Freie Universität Berlin
Institut für Informatik
AG Künstliche Intelligenz und Robotik, Biorobotics Lab

Bachelorarbeit

Yuriy Marchenko
yuriy.marchenko@gmail.com
Matrikelnummer: 4493904

Sichere und performante Methoden für den Datentransfer und die Datenhaltung von biologischen Massendaten für die Verhaltensanalyse von Honigbienenkolonien

Betreuer: Dr. Tim Landgraf, Gutachter: Dr. Tim Landgraf, Prof. Dr. Raúl Rojas

Berlin, 1.02.2016

Zusammenfassung

Im Rahmen des Projekts BeesBook wurden über 370 TB an Bildmaterial von Honigbienenkolonien auf dem Supercomputer des HLRN (Norddeutscher Verbund für Hoch- und Höchstleistungsrechnen) ausgewertet. Diese Arbeit entwickelt eine Software, die die Ergebnisse von dem Supercomputer in eine Datenbank überfuhrt. Damit die Daten schnell in Datenbank gespeichert werden können, wurde den gesamten Ablauf in kleinere und von einander unabhängige Komponente zerlegt. Das ermöglicht eine parallele Ausführung aller Komponente. Des Weiteren wurde die Datenbank zur Verhaltensanalyse von Honigbienenkolonien konzipiert.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder Ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 1.02.2016

Yuriy Marchenko

Inhaltsverzeichnis

1 Einleitung

Die Arbeit ist im Rahmen des Projektes Beesbook [1] in der Biorobotics Lab der Intelligenten Systeme und Robotik AG an der Freie Universität Berlin enstanden.

1.1 Bienen und ihre Kommunikation

Seit langem ist es schon bekannt, dass Bienen mit Hilfe des Bienentanzes kommunizieren [3]. Durch das Tanzen wird Information über Futterquellen vermittelt. Je nach Entfernung der Futterquelle, über welche eine Biene die anderen vermitelln will, wird sie durch zwei verschiedene Tänze angegeben. Für Quellen, die bis ca.100 m von dem Bienenstock sich befinden, wird hauptsächlich der Rundtanz getantzt. Für alle weiter entfernt liegenden Quellen verwenden die Bienen den Schwänzeltanz. Durch die Tänze wird die Richtung und die Entfernung der Futterquelle angegeben.

1.2 Das Projekt Beesbook

Das Ziel des Projektes ist Automatisierung der Überwachung und Analyse des Sozialverhaltens einer Honigbienenkolonie. Am Ende sollten folgende Fragen beantwortet werden wie z.B: bilden Bienen innerhalb eines Bienenstocks Gruppen? Bevorziehen Bienen bei Trophallaxis oder Tanzen bestimmte Bienen? Um Überwachung des Bienevolkes zu automatisieren wurden ca. 2000 Bienen markiert und über Monate mit 4 Kameras abfotografiert.

Abbildung 1: Design der Markierung [1]. Schwarze und weiße Felder bedeuten entsprechend 0 und 1. Die von 0 bis 11 Felder represäntiren eindeutigen binären Kode der Markierung. 12 und 13 dienen zur Orientirung der Markierung und damit die Ausrichtung der Biene.

Pro Kamera wurden von 3 bis 4 Bilder pro Sekunde aufgennomen. Alle Bilder wurden wegen enormen benötigten Speicherplatz auf den Supercomputer des HLRN[1] (Norddeutscher Verbund für Hoch- und Höchstleistungsrechnen) hochgeladen. Da es sich bei dem Computer um ein Cray-System[2] handelt, wird der Supercomputer in der Arbeit hier als Cray bezeichnet. Anschlissend wurden die Ergebnisse mit Hilfe der in der Biorobotics Lab entwickelte Software berechnet. Die wird kurz Pipeline genannt und besteht aus fünf Pipeline-Stufen: Preprocessor, Localizer, Ellipse Fitter, Grid Fitter und Decoder. Dabei wird pro Bild eine csv-Datei generiert, die die Information über Position, Winkel, Bienenummer sowie auch Wahrscheinlichkeitswert einzigen Detektionen enthält.

1.3 Aufbau der Arbeit

Nach dieser Einleitung folgt die Vorstellung der Problemstellung. Danach, in den Sektionen 3 und 4, werden die in der Arbeit entwickelte Lösung zur Speicherung der Ergebnisse in Datenbank, sowie das Design der Datenbank, beschrieben. Die letzte Sektion behandelt die Diskussion und geben einen Ausblick, was besser gemacht werden kann.

[1] **https://www.hlrn.de**.
[2] **http://www.cray.com/**.

2 Problemstellung

Im Saison 2014 wurden insgesamt 65 Millionen Bilder aufgenommen [1]. Das ergibt 65 Millionen Ergebnisdateien, die vom Cray heruntergeladen und für Postprocessing in Datenbank gespeichert werden müssen. Christian Tietz hat in seiner Masterarbeit [6] bereits angefangen, sich mit dem Thema der Übertragung von Ergebnisse zu befassen. Er beschreibt Herunterladen der einzige Ergebnisdatei mit Hilfe von standard Unix-Befehle *ls* und *scp*. Anschließend muss jede Datei geparst werden und in Datenbank geschrieben. Das gesamte Ablauf pro Datei dauert etwa *200 ms*. Damit ist es unmöglich 14 csv-Dateien (4 Kameras mit 3.5 Bilder pro Sekunde) in Echtzeit zu importieren. Der Ansatz des Skriptes von Christian Tietz ist gut, jedoch nicht für das Experiment verwendbar. Die Ergebnisdateien müssen in Echtzeit oder schneller in Datenbank importiert werden. Christian Tietz hat in seiner Masterarbeit auch Datenbank entworfen. Das Format der Ergebnisdateien wurde später geändert, deswegen muss Datenbank angepasst und erweitert werden.

