detect repairs for vocal input or facilitate better detection of turn taking opportunities for subsequent tasks, e.g. in a conversational setting between human and computational agents. We were able to partially reproduce the major claims of the original paper by invoking the system's evaluation scripts on existing data and models in a completely new and independent environment. While comparisons between performance on incremental ASR output and transcription corpora had to be deferred due to licensing constraints, the reproduction could show some of the originally reported behaviour on the transcription corpus itself. The demonstration code for the Deep Disfluency library worked out of the box, enabling future users to adapt the system as a whole for their own corpora. Since the system itself is a complex project with multiple interacting components from data integration to machine learning, we are confident that given enough time and resources the rest of the results could be reproduced in a similar fashion. The research project is already very much aligned with FAIR data principles as it adopts open software practices and makes large parts of the original experiments easily accessible. Overall, this case corresponds to a case of limited reproducibility as the results could be partially reproduced for the offline settings, albeit not exactly.

References

[1] Schlangen D Hough J. Joint, incremental disfluency detection and utterance segmentation from speech. In *Proceedings of the International Conference of the European Chapter of the Association for Computational Linguistics (EACL).*

[2] David DeVault, Kallirroi Georgila, Ron Artstein, Fabrizio Morbini, David Traum, Stefan Scherer, Albert Skip Rizzo, and Louis-Philippe Morency. Verbal indicators of psychological distress in interactive dialogue with a virtual human. In *Proceedings of the SIGDIAL 2013 Conference*, pages 193–202. Association for Computational Linguistics, 2013.

[3] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3, 2016.

9 | Reproducing the analysis of an experiment in sequential visual processing

Rebecca Foerster[2], Philipp Cimiano[1], Werner X. Schneider[2]

1 – Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University
2 – Faculty of Psychology and Sports Science, Department of Psychology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University

Abstract

This chapter describes a case study in reproducing work conducted by the neuro-cognitive psychology research group at Bielefeld University in the area of sequential visual processing. In particular, we describe our effort to independently reproduce the results obtained via the experiment conducted in the paper *'Expectation violations in sensorimotor sequences: shifting from LTM-based attentional selection to visual search'* [1]. The research of the group focuses on the area of visual attention, eye movements, working memory, transsaccadic learning, and sensorimotor learning. The group works on understanding visual processing in humans via controlled behavioral experiments in laboratory environments alongside real-world studies. The main result of the article mentioned above was the finding that expectation violations in a well-learned sensorimotor sequence in humans caused a regression from LTM-based attentional selection to visual search. We describe in this paper our efforts to independently reproduce these results. We conclude that this case is a case of limited analytical reproducibility in that results are reproducible by relying on SPSS as in the original data analysis or by adapting analysis codes to open-source software packages such as R. The data and scripts for this project are available at `https://gitlab.ub.uni-bielefeld.de/conquaire/neurocognitive_psychology`.

Keywords

attention, eye movements, long-term memory (LTM), visual search, sensorimotor action, expectation discrepancy

9.1 Introduction

The neuro-cognitive psychology group at Bielefeld University is mainly concerned with research on visual attention, visual working memory, eye movements, transsaccadic learning, and sensorimotor control and learning. A first key issue is to understand how humans employ their visual attention to control movements. A second issue refers to elementary neuro-cognitive mechanisms as well as to group differences between healthy individuals and patients in visual attention and working memory. In order to achieve these goals, the neuro-cognitive psychology group conducts controlled behavioral experiments in the laboratory as well as real-world studies. The experiments often afford highly precise presentation durations of visual material, which are achieved by employing CRT screens, G-sync LCD monitors [2], high-speed projectors or head-mounted virtual reality devices [3]. Behavioral responses (e.g., letter reports, key presses), eye movements (static and mobile eye tracking), and hand movements (motion tracking, mouse cursor tracking) as well as video data and EEG data are recorded.

Within the Conquaire project, the publication by Foerster and Schneider entitled *'Expectation violations in sensorimotor sequences: Shifting from LTM-based attentional selection to visual search'* [1] was chosen to be reproduced. In that article, the consequences of violating long-term memory (LTM) based expectations about a learned sensorimotor sequence were investigated. Especially for well-practiced sequential sensorimotor actions, such as *driving, making a sandwich or performing sports*, LTM expectations have an important role because they guide the necessary task-adapted sequence of covert shifts of attention, eye movements, and hand and body movements [4, 5, 6, 7, 8]. In the study reported in the manuscript, it was investigated which consequences arise for eye and hand movement control when a learned visuospatial configuration (fixed sequence of spatially distributed mouse clicks) was unexpectedly changed.

Results revealed that the changes of action-irrelevant visual features of the configuration had no effect, neither on hand nor eye movements. In contrast, changes of the visuospatial configuration that forced participants to update their learned sensorimotor sequence partly affected both hand and eye movements. Such changes slowed down the demanded action, they elicited visual search-like scanning that replaced the previously LTM-controlled eye movements, and they reduced the eyehand synchrony. These effects were neither limited to the changed stimuli nor to actions on them.

We describe the specific experimental settings of the original work in Section 9.2. After this description, we provide details on how we attempted to reproduce the main results of the above mentioned paper.

9.2 Methods

Here, we describe the experimental setting and the methods used in the experiments conduced in the paper *'Expectation violations in sensorimotor sequences: shifting from LTM-based attentional selection to visual search'* [1].

9.2.1 Experiment settings and Data acquisition pipeline

In order to investigate the effects of action-relevant and action-irrelevant expectation violations on eye and hand movements in [1], the following experimental design was adopted. Forty right-handed participants (mean age of 25 years, 14 male, 26 female) were recruited at Bielefeld University, with normal or corrected-to-normal vision, to participate in the computer experiment.

All participants were first trained for 60 trials to click as fast as possible with a computer mouse in ascending order on eight numbered unique shapes on a computer screen (1-8). Importantly, the spatial configuration of the numbered shapes was constant over the course of the 60 trials (Figure 9.1), so that participants could learn and automatize the visuospatial configuration of the numbered shapes as well as the clicking sequence.

Typically, an eye movement to a location preceded each clicking action on a location. Thus, participants adopted LTM expectations about the visuospatial characteristics and an LTM-based control of visual attention and eye movements. After the 60th trial, we violated these visuospatial expectations unannounced, so that the 20 consecutive trials had a different configuration.

The 40 participants were divided into four experimental groups of 10 participants each, depending on the changed features during the 20 change trials.

- In the **shape-change group**, the shapes (circle and a plus sign) surrounding the numbers *3* and *6* switched position.
- In the **number-change group**, the numbers *3* and *6* changed position without changes in the surrounding shapes.
- In the **object-change group**, the numbers *3* and *6* switched position together with their surrounding shapes, so that the objects remained constant, e.g., a *plus 3* and a *circle 6*.
- In the **no change control group**, no switch was introduced.

As the **shape change** does not require a change of the learned clicking action, we call this an **action-irrelevant change**. As the **number and object changes** do affect the learned clicking sequence, we call these changes **action-relevant**.

In order to investigate how previously learned expectations and sensorimotor sequences can be re-initiated, 20 reversion trials followed the 20 change trials, in which the configuration was the same as during the 60 pre-change trials for each participant.

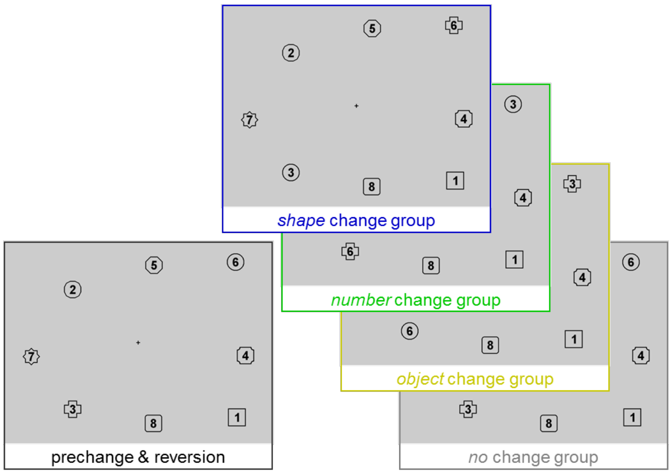


Figure 9.1: Computer display in the clicking task experiment

Figure 9.1 shows the display during the clicking task in the prechange (left), change (right), and reversion (left) phase of the experiment for the even participants of the four change groups (shape, number, object, no). Odd participants started with the plus three in the upper right position and the circle six in the lower left position.

