Cimiano P., Pietsch C., & Wiljes C. (Eds.) (2021). Studies in Analytical Reproducibility: the Conquaire Project. Bielefeld. https://doi.org/10.4119/unibi/2942780

# 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 $\mu$m. 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/`

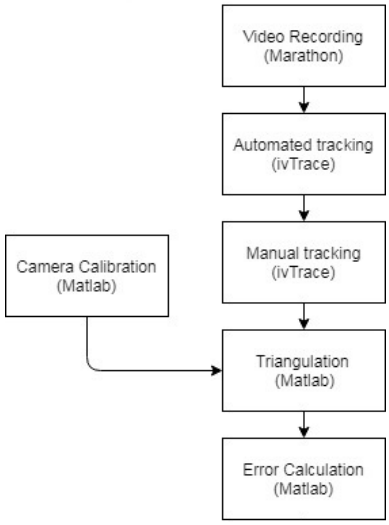


Figure 4.1: Procedure to calculate the trajectories of bumblebee flights, original procedure as described in Lobecke et al. [1]

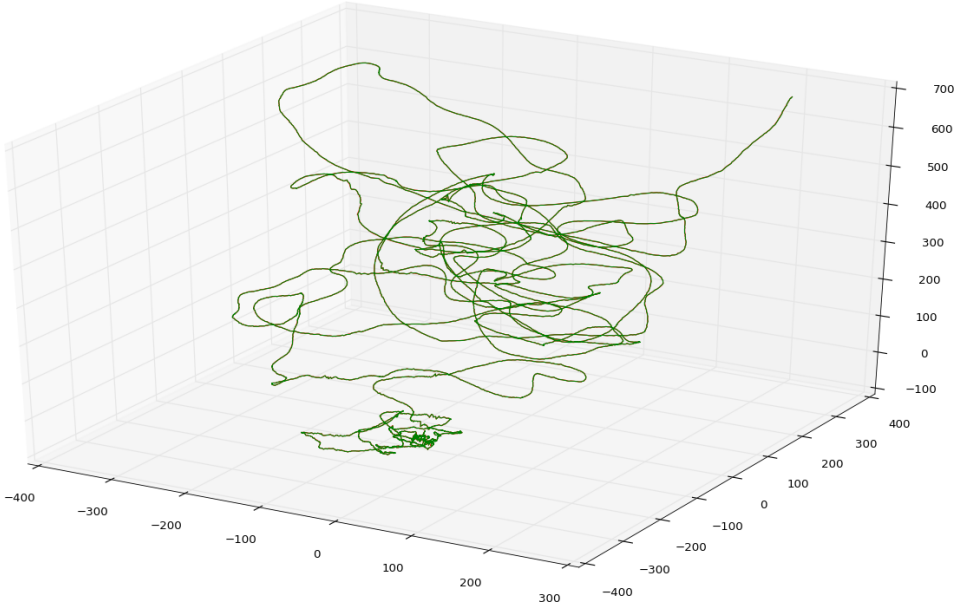


Figure 4.2: Example trajectory of a bumblebee flight, seen in 3D (cf. Lobecke et al. [1])

## 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`

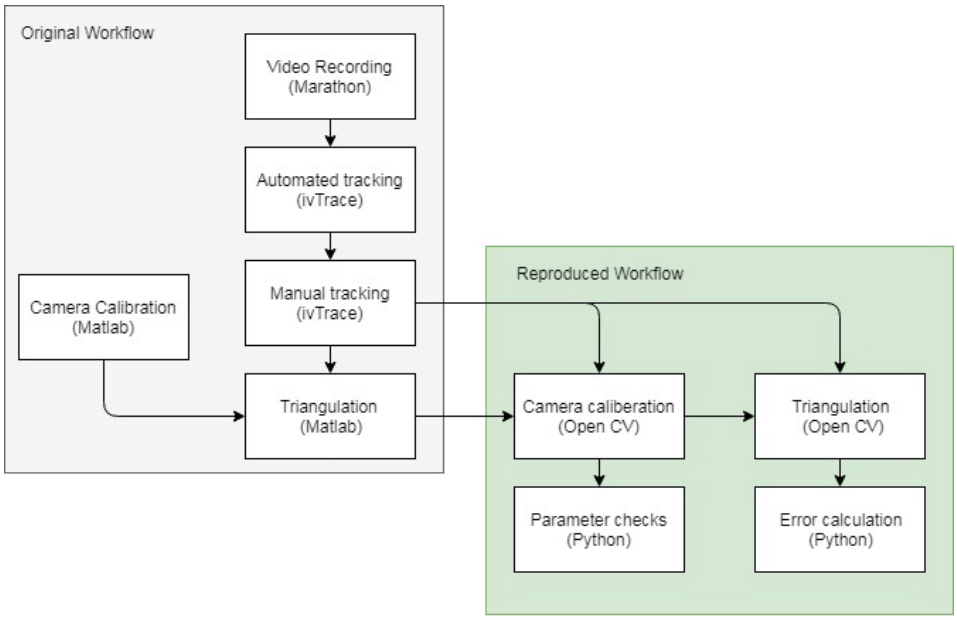


Figure 4.3: Procedure to obtain the trajectories of bumblebees. In shaded gray: The original procedure followed in Lobecke et al. [1]. In shaded green: the reproduced and adapted procedure. In parenthesis, the software/tools used to accomplish the task.

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.