3 Herunterladen der Ergebnisdateien

Diese Sektion beschreibt die Anforderungen, Funktionsweise, Aufbau und Implementierung eines Systems zum Herunterladen der Ergebnisse aus dem Pipeline.

3.1 Anforderungen

Ein System zum Ergebnisdateien Speicherung muss folgende Anforderungen erfüllen:

- **Die Ergebnisdateien müssen schnell in Datenbank importiert werden.** In den ersten zwei Saisons (2014 und 2015) wurden die Bilder auf dem Cray nicht sofort ausgewertet. Es liegt daran, dass das Software ständig optimiert und weiterentwickelt wird. Man braucht auch gewisse Zeit um die beste Parameter für die Auswertung zu finden. Erst wenn alle Probleme behoben sind, werden die Bilder auf dem Cray durchgerechnet. Postprocessing kann erst dann angefangen werden, wenn die Ergebnisdateien in Datenbank vorhanden sind. Um die Wartezeit zu minimieren muss Herunterladen der Ergebnisdateien schnell erledigt werden.

- **Datenverlust vermeiden.** Bei so eine Menge von Daten ist es sehr umständlich herauszufinden an welche Stelle es zum Datenverlust kam. Jeder Datenverlust beeinflusst die Ergebnisse des gesamten Projektes. Es muss sichergestellt werden, dass beim dem Ablauf es nicht zu Datenverlust kommen kann.

- **Protokollierung.** Alle Probleme oder Fehler während des Vorgang müssen protokolliert werden, damit man sie sofort erkennen und beheben kann. Das dient auch dazu, die Probleme nachträglich zu erkennen.

- **Automatisierung.** Der Ablauf vom Herunterladen und in Datenbank einfügen soll automatisch ablaufen.

3.2 Gesamte Ablauf des Herunterladen der Ergebnisdateien

Um den gesamten Ablauf zu beschleunigen, werden die Ergebnisdateien nicht einzeln heruntergeladen, sondern es werden mehrere Dateien in einem Archiv gepackt. Der gesamte Ablauf kann man in 4 unabhängige von einander Teile unterteilen:

- Cray. Dazu gehört Ergebnisdateien finden, in Archiv packen und herunterladen.
- Archiv. Alle von dem Cray heruntergeladene Archive müssen nun entpackt werden.
- Parser. Die csv-Dateien werden in sql-Befehle umgewandelt.
- Datenbank. Führt von Parser bereitgestellte sql-Befehle aus.

Dadurch dass alle 4 Prozesse von einander unabhängig sind, können sie parallel ausgeführt werden. Das Programm wurde in C++ geschrieben. Dieses Programm ruft andere Programme zum herunterladen, archivieren und übertragen der Dateien.

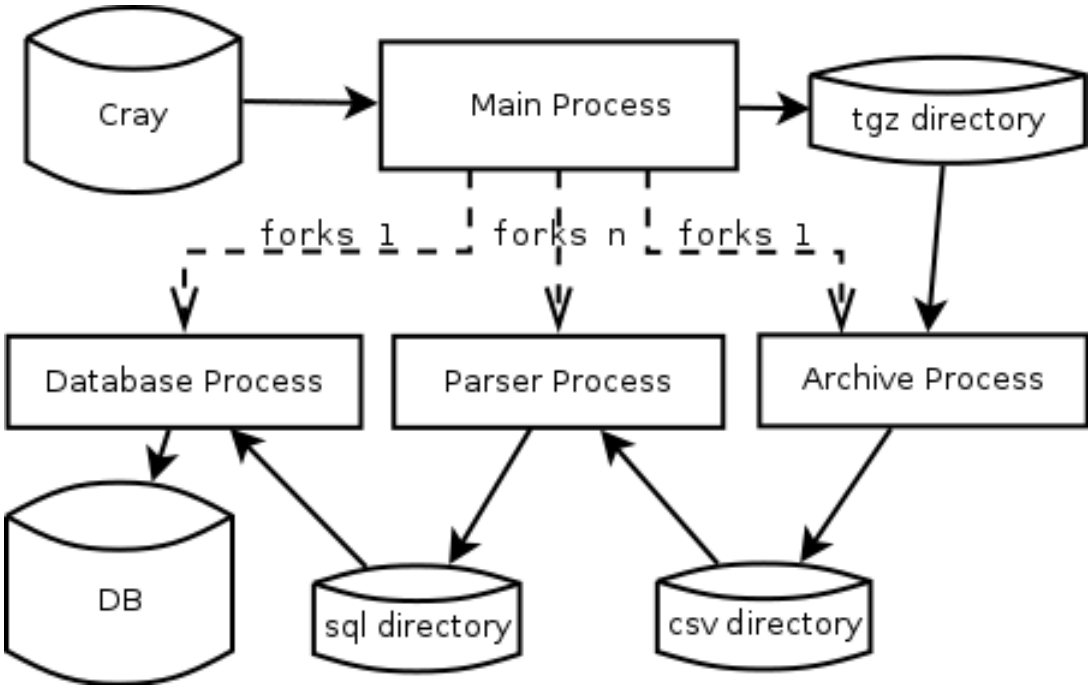

Abbildung 2: Datenfluss.