Throughout the whole experiment, cursor movements on the CRT computer screen (ViewSonic Graphics Series G90fB, 19 inch color monitor @ 1024 x 768 pixels) were recorded with 100 Hz and participants' right gaze positions were recorded with an Eyelink 1000 desktop-mounted eye-tracker (SR Research, Ontario, Canada) with 1000 Hz. A standard computer mouse and an extra-large mouse-pad (32 x 88 cm) were used. A forehead and chin rest was used to fix participants' viewing distance at 71 cm. All stimuli were presented in black on a grey background. The cursor was a black dot subtending approximately 0.45° v.a. (degrees of visual angle) in central vision. A black plus sign with a height and width of 0.45° v.a. was presented in the screen center. The numbers were presented in bold Arial font with a font size of 35. Each number was surrounded by one unique shape with a diameter of about 2.18° v.a. in central viewing. The pre-change arrangement of the numbered shapes was generated randomly with the prerequisite that each outer field of an imagined 3 x 3 grid contained one shape and that the distance between shapes as well as the distance to the screen border was at least 2.18° v.a (border to border). For the generated configuration, the actual minimal distance happened to be 7.20° v.a. between the shapes containing numbers 1 and 4.

All participants saw numbers 1, 2, 4, 5, 7, and 8 in the same individual shapes and at the same location (Figure 1). Even participants saw a plus 3 in the lower-left corner and a circle 6 in the upper-right corner during the pre-change phase, while odd participants began with the switched position of plus 3 and circle 6. Each experimental group consisted of an equal number of odd and even participants, so that possible variations in the difficulties of the trajectories were cross-balanced.

The experiment was controlled by SR Research's Experiment Builder software. A nine-point eye-tracking calibration and validation procedure with an averaged accuracy criterion of 1.0° v.a. preceded the experiment. Calibration accuracy was checked before each trial on the basis of a central fixation on a black ring (0.48° v.a. outer size, 0.12° v.a inner size). Calibration was repeated if necessary.

After reading an initial written instruction on the computer screen, participants completed an example pre-change trial before the experiment started. This practice trial was not included in the analysis. Clicks were counted as correct within a diameter of 3.27° v.a. around a target's center. An incorrect click was followed by a low-pitched tone. After all eight objects were clicked sequentially in the correct order, participants were informed about their trial-completion time via a feedback display. After every block of 10 trials, a display informed participants about the number of blocks completed out of the total number of blocks. Participants started a block and a trial by pressing the space bar. All participants completed the experiment within 40 minutes.

Fixation detection

Fixations were detected by SR Research's default velocity algorithm (not a blink, velocity $<30°$ v.a./s and acceleration $< 8000°$ v.a./s2). The following dependent variables were analyzed:

- trial-completion time,
- number and size of errors,
- number and duration of fixations,
- scan-path length,
- cursor-path length, and
- eyecursor distance.

Error size was measured as the Euclidean distance (° v.a.) from the center of the actual target to the incorrectly clicked location. Scan-path and cursor-path lengths were calculated as 100-Hz cumulative inter-sample distances. Eyecursor distances were calculated as 100-Hz intra-sample distances. For pre-change analyses, repeated measures analyses of variances (ANOVAs) with the within-subject factor block (6) were calculated for each dependent variable over all groups.

9.2.2 Methods applied to analyze the experiment data

For the change analyses, mixed design ANOVAs were calculated with change group (shape, number, object, no) as *between-subject factor* and phase (pre-change, change, reversion) as *within-subject factor*. For more fine-grained analyses, further ANOVAs were calculated including sub-action (8), location (8), and fixation type (searching, guiding, checking) as within-subject factors. *Guiding fixations* are fixations on current action goals, also known as *sequence or directing fixations* [9, 10, 11, 1, 8]. In the study, guiding fixations were operationalized as fixations to the numbered shape that was the current clicking target. *Checking fixations* are fixations to objects and locations that have already been acted on in the nearer past [10, 11, 1, 8]. In the study, checking fixations were operationalized as fixations to numbered shapes that had already been clicked correctly. *Searching fixations* are fixations to objects and locations that are currently not action-relevant, were not relevant shortly before, but might become relevant in the later future [9, 11, 1]. In the study, searching fixations were operationalized as fixations to numbered shapes that had not yet been clicking targets. Fixations were counted as falling on a numbered shape within an area of 3.27° v.a. around its center.

A LTM mode of visual attention is characterized by about one guiding fixation per sub-action of the task and nearly no checking or searching fixations, while searching fixations are indicative for visual search. Paired t-tests were conducted in case of significant two-way ANOVA interactions to reveal whether the values of two phases were significantly different across groups, sub-actions or locations. Violations of sphericity were corrected using Greenhouse-Geisser ϵ, but uncorrected degrees of freedom were reported to facilitate reading. A chance level of 0.05 was applied. Data preprocessing was conducted with MATLAB 2012a, data aggregation and diagrams were compiled in Microsoft Excel 2010, and statistical analyses were conducted with IBM SPSS Statistics 22.

The shape change did not affect any dependent variable significantly, neither when comparing the shape-change to the control group nor when comparing the shape-change phase values to the pre-change values. However, all dependent variables were strongly affected in the number and object change group with their values during the change phase differing from the pre-change values as well as from the control group. Specifically, participants of the number and object change group were slower, made more fixations, had longer scan-paths and cursor-paths and a larger eye-cursor distance during the first change trial than during the last pre-change block (pre-change baseline) as well as compared to the participants in the control group. Note that other pre-change baselines did not change the result pattern. Statistics can be viewed in the original paper.

Moreover, the type and size of the effects did not differ significantly between the number change group and the object change group. Therefore, these two groups were aggregated to one action-relevant change group for further analyses. The main results of these analyses were concerned with the number of fixations and fixation types performed by the action-relevant change group during the change compared to the pre-change phase.

To reveal which mode of attentional selection was predominantly applied before and after the action-relevant change, a repeated measures ANOVA was computed for the number of fixations with phase (pre-change, change) and fixation type (checking, guiding, searching) as within-subject factors. The analysis revealed significant main effects and a significant interaction on the number of fixations (phase: $F(1,19) = 23.97$, $p < 0.001$, $\eta_p^2 = 0.56$; type: $F(2,38) = 89.23$, $p < 0.001$, $\eta_p^2 = 0.82$; interaction: $F(2,38) = 21.49$, $\epsilon = 0.77$, $p < 0.001$, $\eta_p^2 = 0.53$; Figure 9.2, top). Paired t-tests revealed that the interaction was due to the fact that the change increased the number of searching fixations ($t(19) = 7.31$, $p < 0.001$), while there was no significant effect on the number of checking ($t(19) = 1.81$, $p = 0.09$) or guiding ($t(19) = 0.29$, $p = 0.80$) fixations. With nearly no checking (0.26) or searching (0.91) fixations per pre-change trial, LTM-based attention seemed to be the dominant mode of attentional selection after having learned the clicking sequence on the constant visuospatial configuration. The increase to about 5 searching fixations in the trial with the action-relevant number switch indicates a re-initiation of visual search.

Given that number 3 was no longer in the expected location, searching for the 3 when having to act on it is inevitable. Therefore, the question arises, whether searching is restricted to this action 3 or whether visual search is also initiated for other actions. To reveal whether searching fixations were differently prominent for different sub-actions, a repeated measures ANOVA was conducted for the number of searching fixations with the within-subject factors phase (pre-change, change) and action (1-7). The analysis for the number of searching fixations (Figure 2, middle) revealed significant main effects of phase ($F(1,19) = 53.43$, $p < 0.001$, $\eta_p^2 = 0.74$) and action ($F(6,114) = 20.85$, $\epsilon = 0.42$, $p < 0.001$, $\eta_p^2 = 0.52$) as well as a significant interaction ($F(6,114) = 19.74$, $\epsilon = 0.48$, $p < 0.001$, $\eta_p^2 = 0.51$). Paired t-tests revealed that the number of searching fixations was significantly increased during actions 3 ($p < 0.001$) and 4 ($p < 0.01$). Thus, searching fixations increased as soon as the first location-shifted number became the action target, but their increase was not limited to this action.

Do participants really search or is the increase in searching fixations completely explained by the fixations to the old position of number 3, i.e. the new number 6, which is by definition a not yet completed target? A repeated measures ANOVA for searching fixations with the within-subject factors location (2-8) and phase (pre-change, change) revealed two main effects and a significant interaction (location: $F(6,114) = 7.32$, $\epsilon = 0.28$, $p < 0.001$, $\eta_p 2 = 0.28$; phase: $F(1,19) = 44.80$, $p < 0.001$, $\eta_p^2 = 0.70$; interaction: $F(6,114) = 5.80$, $\epsilon = 0.45$, $p < 0.01$, $\eta_p^2 = 0.23$; Figure 2, bottom). Paired t-tests revealed that significantly