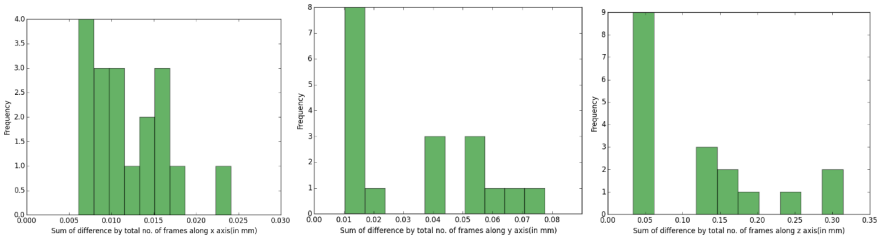


Figure 4.5: Distribution of differences between original MATLAB and new Python calculation for the three dimensions, x, y and z respectively

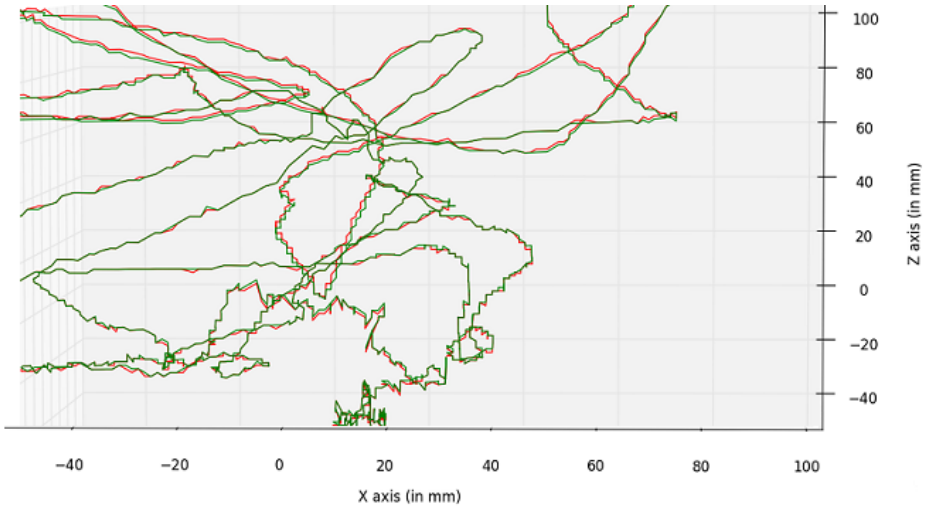


Figure 4.4: A close magnified snapshot displaying comparing the 2D projections to x- and z-axes of the 3D trajectories computed by the MATLAB analytical workflow (red) and the Python-based workflow (green).

### 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

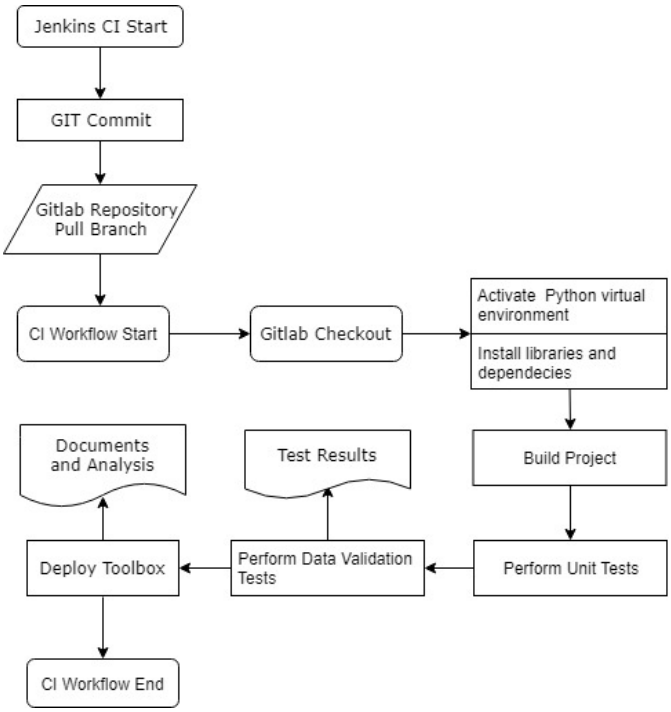


Figure 4.6: Flow Chart of Jenkins Continuous Integration pipeline.

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.

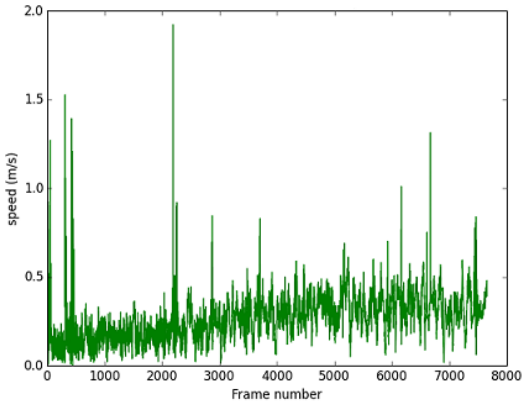


Figure 4.9: Flight speed for a bee which is well within the range as defined by the researcher (below 10m/s).

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.

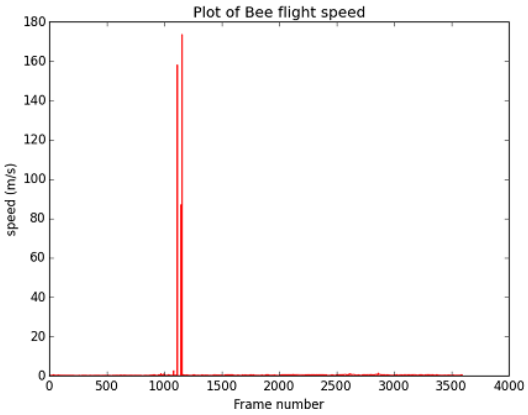


Figure 4.10: Flight speed for a bee in which an error in the data was found. The erroneous speed was above 150 m/s, which is far outside the acceptable range.

## 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.