Diese Abbildung zeigt Datenfluss von Cray nach Datenbank. Zunächst werden jeweils einen Archiv und einen Datenbank Prozesse gestartet. Die Anzahl von Parser Prozesse ist einstellbar und hängt hauptsächlich von der Anzahl der CPU-Kerne, sowie auch I/O-Leistung des Rechners ab. Es soll höchstens so viele Parser Prozesse gestartet werden, wie die Anzahl von CPU-Kerne.

Main Prozess, falls auf dem Cray Ergebnisdateien vorhanden sind, lädt sie in lokale tgz-Ordner herunter. Archiv Prozess holt in einer Schleife die Liste von komplett heruntergeladene Archive und entpackt die in csv-Ordner. Die Parser und Datenbank Prozesse funktionieren ähnlich wie der Archiv Prozess, indem sie Dateien in entsprechend csv- und sql-Ordner regelmäßig abfragen.

3.3 Main Prozess

Die Ergebnisdateien liegen in tar-Archive auf dem Cray. Jeder solche Archiv erhält 256 csv-Dateien. Die csv-Dateien alle einzeln auf dem Cray zu speichern ist wegen Anzahl der Dateien in Dateisystem Einschränkung nicht möglich.

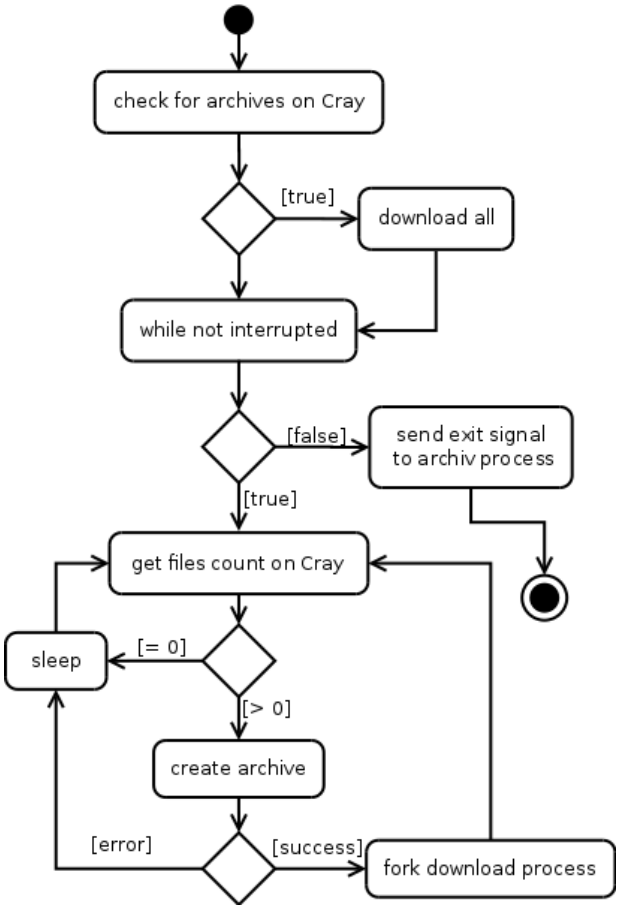

Abbildung 3: Main Prozess Workflow

Diese Abbildung zeigt die Funktionsweise des Main Prozesses, dessen Aufgabe ist Ergebnisdateien von dem Cray herunterzuladen. Dafür wird zunächst geprüft, ob es noch Dateien, die heruntergeladen werden müssen, gibt. Falls ja, wird ein Archiv angelegt.