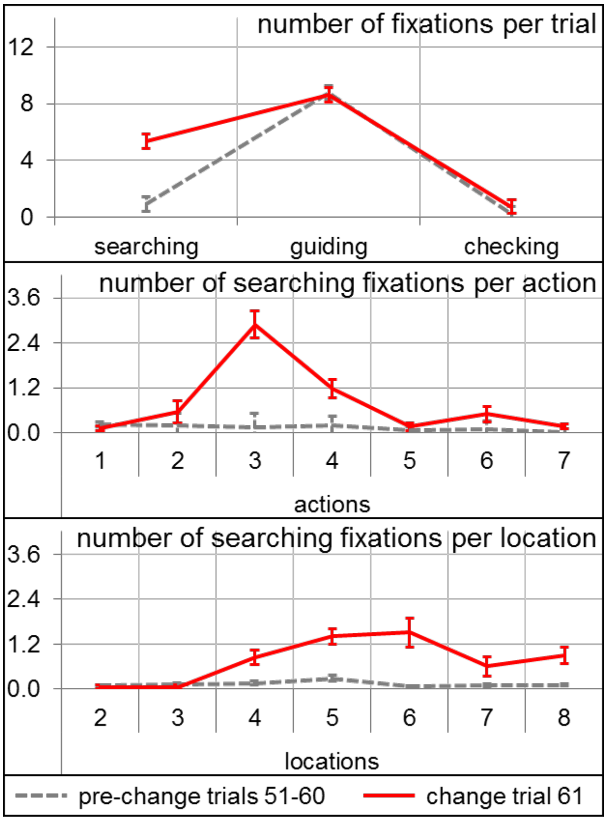


Figure 9.2: **Top panel**: Number of fixations per trial of the three fixation types searching, guiding, and checking. This is Fig.4 (top) from the original paper. **Middle panel**: Number of searching fixations per action (1-7). No searching fixations can be made during action 8 as there are no future targets. This panel is Fig. 5c from the original paper. **Bottom panel**: Number of searching fixations per location (2-8). No searching fixations can be made on location 1, as this location is never a future target. This figure is not in the original paper!

more searching fixations went to the locations 4-6 and 8 (all *p*s $< .0.5$), but not to the locations 2 ($p = 0.48$), 3 ($p = 0.24$), and 7 ($p = 0.06$). Thus, the increase in searching fixations is not limited to the new location of the 6, indicating that participants really search through the display.

To reveal how long it takes to incorporate the new clicking sequence, the number of searching fixations during the subsequent change trials was compared via paired *t*-tests to the pre-change baseline. Results revealed significantly more searching fixations compared to the pre-change baseline in the first 15 repetitions of the changed number display (all *p*s < 0.05; Figure 9.3). This result indicates that far more than a single trial is necessary to update parts of a learned sensorimotor sequence. Thus, sensorimotor updating of unexpected target locations can clearly be differentiated from surprise effects to unexpected visual items as surprise effects are typically very short-lived [12, 13, 14].

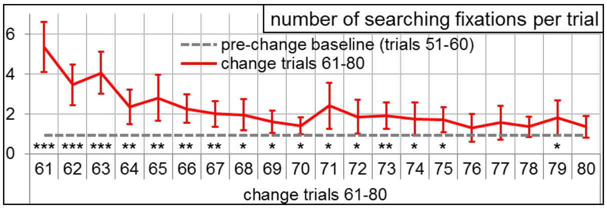


Figure 9.3: Searching fixations per change trial

Figure 9.3, which is not a part of the original published paper, shows the number of searching fixations per change trial (trials 61-80) in red solid lines along with the pre-change baseline (average of trials 51-60). The error bars represent the two-sided 95%-confidence intervals of the paired *t*-tests comparing the respective trial to the pre-change baseline. Asterisks indicate the two-tailored significance level (*<0.05, **<0.01, and ***<0.001).

In summary, *only the action-relevant expectation violations affected participants' manual performance and eye movements*. In this case, participants are forced to update a learned sensorimotor sequence. Thus, they *regressed from LTM-based attention and gaze control to visual search*. They maintained this search mode after having acted on the first changed number in the sequence as well as for up to 15 repetitions of the new configuration.

9.3 Analytical Reproducibility

The goal of the reproducibility experiment was to independently verify the report about performance improvements during the prechange/acquisition phase ensuring that participants adopted LTM-based attentional selection for the sensorimotor sequence. Secondly, we verified the effects of different expectation-violation manipulations on performance, eye movements and the three fixation types, allowing conclusions about the modes of attention selection, i.e. LTM versus visual search. Lastly, we verified the analysis of the repeated expectation violations updating the sensorimotor sequence based on the previously learned visuospatial task configurations which affected the mode of attentional control.

9.3.1 Research Data

The data for the entire research group are analyzed by proprietary as well as open and self-made analysis tools including SR Research's Data Viewer, SMI

BeGaze, Matlab, Python, R, SPSS, Excel, Annotation Tools [4, 15], and FuncSim [16, 17]. Experiments are programmed with SR Research's Experimental Builder, Matlab and PsychToolbox, Python and PsychoPy, E-Prime, or SMIs Experiment Suite. The data and scripts for the original work are available at `https://gitlab.ub.uni-bielefeld.de/conquaire/neurocognitive_psychology`. The folder structure is as follows:

- /loadevents
- /MatlabSkripteNFunctions
- /saveevents
- /SPSStabs
- SPSS command script
- Other project files (XLS sheets with results, etc.)

Primary Data

The data resulting from the experiment were available in a text format in the above mentioned */loadevents* folder.

Analytical Workflow

The researchers carried out their data analysis and processing in the MS Windows environment and for programming and computational analysis, they used Matlab and SPSS scripts that processed their data stored in text files. The first folder loadevents holds the data collected for each participant in six files (blinks, fixations, messages, results, saccades, and samples). Thus, the data recorded for 40 participants is held in 240 ".txt" files which became the source of input for further processing. The second folder MatlabSkripteNFunctions has 24 Matlab function scripts to perform the intermediate processing of combining and segregating data into separate event files and further input for Statistical Analysis through SPSS commands. The output of Matlab functions were stored in the third and fourth folders namely *saveevents.* The processing workflow is summarized in Figure 9.4

9.3.2 Analytical Reproducibility status

In order to reproduce the results described in the paper *'Expectation violations in sensorimotor sequences: shifting from LTM-based attentional selection to visual search'* [1], we reproduced the pipeline that was originally used to generate the results of the ANOVA and *t*-tests as described above. We could reproduce all the results as published in the original paper. For this, we acquired a 14-day

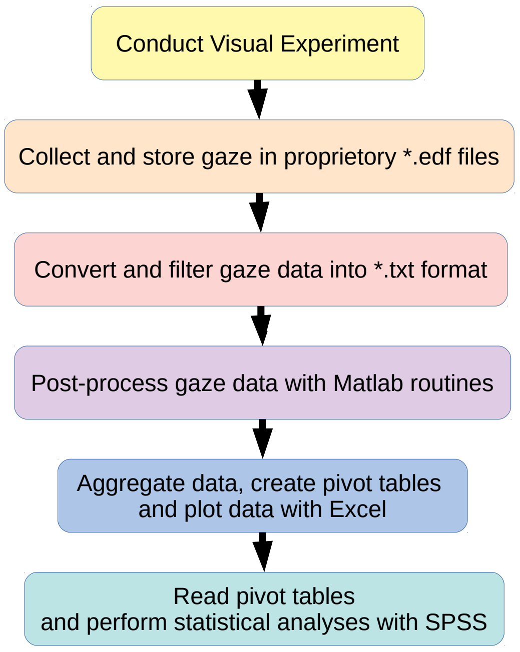

Conduct Visual Experiment
Collect and store gaze in proprietory *.edf files
Convert and filter gaze data into *.txt format
Post-process gaze data with Matlab routines
Aggregate data, create pivot tables and plot data with Excel
Read pivot tables and perform statistical analyses with SPSS

Figure 9.4: Schematic representation of the analytical workflow used in the paper by Foerster and Schneider[1]

trial version of SPSS package for the MS Windows environment from the IBM website. The SPSS script for analysis was processed to get the results published in the paper by the researchers, who confirmed that the output results were the same as the statistical results already published in the paper.

As we relied in this pipeline on proprietary and commercial software that is not freely available (Microsoft Excel, Matlab and SPSS), this is a case of limited reproducibility according to the taxonomy introduced as described in chapter 1. Thus, we also attempted to investigate whether the results could be reproduced using free and open software, in particular **Gnu-PSPP**[1] and **R**[2]. We briefly document the results of this experiment below:

SPSS vs. PSPP

PSPP was quite similar to SPSS and accepted the same code commands and in the same format as SPSS does. After making a few changes in the SPSS command file **ExpectDiscrep.sps**, the statistical tests for **NPAR TEST** and **T-TEST** ran successfully. However, other statistical functions, like **GLM TEST**, **UNIANOVA** failed as these functions are not yet implemented in PSPP. In SPSS, the GLM implements 'marginal means' but in PSPP, the **GLM** implementation is an experimental model of one-way and multiple regression linear model. So we could not reproduce the main results of the paper using PSPP.

SPSS vs. R

After investigating the use of the R-package ezANOVA, we found out that the results of the ANOVA tests could be reproduced. For the case of trend analysis, one needs to retrieve the used trends from SPSS by adding the print command 'TEST(MMATRIX)' and then insert the used contrast for the linear trend into the ezANOVA trend analysis in R. Our conclusion is thus that the results are also in principle reproducible with free and open software.

9.3.3 Discussion of reproducibility experiment

The results of the ANOVA and *t*-tests as reported in the original paper by Foerster and Schneider [1] could be independently reproduced by recreating the original analytical pipeline using the very same software stack and tools as used in the original work. This was possible because the primary data, scripts (Matlab, SPSS) and spreadsheets (Excel) were made available to the Conquaire project. Inspite of all data being available and the results being in principle reproducible, we classify this case study as an example of *limited reproducibility* as defined in the introduction to this book (see chapter 1) due to the following reasons:

[1]`https://www.gnu.org/software/pspp/`
[2]`https://www.r-project.org/`

- The analytical workflow could be reconstructed in close interaction with the authors of the original paper. The analytical workflow is not documented, so that the reproduction without guidance of the original authors is cumbersome.

- The analytical workflow relies on having installed proprietary and commercial software such as Matlab and SPSS, requiring a Windows environment for the latter. Our experiments show that substituting parts of the workflow with FOSS components, R in particular, is feasible, but this requires reprogramming the tests in R. While this is feasible, one runs the risk of creating a pipeline that is not functionally equivalent to the original one as the implementations of the tests might differ.

9.4 Conclusion

This chapter has described a case study in analytical reproducibility in the area of neuro-cognitive psychology. In particular, we have described our effort to reproduce the main results of the article by Foerster and Schneider: *'Expectation violations in sensorimotor sequences: Shifting from LTM-based attentional selection to visual search'* [1]. The main result of the article mentioned above was the finding that expectation violations in a well-learned sensorimotor sequence in humans caused a regression from LTM-based attentional selection to visual search. The authors of the original publication (also co-authors of this article) provided the Conquaire project with all primary data and all scripts and spreadsheets used to reproduce the results. While we were successful in reproducing the results, we classify this use case as one of *limited analytical reproducibility*. The reason for this is that some parts of the analytical pipeline rely on proprietary and commercial tools such as Matlab or SPSS that can not easily be replaced by open and free tools. Further, the lack of documentation of the pipeline requires interaction with the original authors to reproduce the pipeline faithfully. Both limitations could be easily overcome if further efforts are invested.

Acknowledgements

We thank Lukas Biermann and Cord Wiljes for assistance with the reproduction of the analyses.

References

[1] R.M. Foerster and W.X. Schneider. Expectation violations in sensorimotor sequences: Shifting from ltm-based attentional selection to visual search.

Annals of the New York Academy of Sciences, 1339(1):45–59, 2015.

[2] C. H. Poth, R. M. Foerster, C. Behler, U. Schwanecke, W. X. Schneider, and M. Botsch. Ultrahigh temporal resolution of visual presentation using gaming monitors and g-sync. *Behavior Research Methods*, 50(1):26–38, 2018.

[3] R. M. Foerster, C. H. Poth, C. Behler, M. Botsch, and W. X. Schneider. Using the virtual reality device oculus rift for neuropsychological assessment of visual processing capabilities. *Scientific Reports*, 6(1), 1995.

[4] R. M. Foerster, E. Carbone, H. Koesling, and W. X. Schneider. Saccadic eye movements in a high-speed bimanual stacking task: Changes of attentional control during learning and automatization. *Journal of Vision*, 11(7):1–16, 2011.

[5] M. M. Hayhoe, A. Shrivastava, R. Mruczek, and J. B. Pelz. Visual memory and motor planning in a natural task. *Journal of Vision*, 3:49–63, 2003.

[6] M. F. Land, Mennie N., and J. Rusted. The roles of vision and eye movements in the control of activities of daily living. *Perception*, 28(11):1311–1328, 1999.

[7] B. W. Land, M. F.and Tatler. Steering with the head: The visual strategy of a racing driver. *Current Biology*, 11(14):1215–1220, 2001. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/11516955.

[8] M.F. Land and B.W. Tatler. *Looking and acting.* New York: Oxford University Press., 2009.

[9] J. L. Epelboim, R. M. Steinman, E. Kowler, M. Edwards, Z. Pizlo, C. J. Erkelens, and H. Collewijn. The function of visual search and memory in sequential looking tasks. *Vision Research*, 35(23-24):3401–3422, 1995. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/8560808.

[10] R. M. Foerster. looking-at-nothing during sequential sensorimotor actions: Long-term memory-based eye scanning of remembered target locations. *Vision Research*, 144:29–37, 1995.

[11] R. M. Foerster and W. X. Schneider. Anticipatory eye movements in sensorimotor actions: On the role of guiding fixations during learning. *Cognitive Processing*, 16:227–231, 2015.

[12] R. M. Foerster. Task-irrelevant expectation violations in sequential manual actions: Evidence for a check-after-surprise mode of visual attention and eye-hand decoupling. *Frontiers in Psychology*, 7:1–12, 2016.

[13] G. Horstmann. Latency and duration of the action interruption in surprise. *Cognition & Emotion*, 20(2):242–273, 2006.

[14] A. Schützwohl. Surprise and schema strength. *Learning, Memory and Cognition*, 24(5):1182–1199, 1998. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/9747529.

[15] R. M. Foerster, Carbone E., H. Koesling, and W. X. Schneider. Saccadic eye movements in the dark while performing an automatized sequential high-speed sensorimotor task. *Journal of Vision*, 12(2):1–15, 2012.

[16] R. M. Foerster and W. X. Schneider. Tfuncsim toolbox for matlab: Computation of eyetracking scanpath similarity, 2013.

[17] R. M. Foerster and W. X Schneider. Functionally sequenced scanpath similarity method (funcsim): Comparing and evaluating scanpath similarity based on a tasks inherent sequence of functional (action) units. *Journal of Eye Movement Research*, 6(5):1–22, 2013.

10 | Reproducibility in Human-Robot Interaction Research: A Case Study

Florian Lier[1], Sebastian Meyer zu Borgsen[1], Sven Wachsmuth[1], Jasmin Bernotat[2], Friederike Eyssel[2], Robert Goldstone[3], Selma Šabanović[3]

1 – Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University
2 – Faculty of Psychology and Sports Science, Department of Psychology, Bielefeld University
3 – Indiana University Bloomington

Abstract

Studies in human-robot interaction (HRI) typically involve computational artifacts, i.e. the robotic system, as the subject of investigation. Thus, the reproducibility of any result in HRI studies directly relates to the reproducibility of this computational artifact in the first place. This has certain consequences for appropriate workflows that will be discussed in this chapter. We argue for a higher awareness, improved standards, and further automation of tool chains used to conduct robotic experiments. We identify this as a research topic in its own right, especially in cases where robotic systems are used in interdisciplinary research. This inherently includes that technically complex robotic experiments should also be reproducible by scientists with a *non-technical* background. We analyze and discuss a dedicated study by the CITEC Central Lab Facilities and an international team demonstrating that it is possible to replicate a relatively complex HRI experiment in two different laboratories across the globe by a research assistant with no experience in robotics at all.

Keywords

Human-Robot Interaction, Reproducibility, Robotic experiments

10.1 Introduction

The Central Lab Facilities (CLF) group of the Excellence Cluster Cognitive Interaction Technology (CITEC) at Bielefeld University aims to develop and improve technology, workflows, and tool chains for building as well as experimenting with interactive intelligent systems [1, 2, 3, 4, 5]. An important application and research field is human-robot interaction (HRI), which requires sophisticated robotic research platforms that include many software and hardware challenges besides the core areas of perception, behavior generation, and interaction design. Thus, research in HRI is a highly interdisciplinary endeavor. It aims to model the physical as well as mental dynamics between a human and a robot in a communicative or cooperative situation. It builds upon concepts and ideas from the area of human-human interaction in order to make the human-robot interface as smooth and intuitive as possible. Dealing with physically embodied agents, this includes many engineering issues towards flexible and save movements, many issues from machine perception, e.g. recognizing the interaction partner, many issues from artificial intelligence towards an interpretable and goal-oriented behavior of the robot, as well as many issues explored by the social sciences (psychology, linguistics, cognitive science, etc.) in order to understand associations, attributions, and expectations that humans have when interacting with a robot. Last but not least, any experiment with an autonomous robot includes many system engineering challenges including significant complexity issues on the software side which are frequently underestimated. Although there has been considerable progress in robot technology including available robotic standard platforms (e.g. iCub, Softbank's Nao and Pepper, Toyota's HSR), software frameworks [6, 7, 8, 9], and benchmarking activities [10, 11, 12, 13], the theoretical and practical foundations for experimental replicability of experiments in robotics is still in its infancy [14]. In this regard, Bonsignorio et al., e.g., states that *'even determining the information required to enable replication of results has been subject of extensive discussion'* [14].