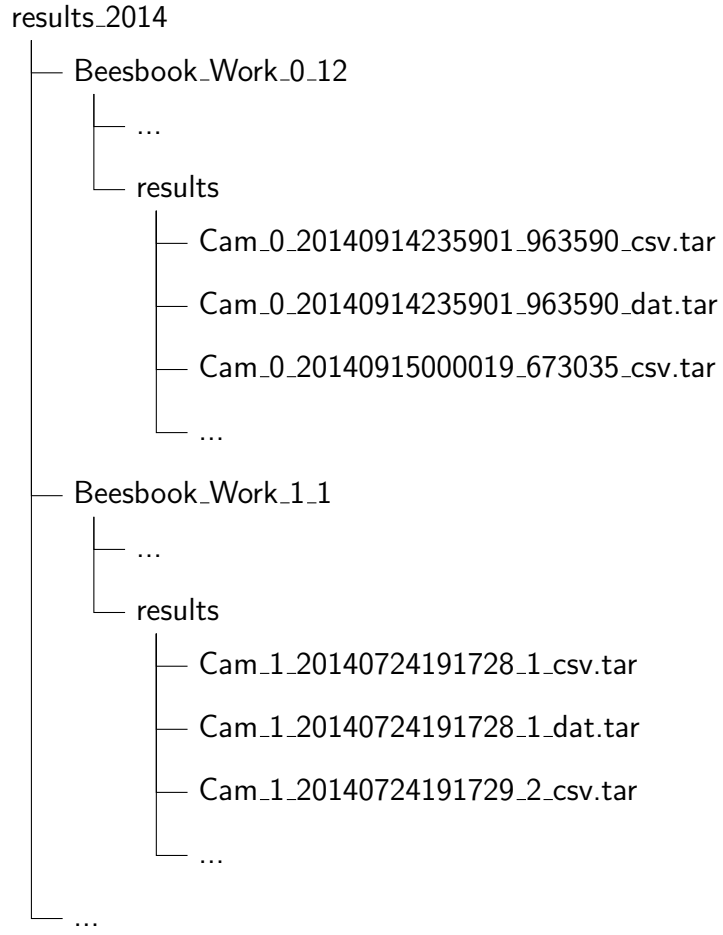

Abbildung 4: Ordnerstruktur der Ergebnisdateien auf dem Cray

Damit es sichergestellt werden kann, dass während des Herunterladen der Ergebnisdateien es nicht zum Dateiverlust führt, dürfen die Ergebnisdateien (*_csv.tar) nicht verändert werden. Die Dateien dürfen nicht umbenannt oder verschoben werden. Damit man zwischen Dateien die schon heruntergeladen wurden und die noch nicht unterscheiden kann, wird eine Dateiliste erstellt. Alle tar-Datei, die schon bearbeitet wurden, werden in diese Liste gespeichert. Da es bei dem Cray um ein Unix System handelt, kann man standard Unix-Befehle verwenden. Hier folgt ein Beispiel:

```
ssh user@blogin.hlrn.de 'cd /gfs1/work/bebesook/
    results_2014/ && find Beesbook_Work_0_12  -iname "*_csv
    .tar" -type f 2> /dev/null | sort | grep -Fxvf $WORK2/
    done_list | head -n 20 > $WORK2/list && tar -T $WORK2/
    listN -czf $WORK2/bb_archN.tgz && mv $WORK2/bb_archN.
    tgz $WORK2/bb_done_archN.tgz && cat $WORK2/listN >>
    $WORK2/done_list ; rm $WORK2/listN && echo ok'
```

- Um die Ausführung des *find* Befehls zu beschleunigen wird es zuerst in dem ersten Ordner (ggf. *Beesbook_Work_0_12*) nach Dateien deren Name mit _csv.tar endet gesucht. Alle Ordner werden nacheinander bearbeitet.
- Die Liste aller Dateien wird danach sortiert.
- Es wird eine Datei *$WORK2/listN* angelegt und die ersten 20 Dateieinnamen, die noch nicht in *$WORK2/done_list* sind, aus der sortierte Liste geschrieben. N ist die aktuelle Nummer für Archiven. Sie wird einfach hochgezählt.
- Die Dateien aus der *$WORK2/listN* werden in *$WORK2/bb_archN.tgz* Archiv gepackt.
- Falls Archiv erfolgreich angelegt wurde, wird er nun umbenannt.
- Anschließend werden Namen aller Dateien, die gerade in Archiv gepackt wurden zu der Liste *$WORK2/done_list* hinzugefügt.
- Falls alle Schritte fehlerfrei ausgeführt wurden, wird ok auf der Standardausgabe ausgegeben.

Falls es in einem Schritt Fehler gab, wird dieser Fehler auf der Standardausgabe ausgegeben und protokolliert. Jetzt kann Archiv heruntergeladen werden. Main Prozess forkt einen Kind, das den Archiv herunterlädt. Die Archivdatei wird mit *scp* Befehl wie folgt heruntergeladen:

```
scp user@blogin.hlrn.de:$WORK2/bb_done_archN.tgz archive/
```

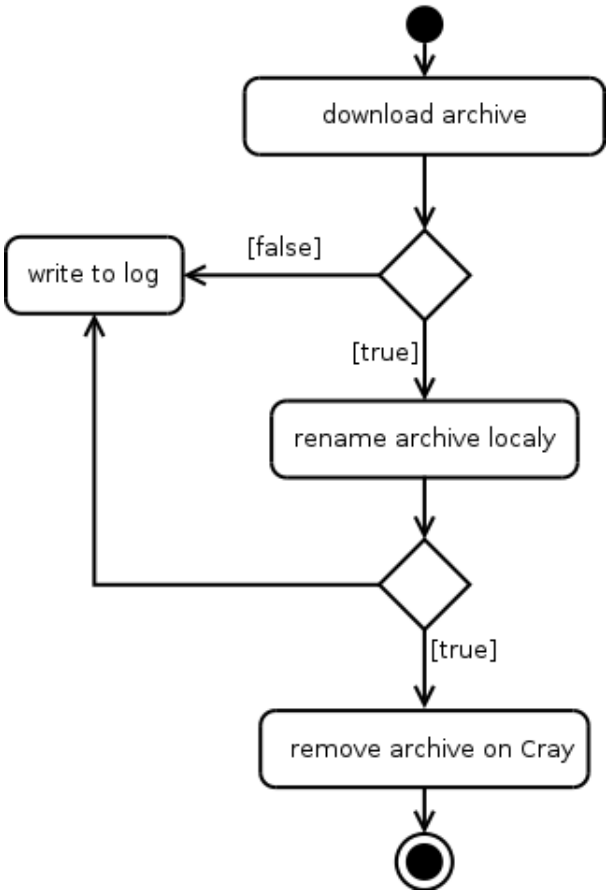

Abbildung 5: Download Prozess Workflow

Sobald ein Archiv fertig heruntergeladen ist, wird er nach *bb_completedN.tgz* umbenannt um den Archiv Prozess zu signalisieren, dass das Herunterladen erfolgreich abgeschlossen ist. Main Prozess wird normalerweise mit Strg+C beendet. Die Signale SIGINT und SIGTERM werden abgefangen und die globale Variable *interrupted* wird auf true gesetzt. Der Prozess wird nicht sofort beendet, sondern es wird gewartet bis aktuelle Aufgabe vollständig abgeschlossen ist. Das gilt auch für alle Kind-Prozesse. Alle Prozesse können erst nach einige Minuten tatsächlich beendet werden. Das Signal SIGKILL (kill -9) kann nicht abgefangen werden und kann zu Datenverlust führen.

3.4 Archiv Prozess

Archiv Prozess wartet auf die Archive und mit Hilfe von *tar* Befehl entpackt die csv-Dateien.

```
tar --to-command='tar -C"tmp" -xf -' -xzf bb_done_archN.
    tgz --strip-components=2
```

Trotz der csv-Dateien verschachtelt archiviert sind (zuerst als tar-Datei und dann als tgz Datei), werden alle csv-Dateien in *tmp* Ordner einzeln entpackt.

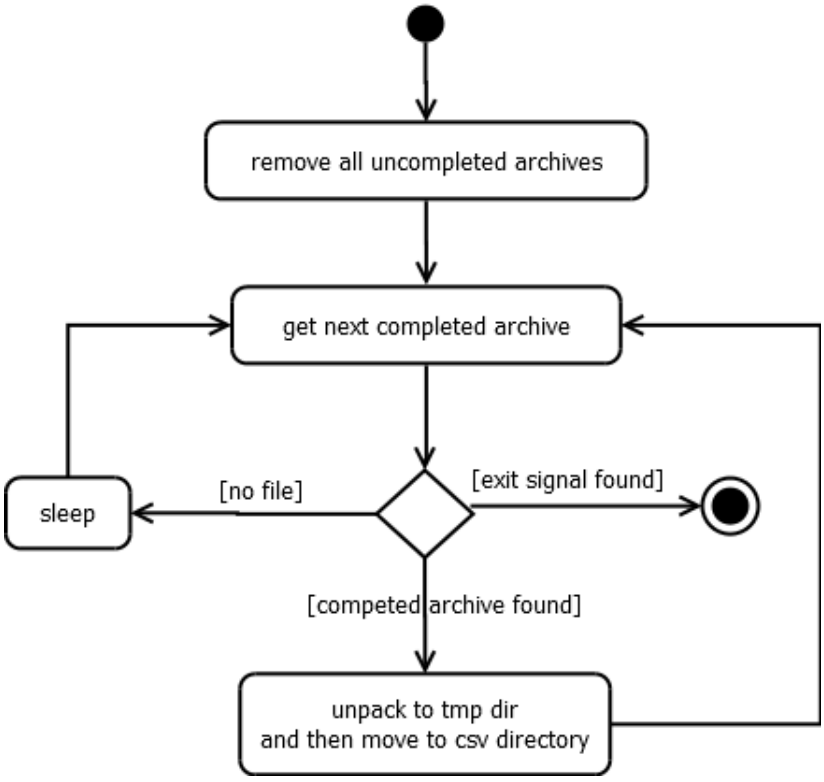

Abbildung 6: Archiv Prozess Workflow

Archiv Prozess ist als bash-Skript implementiert. Wenn Main Prozess beendet wird, wird eine Datei in archiv-Ordner angelegt. Wenn der Archiv Prozess die Datei findet, löscht er die Datei und beendet sich.

3.5 Parser Prozess

Parser Prozess wandelt csv-Datei in gültige sql-Befehle. Da es mehrere Parser Prozesse parallel ausgeführt werden können, müssen die Prozesse die csv-Dateien teilen. Jeder Prozess kriegt eine Nummer. Erster Prozess kriegt eine 0, zweiter eine 1 usw. Die Ergebnisdateien sind nach folgendem Schema ge-

nannt: Cam_0_20140915001422_718255.csv. 0 steht für Kamera, 2014/09/15 00:14:22 ist der Zeitstempel und 718255 sind die Mikrosekunden. Jeder Prozess entscheidet, ob er diese Datei parsen soll, indem er *(Sekunde % ProzessAnzahl) == PozessNummer* prüft. Dabei wird es davon ausgegangen, dass die Sekunden in Ergebnisdateien Namen gleichverteilt sind. Es wird eine sql-Datei für jede Minute erzeugt. Da es mehrere Parser Prozesse in die gleiche sql-Datei schreiben können, müssen sie synchronisiert werden. Dafür wird atomare Operation *mkdir* verwendet [4].