In this chapter, we argue for a higher awareness, improved standards, and further automation of tool chains used to conduct robotic experiments. We identify this as a research topic in its own right, especially in cases where robotic systems are used in interdisciplinary research. This inherently includes the fact that technically complex robotic experiments should also be reproducible by scientists with a *non-technical* background. While this goes beyond the goals of Conquaire to reproduce the analytical part of an experiment only, in human-robotic interaction studies the replication of the technical settings is essential to understand the experimental results. The other Conquaire studies mostly deal with computational workflows and tools that are applied to datasets *after* these have been recorded in an experiment. Because most studies in, e.g., the natural sciences deal with 'natural' phenomena – i.e. they are not produced by an artificial artifact – the dataset can be interpreted with regard to this phenomenon at any place in the world. This is not the case for experiments including robots. The dataset can only be interpreted with regard to the specific artifacts used in the experiment. As a consequence, the reproducibility of an experiment and the validity of the data must include the possibility to reproduce also the robotic system and its behavior in the study.

In the following, we report our experiences and lessons learned in analysing a replication study conducted by the Central Lab Facilities involving a human-robot interaction (HRI) experiment in Bielefeld and at a partner site of the DFG Excellence Cluster CITEC within the DAAD Thematic Network Interactive Intelligent Systems. The study investigates an extended version of Stenzel et al.'s *'Joint Simon effects for non-human co-actors'* [15], in two labs in different institutions and continents. In psychology, the Joint Simon effect is used to investigate to what extent people mentally represent their own and other agent's actions in a joint task. This leads to delayed decision effects when a human is prompted with stimuli that are spatially incompatible with the roles in a team. The effect disappears when people think that they interact with a non-biological, technical artifact. Thus, it is an open question to which degree humanoid robots are perceived as social agents or team mates and if this can be shown using the Joint Simon effect (see Sec. 10.2.1 for more details).

To this end, the CLF researchers applied a novel software tool chain and methodology that implements state-of-the-art techniques with the objective of facilitating reproducibility in robotics research. The experiment was designed in cooperation between Bielefeld University and Indiana University Bloomington by a team of interdisciplinary scientists originating from psychology & brain sciences, informatics and robotics. The team initially conducted the study in Bielefeld before a replication attempt in Indiana was conducted. In this context, they specifically chose the following constraints in order to impose the same restrictions and obstacles encountered in 'regular' replication attempts:

1. The experiment must be replicated by a staff member who is *not* part of the research project.
2. The only starting point for replication is an online manual explaining our approach and the literature references therein.
3. Assistance from Bielefeld is only provided in otherwise irresolvable situations.

A replication of this experiment at different sites is an interesting case study from two different points of view. On the one hand, it is interesting to investigate whether there are cultural factors that affect the results. On the other hand, the setup includes a behavioral study with a robotic platform (the NAO robot), which is programmed to physically press a button where timing matters. Thus, from the perspective of reproducibility and the lessons learned from the Conquaire project, there are the following research questions: **(H1)** Is the tool chain and methodology been suitable to represent all aspects required for successful replication? **(H2)** What can we learn about reproducibility in general with respect to unexpected technical obstacles or situations one did not anticipate? **(H3)** Can the second study cross-validate the results obtained in the original Bielefeld study?

10.2 Experimental Settings and Methods

The following part of this contribution will cover the replication approach and the lessons learned. Important parts of the study and tool chain have been published previously [16, 17]. A final evaluation of the second study is still ongoing work. First, we will shortly introduce the theoretical background of the experiment. Then, we present the procedure and methods, and finally discuss our findings.

10.2.1 The JSE Experiment

The study was designed out to reproduce a variant [18] of a well-documented psychological effect, the Joint Simon Effect (JSE) [15]. The JSE describes a difference in reaction time depending on identity (*compatibility*) or disparity (*incompatibility*) of a stimulus' and the co-actors' spatial position in relation to the participant during a shared go/no-go task. The team aimed at reproducing this effect with a robot as co-actor as described in [18] and adopted the stimuli and procedure attributes. The original experiment was extended with a *robot position* condition to additionally test the influence of the robot's spatial relation to the human subject. While more detailed information about the JSE experiment can be found in the paper by Dolk et al. [18], we will briefly describe the experiment setup variant used in the particular study described in this chapter. Due to its wide distribution and availability, the team used the humanoid robot NAO as the participant's co-actor (Figure 10.1). The robot kneels next to the test subject on a table or chair. The barycenter of the robot is approximately at elbow height of a sitting subject.

The participant and the robot each have their own keyboard of identical type. The keyboards are directly adjacent and on the same level. During the experiment, stimuli, e.g. a square and a diamond, are displayed on a screen at randomized positions and in randomized order. Based on the initial assignment, either the robot or the human have to press the space-bar key as soon as the assigned stimulus appears. The corresponding reaction times (RT) of the human co-actor are measured.

In Bielefeld, the team tested 47 subjects from the nearby campus (M age = 24.61 years, SD age = 4.01 years). Each run consisted of 512 trials with short breaks per 128 trials and took approximately 30 minutes. The findings were similar to those found by Stenzel et al., the experiment showed a significant main effect of compatibility when analyzing the response times (RT), $F(1{,}48) = 11.639$, $p < 0.001$, partial $\eta^2 = .43$, indicating shorter RTs in compatible (423 ms) compared to incompatible trials (434 ms), which confirms the presence of an overall JSE. The team did not find a significant interaction between *compatibility* and *robot position*.

The data of the experiment were logged within the software tool jsPsych [19] that controlled the prompting, triggered the execution of robot movements, and recorded the reactions of the human participants (execution protocol of the experiment including timing events for prompting and robot, spatial configuration of prompts, etc.). The data is stored as comma-separated-value files which are preprocessed with documented shell commands and python scripts. Data analysis was conducted with SPSS[1] or R[2] tools.

10.2.2 Replication in Indiana

In order to reproduce the experiment in Indiana, under consideration of the demands and requirements in the current literature and the issues presented in section 10.1, there are two core issues to be solved:

1. A systemic solution for deployment, configuration, and integration of all necessary software artifacts.
2. A structured methodological 'how-to' for setup and execution considering user groups and tools from other disciplines, here, psychology.

This should not come as overhead for the replication of an experiment. It is essential that the replication tool chain is already in place and used when the first experiment is developed and conducted. Thus, the replicability of an experiment including software-intensive systems as core components has to be planned already when setting up the original experiment.

The replication tool chain

In order to address the above issues, the research team developed a software tool chain that has been explicitly designed to foster reproducibility of software intensive experiments in robotics — the *Cognitive Interaction Toolkit* (CITK) [3]. More detailed technical information is provided by Lier et al. [1, 2]. The requirement to support disciplinary tools to design and run experiments will be additionally covered by jsPsych [19].

At its core, the CITK provides a template-based "artifact-description" repository in order to pool and aggregate all required artifacts of a robotics experiment (cf. 10.2). There are basically two types of descriptions. The first is called

[1]https://www.ibm.com/de-de/products/spss-statistics
[2]https://www.r-project.org/
[3]https://toolkit.cit-ec.uni-bielefeld.de

recipe: it defines required system artifacts, e.g, software components, downloadable data sets, or system configuration files. Templates for new types of artifacts can be added on-the-fly by developers. With regard to pure software aspects, the existing set of templates contains macros for the most common build tools like autotools, maven, CMake, and ROS/catkin[4], enabling native builds of various kinds of software. These macros also help to remove redundancy and keep the recipes clean and well-structured. The second type is called distribution. A distribution is a composition of a number of arbitrary recipes and hence determines an entire system. Distributions, as well as recipes, mandatorily reference *versions*, e.g., tags, branches, or commit hashes of an artifact, such that a distribution reflects a *fixed* description of a system. Recipe and distribution files are publicly available in our Git-repository[5]. Another core-component is a prepackaged, i.e, download and run it, no configuration required, CI server. It is utilized to compile, deploy, and run entire software systems defined in distribution files. The server provides a web front-end that can be accessed via a browser for ease of use. In order to deploy and run a system, the CITK implements a generator-based approach. A so-called build-job-configurator tool automatically creates all required build-jobs (for every recipe in a distribution) on the server. A user merely selects the desired distribution file. Moreover, it is also possible to connect a physical robot to the machine that runs the CI server in order to control/actuate it. Lastly, our approach also provides a framework to automatically bring up (statefull execution), stop, and introspect a robotics software system. Executing a system merely requires to select and activate a designated build-job in the web front-end. Data that is acquired/logged during each system run is also stored on the server and accessible via web browser. By utilizing this part of our structured CITK approach, the team could ensure technical reproducibility of all required artifacts and also repeatable experiment execution regarding the software side of an experiment. An exemplary CITK tool chain demonstration video can be watched here: `https://vimeo.com/205541757` With respect to experiment design and orchestration, the study additionally made use of a framework called jsPsych. jsPsych is a JavaScript library for creating behavioral experiments in a web browser. It provides a description of the experiment structure in the form of a time line. It handles which trial to run next and storing the obtained data. jsPsych uses plugins to define what to execute at each point on the time line. The functionality of jsPsych was extended in order to i) trigger an experiment run on the CI server and ii) execute experiment-specific behaviors of the NAO/Pepper robots, e.g, based on the current state of the time line in jsPsych. Detailed information about jsPsych can be found in [19].