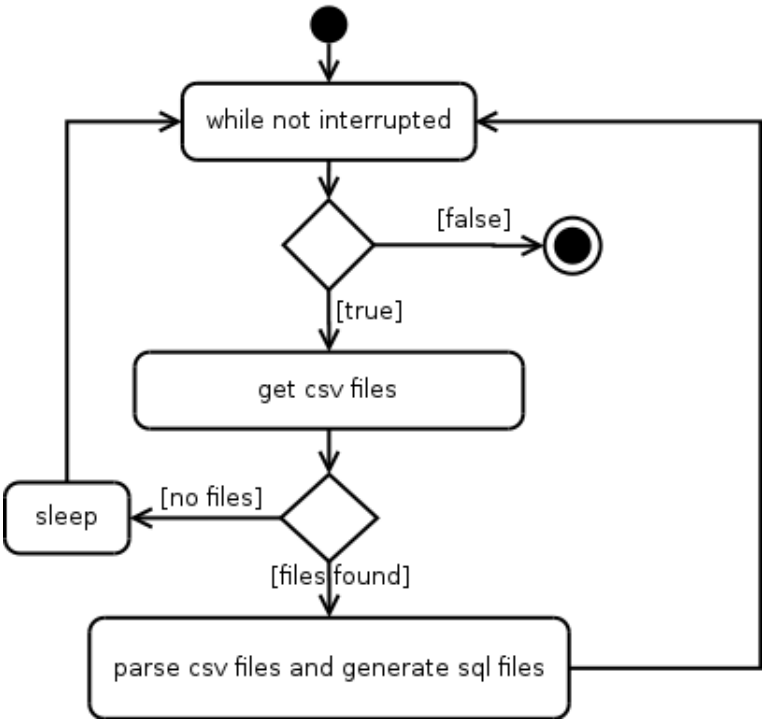

Abbildung 7: Parser Prozess Workflow

3.6 Datenbank Prozess

Datenbank Prozess überwacht den sql-Ordner und sucht nach sql-Datei. Jede sql-Datei wird einzeln eingelesen und ausgeführt. Der Zugriff auf die Datenbank erfolgt über die von Postgres mitgelieferte API (libpg-fe.h)[3]. Der Zugriff auf sql-Datei wird mit Parser Prozesse auch mit Hilfe von *mkdir* Befehl synchronisiert. Tritt bei der Ausführung einer sql-Datei einen Fehler, wird die

[3] **http://www.postgresql.org/docs/current/static/libpq-build.html**.

Fehlerbeschreibung mit Hilfe von *PQerrorMessage* Funktion empfangen und protokolliert. Die sql-Datei wird in error-Ordner verschoben.

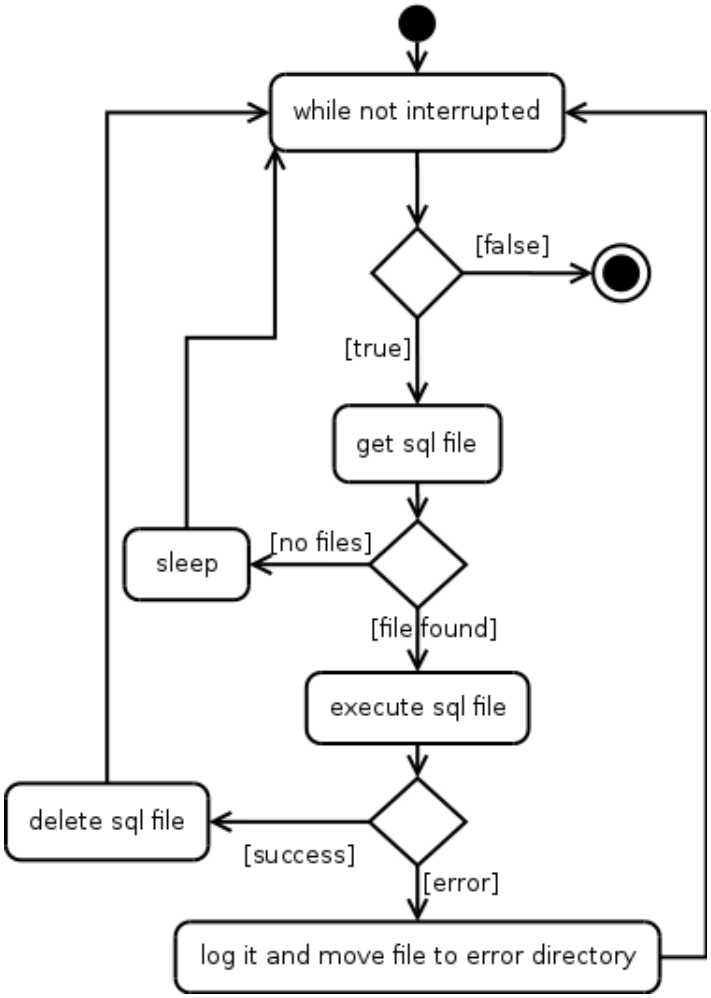

Abbildung 8: Datenbank Prozess Workflow

4 Datenbank

Diese Sektion beschreibt die Änderungen, die in Datenbank Design gemacht werden müssen, um die Ergebnisdateien im neuen Format zu speichern. Hier werden auch weitere Tabelle zum Tracken von Bienenpfaden entworfen.

4.1 Ergebnisdateien Format

Ergebnisdateien erhalten folgende Information:

- **tagIdx**. Eindeutige sequenzielle Nummer der Markierung
- **candidateIdx**. Sequenzielle Nummer des Kandidaten pro Markierung
- **gridIdx**. Sequenzielle Nummer des Gitters / Decodierung pro Kandidat
- **xpos**. X-Koordinate der Rastermitte
- **ypos**. Y-Koordinate der Rastermitte
- **xRotation**. Drehung des Gitters in X-Ebene
- **yRotation**. Drehung des Gitters in Y-Ebene
- **zRotation**. Drehung des Gitters in Z-Ebene
- **vote**. Kandidatenpunktzahl
- **id**. Dekodierte Bienennummer

Es kann pro Markierung bis zu 3 verschiedene Kandidaten und pro Kandidat bis zu 3 Gitter geben. Insgesamt sind es pro Markierung maximal 9 mögliche Bienennummern. Um alle Daten optimal zu speichern, wurde folgende Datenbank Schema entworfen:

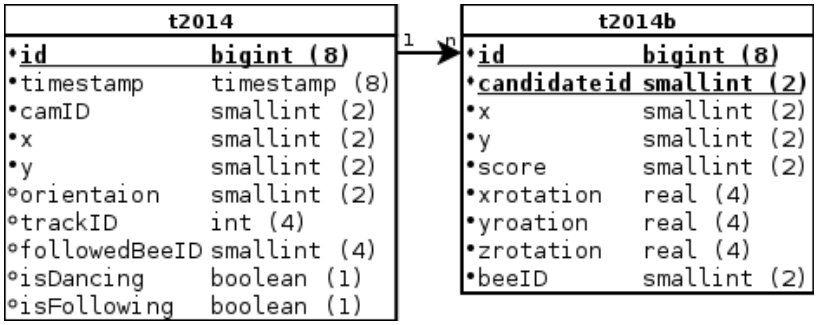

Abbildung 9: Datenbank Schema. Die Zahlen in Klammern sind die Länge der Datentypen in Bytes.

Id ist der Primärschlüssel der t2014 Tabelle. Ursprünglich in der Arbeit von Christian Tietz, bestand Primärschlüssel aus timestamp, CamID, x und y. Der neu Primärschlüssel wurde hinzugefügt, um die Referenzierung der Einträge zu optimieren. In der 2014b Tabelle muss man somit nur ein Feld id (8 Byte) speichern. Der Primärschlüssel wird folgendermaßen bitweise zusammengestellt:

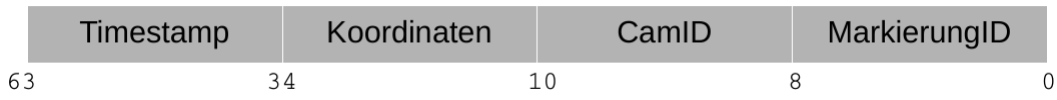

Abbildung 10: Aufbau des Primärschlüssels. MarkierungID ist tagIdx modulo 255. Timestamp sind Zentisekunden seit dem Experiment Beginn.

Die Reihenfolge der Felder in Tabellen spielt eine große Rolle. Die eigentlichen Nutzdaten (Spalten der Zeile) muss immer ein Vielfaches der MAXALIGN Abstand für die Plattform sein [5]. MAXALIGN Abstand ist für die meisten 64-Bit Systems 8 Byte lang. Zur Veranschaulichung sei folgendes Beispiel betrachtet:

- Tabelle A hat 3 Felder: b1 boolean (1 Byte), t1 timestamp (8 Bytes) und b2 boolean (1 Byte). Pro Zeile wird es 24 Bytes verbraucht.
- Tabelle B hat die gleiche 3 Felder: t1 timestamp (8 Bytes), b1 boolean (1 Byte) und b2 boolean (1 Byte). Pro Zeile wird es nur 16 Bytes verbraucht.

4.2 Trajektorien

Die erste Aufgabe des Postprocessing ist Bewegungspfade der Bienen zu erstellen. Das sind die Grunddaten für weitere Analysen des Sozialverhalten der Bienen. Da der Decoder für jede Detektion mehrere mögliche Bienennummern liefert, müssen Lücken und Fehler überbrückt werden [2]. Jeder Pfad besteht aus eine ausgefilterte Bienennummer und zu dem Pfad gehörigen Detektionen. Für jeden gefundenen zusammenhängenden Bewegungspfad einer Biene, wird zuerst die ausgefilterte Bienennummer in die *Tracks* Tabelle eingefügt. Im 2. Schritt müssen alle beteiligte Detektionen in *t2014* Tabelle aktualisiert werden, indem die Pfadnummer in trackID Feld geschrieben wird. In die *tracks* Tabelle werden somit alle gefundene Bewegungspfade gespeichert. Um die Struktur der Datenbank zu vereinfachen, kann man Ergebnisse nicht in Minuten Tabellen, sondern in eine große Tabelle speichern. Die Anfragen werden zwar langsamer ausgeführt, aber immer noch genügend für

den Tracking Algorithmus. Die Idee dabei ist, dass man die Daten parallel zu der Algorithmus Ausführung anfragt. Nachdem die *tracks* Tabelle befüllt ist, wird für jede Biene eine Tabelle erstellt. Diese Tabelle wird *trackBeeX* genannt, wobei X ist die ausgefilterte Bienennummer, und enthält alle Teilpfade einer Biene. Die Tabellen werden mit Hilfe des Skripts erzeugt und dienen dazu, spätere Analysen zu vereinfachen und Zugriffszeiten zu minimieren.