[4]`http://wiki.ros.org/catkin/conceptual_overview`
[5]`https://opensource.cit-ec.de/projects/citk`

The replication experiment

Due to the fact that the entire software system was already modeled using the CITK for the Bielefeld study [6], no additional work, besides the translation from German to English, e.g, in the jsPsych time-line slides was required. Hence, the software part including robot movement control interfaces, calibration procedures, and jsPsych experiment orchestration was already at hand. Since there was no prior knowledge about the (scientific) background of the staff member who would eventually replicate the experiment in Indiana, the team implemented a generic GUI-based application for all crucial technical steps with respect to the robot *hardware*, e.g, the calibration procedures. Finally, a detailed instruction was compiled on a public GitHub page (final version [7]). This online manual included the following steps:

1. Introduction
2. Hardware Requirements and Prerequisites
3. Software Requirements and Prerequisites
4. Physical Experiment Setup
5. Subjects
6. Executing the Experiment
7. Results
8. Literature

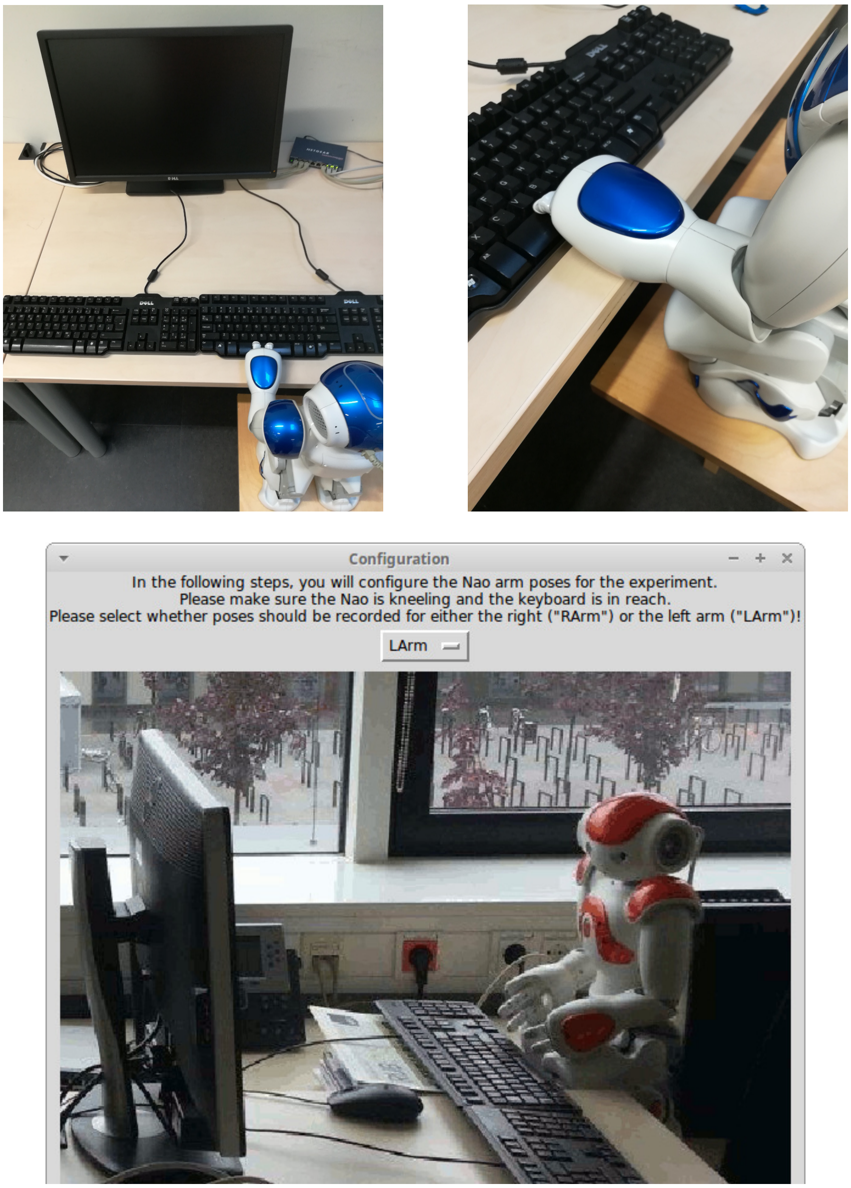
In summary, the manual included the following content:

- a brief introduction to the research topic and study goals, plus references to related literature,
- a specification of the required hardware, e.g, a NAO robot acquired within 2-3 years,
- a PC or laptop including CPU and RAM specifications including the size, resolution and refresh rate of the utilized screens,
- a specification of the operating system requirements, i.e., Ubuntu Xenial (16.04, 64 bit),
- an explanation of how to setup the physical experiment, such as height and position of the robot, position of the keyboards, monitors, etc., and

[6]`http://www.webcitation.org/6xlwomECk`
[7]`https://Git.io/vAxml`

- a brief explanation of the network setup.

Moreover, the document included detailed instructions about the installation and usage of the CITK in order to deploy the software system, calibrate the robot, and run the experiment. The instructions also provided information about the subjects, the welcome and actual experiment procedure. Lastly, it was explained how to obtain and inspect the gathered data. So far, the documentation included detailed information with respect to technical (soft- and hardware), as well as methodological/procedural aspects, to reproduce the study as it was conducted in Bielefeld. The team also established a communication channel via instant messaging using Slack. The channel was intended to provide 'emergency support', but only in case of an otherwise irresolvable situation. Hence, the chat history could also be exploited for post-experiment analysis, if required.

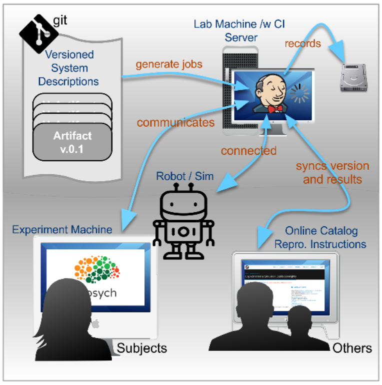


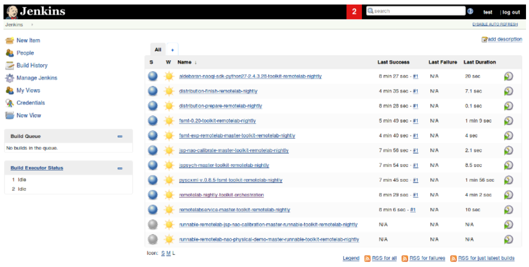
Configuration

In the following steps, you will configure the Nao arm poses for the experiment.
Please make sure the Nao is kneeling and the keyboard is in reach.
Please select whether poses should be recorded for either the right ("RArm") or the left arm ("LArm")!

LArm

Figure 10.1: Top left: The NAO JSE setup used in one of our Bielefeld setups; Top right: NAO keypress pose; Bottom: Screenshot of the robot calibration GUI




Jenkins 2 search test | log out

Jenkins › DISABLE AUTO REFRESH

New Item
People
Build History
Manage Jenkins
My Views
Credentials
New View

Build Queue
No builds in the queue.

Build Executor Status
1 Idle
2 Idle

add description

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration	
●	☀	aldebaran-naoqi-sdk-python27-2.4.3.28-toolkit-remotelab-nightly	8 min 27 sec - #1	N/A	20 sec	
●	☀	distribution-finish-remotelab-nightly	4 min 35 sec - #1	N/A	7.1 sec	
●	☀	distribution-prepare-remotelab-nightly	8 min 28 sec - #1	N/A	0.1 sec	
●	☀	fsmt-0.20-toolkit-remotelab-nightly	5 min 49 sec - #1	N/A	1 min 9 sec	
●	☀	fsmt-exp-remotelab-master-toolkit-remotelab-nightly	4 min 40 sec - #1	N/A	4 sec	
●	☀	jsp-nao-calibrate-master-toolkit-remotelab-nightly	7 min 56 sec - #1	N/A	2.1 sec	
●	☀	jspsych-master-toolkit-remotelab-nightly	7 min 54 sec - #1	N/A	8.5 sec	
●	☀	pyscxml-v.0.8.5-fsmt-toolkit-remotelab-nightly	7 min 45 sec - #1	N/A	1 min 56 sec	
●	☀	remotelab-nightly-toolkit-orchestration	8 min 29 sec - #1	N/A	4 min 2 sec	
●	☀	remotelabservice-master-toolkit-remotelab-nightly	8 min 6 sec - #1	N/A	10 sec	
○	☀	runnable-remotelab-jsp-nao-calibration-master-runnable-toolkit-remotelab-nightly	N/A	N/A	N/A	
○	☀	runnable-remotelab-nao-physical-demo-master-runnable-toolkit-remotelab-nightly	N/A	N/A	N/A	

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Figure 10.2: Toolchain for the replication of robotics experiments: An experiment for which the associated computational artifacts are documented in a browsable online catalog which is linked to the corresponding repositories. From here a researcher is instructed to automatically roll out the system distribution using a continuous integration server which is linked to the robot platform as well as the computer controlling the behavioral experiment using JsPsych. Left: Overview of the toolchain, Right: Screenshot of the CI server web front-end. Each row corresponds to a recipe that has been translated into a build job. Build jobs can be activated by selecting the most right icon (stopwatch). Execution of an experiment can also be triggered using this front-end.

10.3 Analytical Reproducibility: Results & Lessons Learned

We report on the lessons learned in a time line based manner. Depending on the reader's background, either in computer science or the humanities for instance, some of the reported obstacles may appear 'trivial'. However, we claim that it is crucial to raise awareness for false assumptions made by domain experts, e.g., with regard to common knowledge about specific technological or methodological aspects of an experiment, which are by far not so obvious/common for others outside their domain. Furthermore, we would like to point out that the reported observations are based on a *practical interdisciplinary replication attempt*, which is especially valuable in order to learn about all *the different characteristics* and challenges concerning replicability of robotics systems.

The replication attempt of this JSE experiment was conducted by a research assistant (RA) with a background in psychology. With respect to interdisciplinary research this was, on the one hand, an almost ideal scenario, on the other hand however, a technically-challenging one as well.

10.3.1 Technical Obstacles & Procedural Issues

The following issues were reported during the replication study in Indiana. The research assistant started with a plain laptop. Thus, the first issue was reported shortly after the study officially started. Even though the deployment of the required software components (using the CITK) was successful on first attempt, the RA faced a couple of issues with the installation routine of Ubuntu. The team in Bielefeld could resolve these issues by pointing the RA to the correct Ubuntu documentation. The second technical issue was reported a few days later. The operating system as well as the robot software environment were already installed successfully. Nonetheless, during the required robot calibration procedure, a connection to the robot could not be established via local network. The team in Bielefeld resolved this issue by instantly updating the online manual for the network setup which is also hosted in the linked repositories. The third issue was reported after a first test run of the experiment. So far, the entire software system was deployed, the robot calibrated, and also the physical experiment setup was in place. However, during the run, the RA noticed that the translation of two single lines of text on a slide in the jsPsych time line was missing. The team in Bielefeld could resolve this issue by correcting the error in the code base and subsequently updating the Git repository. In Indiana, the RA just had to re-trigger the corresponding build job, thus automatically installing the updated version of the experiment. The fourth and last reported obstacle occurred in an early stage of the actual experiment. Since the tool chain allows to download and inspect already gathered data via web browser, the colleagues in Indiana soon took a first look at intermediate results. They noticed that the distribution of the participants' position with respect to the robot indicated a strong preference towards only *one* side. The team in Bielefeld discovered that the instructions provided for the experimenter in jsPsych, addressing the procedure of subject positioning, were not as precise as they should have been. This could be corrected by updating the description in the repository.

10.3.2 Results of the Pilot Study on Reproducibility in HRI

At time of writing, the JSE experiment in Indiana has been finished; first results show a weaker but observable Joint Simon effect. However, besides the obstacles already discussed, there were several positive lessons learned. It is very difficult, if not impossible, to foresee all pitfalls faced by the researcher replicating the experiment. As a local solution or patch does not solve the issue in a consistent manner, a flexible tool chain is required that allows for almost instant patching and deployment of experiment artifacts. In this regard, the technical complexity of the deployed robotic system (hard- and software) was completely hidden to the research assistant (RA) in Indiana. The time required to setup the entire software system was limited to a few hours, including the installation of an operating system. Moreover, the acceptance threshold and usability of the CITK tool chain appeared to be positive, given the fact that it was easily usable by the RA. Also, the transition from design, implementation and execution of the experiment in Bielefeld to the deployment in Indiana merely required sending a link to an online manual. In a short post-experiment interview we asked the RA for a self-assessment regarding the experience with Linux-based systems, robotic hardware, robotic software, the Linux network stack, conducting HRI experiments, and conducting psychology experiments. In summary, the RA was reasonably experienced in conducting experiments in psychology and, having used Linux before, knew a few basic Linux commands. Regarding the remaining topics, the RA was an inexperienced user, i.e, had never operated a single robot before.

10.4 Analysis of reproducibility experiment

In this section, we discuss lessons learned from our cross-site replication study of a robotic study.

Reproducibility is decided at development time: We would like to stress that without having the tool chain in place at the development and preparation time of the study in Bielefeld, a replication study at Indiana would have been extremely time consuming if not impossible. Thus, any tool chain used for the replication of results should be established in the daily workflow of the researchers understanding it as a *development tool* instead of a replication tool.

Experimental protocols: Besides the technical requirements and issues involved in replicating studies and their scientific results, it is also important to neatly document the experimental protocol. Typically, this is solved by workflows, policies, and tools within the specific discipline without being integrated in the technical framework of a robotic experiment. In the study reported, a tool from psychology was integrated for experimental control. This is also a prerequisite for a systematic logging of all experimental data. However, we can observe in the study that the non-technical aspects of the experimental protocol were not sufficiently described, which raised several questions when intermediate results were analysed and discussed. Thus, there is still an open issue to more formally describe the experimental protocols.

Scientists with a non-technical background: An interdisciplinary field like HRI involves experts from different backgrounds. Reproducibility should not depend on having a robotic expert on-site. Even though the current approach demonstrated that *it is possible*, even by an inexperienced user, these first obstacles were not even close to what a robotics engineer would consider 'an obstacle'. On this account, we deem this lesson learned even more valuable. Furthermore, these kind of 'low-level' obstacles can be easily mitigated by providing detailed *beginner-level instructions*.

Automated documentation roll-out: It appears extremely useful to be able to quickly and dynamically alter instructions provided for replication attempts if errors are reported/discovered. SCM-based repositories, not only for source code, but also for this kind of manuals seem to be a well-applicable solution. Further, adding replication instructions to the corresponding source code of a publication is not labor intensive at all.

Report intermediate results: The issues and obstacles discussed before imply that it is important to automate the collection and evaluation of (also) intermediate results to prevent subsequent failures, especially if data acquisition is time-consuming. Thus, the requirement for an analytic reproducibility also applies to intermediate results. In the case of the experiment considered here, all preprocessing steps and scripts were precisely documented. Further, the 'R'-toolbox can be used as an open source alternative to SPSS for the statistical analysis of the data.

Open issues: Regarding the limitations of the approach presented, the toolbox currently does not incorporate any standardized benchmark procedures for more general HRI experiments. It does not provide any tool or template support for metrics (if existent/agreed-upon) with respect to comparability of HRI systems. We are open for discussion and welcome contributions concerning this topic.

Final remarks: In this contribution we discussed and analyzed a dedicated study on the replication of a reasonably complex HRI experiment in two different laboratories across the globe without a) flying experts in and b) making a single video/phone call — by a research assistant with *a non-technical background* and no experience in robotics at all. We reported on the lessons learned during this practical replication process.

10.5 Conclusion

This chapter has shown that it is possible to reproduce a robotic experiment at different sites, reproducing the same effect. The chapter has presented a workflow that provides end-to-end support for researchers wanting to reproduce a certain experiment. In this particular case, the workflow was based on the CITK toolkit developed at CITEC. In the particular case, the experiment involved a reproduction of the well-known Joint Simon effect known from psychology research. Using the end-to-end experimental workflow described in this chapter it was possible for a psychologist from Indiana University not expert in robotics to reproduce an experiment originally carried out at Bielefeld University. We regard this as a clear success story of experimental reproducibility and see this as a best practice of reproducibility.

References

[1] Florian Lier, Marc Hanheide, Lorenzo Natale, Simon Schulz, Jonathan Weisz, Sven Wachsmuth, and Sebastian Wrede. Towards Automated System and Experiment Reproduction in Robotics. In Wolfram Burgard, editor, *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016.

[2] Florian Lier, Johannes Wienke, Arne Nordmann, Sven Wachsmuth, and Sebastian Wrede. The cognitive interaction toolkit–improving reproducibility of robotic systems experiments. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 400–411. Springer, 2014.

[3] Florian Lier, Ingo Lütkebohle, and Sven Wachsmuth. Towards automated execution and evaluation of simulated prototype HRI experiments. Proc. 2014 ACM/IEEE Int. Conf. on Human-robot interaction, pages 230–231. ACM, 2014.

[4] Severin Lemaignan, Marc Hanheide, Michael Karg, Harmish Khambhaita, Lars Kunze, Florian Lier, Ingo Luetkebohle, and Gregoire Milliez. Simulation and hri recent perspectives with the morse simulator. LNAI Lecture Notes in Artificial Intelligence. Springer, 2014.

[5] S. Meyer zu Borgsen, P. Renner, F. Lier, T. Pfeiffer, and S. Wachsmuth. Improving human-robot handover research by mixed reality techniques. In *Proceedings of VAM-HRI 2018. The Inaugural International Workshop on Virtual, Augmented and Mixed Reality for Human-Robot Interaction*, 2018.

[6] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[7] Herman Bruyninckx. Open robot control software: the orocos project. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 3, pages 2523–2528. IEEE, 2001.

[8] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. Yarp: yet another robot platform. *International Journal on Advanced Robotics Systems*, 3(1):43–48, 2006.

[9] Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*, pages 46–51. IEEE, 2009.

[10] Sven Wachsmuth, Dirk Holz, Maja Rudinac, and Javier Ruiz del Solar. Robocup@home - benchmarking domestic service robots. In *AAAI*, 2015.

[11] Pedro U. Lima, Daniele Nardi, Gerhard K. Kraetzschmar, Rainer Bischoff, and Matteo Matteucci. Rockin and the european robotics league: Building on robocup best practices to promote robot competitions in europe. In Sven Behnke, Raymond Sheh, Sanem Sarıel, and Daniel D. Lee, editors, *RoboCup 2016: Robot World Cup XX*, pages 181–192, Cham, 2017. Springer International Publishing.

[12] F Amigoni, A Bonarini, G Fontana, M Matteucci, and V Schiaffonati. Benchmarking through competitions. In *European Robotics Forum–Workshop on Robot Competitions: Benchmarking, Technology Transfer, and Education*, 2013.

[13] Ana Huaman Quispe, Heni Ben Amor, and Henrik Christensen. A taxonomy of benchmark tasks for robot manipulation. In *Robotics Research*, pages 405–421. 01 2018.

[14] Fabio Bonsignorio. Reproducible research in robotics: Current status and road ahead. `http://www.heronrobots.com/EuronGEMSig/gem-sig-events/icra2017workshoprrr`, May 2017. (Accessed on 06/03/2018).

[15] Anna Stenzel, Eris Chinellato, Maria A Tirado Bou, Ángel P del Pobil, Markus Lappe, and Roman Liepelt. When humanoid robots become human-like interaction partners: corepresentation of robotic actions. *Journal of Experimental Psychology: Human Perception and Performance*, 38(5):1073, 2012.

[16] Florian Lier, Phillip Lücking, Joshua R. de Leeuw, Sven Wachsmuth, Selma Sabanovic, and Robert Goldstone. Can we reproduce it ? toward the implementation of good experimental methodology in interdisciplinary robotics research. In *ICRA 2017 Workshop on Reproducible Research in Robotics: Current Status and Road Ahead*, 2017.

[17] P. Lücking, F. Lier, J. Bernotat, S. Wachsmuth, S. Sabanovic, and F. A. Eyssel. Geographically distributed deployment of reproducible hri experiments in an interdisciplinary research context. pages 181–182, Chicago, IL,USA, 2018. ACM.

[18] Thomas Dolk, Bernhard Hommel, Lorenza S Colzato, Simone Schütz-Bosbach, Wolfgang Prinz, and Roman Liepelt. The joint simon effect: a review and theoretical integration. *Frontiers in Psychology*, 5, 2014.

[19] Joshua R De Leeuw. jspsych: A javascript library for creating behavioral experiments in a web browser. *Behavior research methods*, 47(1):1–12, 2015.

11 | Conclusion

The DFG-funded Conquaire project has been concerned with investigating the feasibility of reproducing the analytical phase of research in experimental sciences. We have conducted eight case studies in various areas such as biology, linguistics, psychology, robotics, economics and chemistry as a basis to understand obstacles and best practices towards ensuring reproducibility of scientific results.

The reproduction of analyses still involves substantial effort. Originally, we had set ourselves the goal to invest a full working week (40 hours) into the reproduction of each of these case studies. In many cases, the time needed to reproduce a result has exceeded this amount by a factor of three. The reason is that, in many cases, while data and scripts were available, the documentation was not sufficient to reproduce the analyses without step-by-step guidance of the authors of the original publication that we set out to reproduce.

In addition to the effort devoted to the reproduction itself, the Conquaire project has performed a number of workshops with all the researches from the eight use cases to introduce them to the goals of the project, to introduce Git, etc.

As a conclusion, we can say that the success rate for reproduction was very high. We were able to reproduce the results within all case studies. Yet, the level of reproducibility was not the same for all project. According to the taxonomy of levels of reproducibility introduction in chapter 1, we have one clear case of full analytical reproducibility and three further cases that reached the category of full analytical reproducibility by the end of the project after recoding analytical workflows using open and free programming languages. Four case studies have the status of *at least* limited reproducibility as the reproduction of their work (still) involves obtaining third-party commercial licenses for tool. It requires a minimal further investment to bring these cases into the level of full analytical reproducibility. This is a clear success in our view, clearly showing that analytical reproducibility is feasible.

The main obstacles for analytical reproducibility found were i) the lack of documentation and thus reliance on guidance by the original authors, ii) the reliance on some manual steps in the analytical workflow (e.g. clicking on a GUI) , iii) the reliance on non-open and commercial software, and iv) lack of information about which particular version of software and/or data was used to generate a specific result.

An institutional policy and infrastructure can alleviate most of the problems mentioned above. Our experience shows that using a distributed version control system is a best practice to be followed and a basic step towards reproducibility. Our experience shows that scientists in any field can quickly learn to work with Git, in particular if GUIs such as GitLab are provided. Most of the scientists involved in case studies in Conquaire had no issues in uploading their data to a Git repository.

Our experience also shows that scientists are deeply motivated to make their results reproducible, even if this leads to a level of exposure that might lead to errors being discovered. In some cases we discovered minor errors in plots, scripts etc., and the involved scientists were more than happy to correct these minor issues. The exposure and independent validation brings benefits that are generally appreciated. This is indeed an important conclusion from Conquaire. At the beginning of the project we were sceptic how openly scientists would be willing to make their research artifacts available and support reproduction. At the end of the project we can corroborate that there is a strong culture within science of being as open as possible to ensure external scrutiny or validation of scientific results. Our experience has been positive thus and we would like to encourage research organizations world-wide in setting up policies encouraging their researchers to make their results analytically reproducible. On the basis of the results of Conquaire, Bielefeld University is working towards the establishment of policies in this respect.

We would like to end this book with a number of clear recommendations to research institutions wanting to support their scientists in making their results reproducible:

- **Organization-wide version control system:** Rolling out an organization-wide version control system is the basis for reproducibility. It makes transparent when and by whom data and scripts were collected or created and allows to uniquely reference a particular version of the data and code. Such a system can also support persistent storage of data and has a back-up function for the researchers. We recommend using Git.

- **Committing scripts before data collection:** When using a Git repository, our recommendation is to develop policies that encourage scientists to commit their analytical scripts before they collect data. After committing scripts, dummy data could be committed to check that the script works and produces the results on an independent server that is not under control of the scientist. After data collection, the data can be committed and the results generated automatically on the server in a continuous integration like manner. This reduces possibilities for tampering with data to produce a desired result or at least makes post-data-collection modifications transparent.

- **Creating incentives for providing documentation:** Organizations should create incentives to foster documentation of datasets, analytical workflows and adopt and enforce standards for describing author metadata, licensing information, etc.

- Independent code execution / result validation: Organizations should implement services and infrastructure that supports the independent execution of software / code to reproduce a certain result. Continuous integration servers fulfill this purpose.
- **Gamification:** Principles of gamification might create incentives for ensuring high quality of data. We have positive experience with introducing a badge system. Yet, more investigation and experimentation is needed here.
- **Open software:** We clearly recommend to set up policies that encourage researchers to rely on open, free and non-commercial software to facilitate reproduction of results on independent machines without the need to install commercial software and pay high license fees.
- **Metadata:** Organizations should train and support researchers in creating high-quality metadata for their data and also train them in selecting and specifying under which licenses their data can be used. Consulting on data exploitation and re-use while taking into account privacy aspects is crucial. Bielefeld university has created a center for research data management with the mission of consulting and training researchers on such dimensions.

However, the most important lesson learned is that analytical reproducibility should not be considered as an afterthought and delayed to the end of a research project. Analytical reproducibility is easy to achieve if one designs experiments and software environments from the start with the goal to make analytical workflows executable on any server by a third party. This minimizes efforts needed as workflows are not disrupted in the middle of a project and minimizes the opportunity to post-modify data and results, thus creating transparency. Applying continuous integration principles from the start and taking into account data quality and publishing data and scripts early in the research process as well as specifying tests that monitor data quality and run analytical workflows independently of the researchers carrying out the research as well as publishing results continuously and transparently in some repository is an effective way of fostering analytical reproducibility.

This book presents the results of the Conquaire (Continuous Quality Control for Research Data to Ensure Reproducibility) project. The goal of the project was to investigate if principles of continuous quality control and test-driven development can be applied to the management of research data to increase its quality and potential for re-use.

We have been working closely together with researchers from different disciplines ranging from biology through chemistry, economics, linguistics, psychology to computer science and robotics. Within eight case studies we have aimed at reproducing one central part of a previously published research article, focusing on what we call "analytical reproducibility", that is, reproducing the computational evaluation of an experiment.

The work conducted in the case studies has provided us with a detailed understanding of the analytical workflows used by all the case study partners as well as with a deep understanding of barriers and challenges to reproduction. The book concludes with recommendations and lessons learned from the practical attempt to reproduce a number of published results.

ISBN 978-3-9433-6306-7