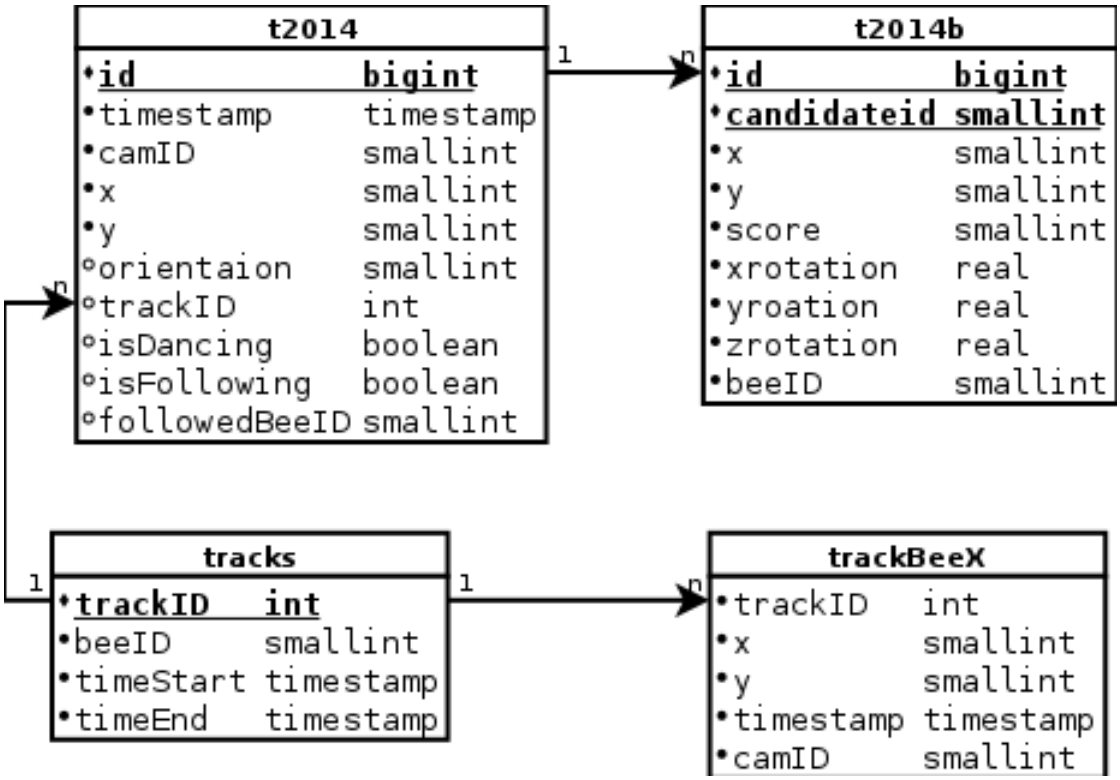

Abbildung 11: Gesamte Datenbank Schema

5 Diskussion und Ausblick

Ziel ersten Teils dieser Arbeit war es, die Ergebnisse vom Cray in eine Datenbank zu überführen. Die Schwerpunkte sind Sicherheit im Sinne, dass die Daten nicht verloren gehen und Geschwindigkeit. Die Daten sollen mindestens in Echtzeit übertragen werden. Es wurde nicht geschafft die gesamte Ergebnisdateien in Datenbank zu überfuhren. Die Grund dafür ist fehlende Datenbankserver mit ausreichend Kapazität. Es wurden die Ergebnisse von ersten 14 Tagen aus dem Saison 2014 von nur eine Kamera 0, sowie einige Tagen aus dem Saison 2015 in lokale Datenbank gespeichert. Es wurde damit, den gesamten Ablauf getestet. Es hat etwa einen Tag gedauert, bis die Daten in Datenbank überfuhrt wurden. Die Anforderung, dass die Daten mindestens in Echtzeit übertragen werden müssen, ist damit erfüllt. Die allgemeine Verbesserungen an der Software für das Herunterladen der Ergebnisdateien betreffen meist die Lesbarkeit und Wartbarkeit des Codes. Parser Prozess entscheidet welche csv-Datei er bearbeiten soll, anhand der Minute in Datei Name. Es wird dabei momentan davon ausgegangen, dass die Minuten gleichverteilt sind. Es wäre eventuell besser, wenn man eine hash-Funktion verwendet, die die Gleichverteilung der csv-Datei für alle Parser Prozesse gewährleisten kann. Eine weitere Verbesserung wäre zusätzliche Benachrichtigung, beim Auftreten eines Fehlers, an der Administrator zu schicken. Momentan werden alle wichtige Ereignisse Protokolliert und es ist nötig, dass jemanden ab und zu die Log-Datei sich anschaut um Probleme fest zu stellen.

Im zweiten Teil dieser Arbeit wurde Datenbankdesign für das neue Ergebnisformat angepasst und erweitert. Die benötige Speicherplatz für Tabellen wurden optimiert, indem die Reihenfolge der Felder geändert wurde. Weitere Tabellen für Speicherung der Trajektorien wurden entworfen. Es wurde auch geprüft, ob man die Ergebnisse in eine Tabelle speichern kann. Damit der Tracking Algorithmus nicht durch langsame Datenbank Abfragen verlangsamt, könnte man die notwendige Daten (z.b alle Zeitstempel) vorher in eine temporäre Tabelle speichern. Den gesamten Speicherbedarf für eine Saison lässt sich nicht einfach einschätzen. Es wird ca. 10 GB für einen Tag für Kamera 0 benötigt. Wenn man die Bilder vom Experiment anschaut, sieht man, dass am Ende des Experiments nur noch wenige Bienen eine Markierung tragen. Daraus folgt, dass es deutlich weniger Einträge gibt als zu Beginn. Es ist auch zu beachten, dass es mehr Bienen bzw. Markierungen auf den Bildern, die mit Kamera 0 gemacht wurden, zu sehen sind. 2400 GB (10 GB * 4 Kamera * 60 Tage) ist die obere Grenze. In der Wirklichkeit wird es sicherlich deutlich weniger Platz für Datenbank gebraucht.

Literaturverzeichnis

[1] *Automatic methods for long-term tracking and the detection and decoding of communication dances in honeybees*. 25. Sep. 2015. URL: http://journal.frontiersin.org/article/10.3389/fevo.2015.00103/full.

[2] Ronja Deisel. "Modellierung eines probabilistischen Verfahrens zum Tracken von Bienenpfaden und Analyse der zugrunde liegenden Daten". Bachelor's thesis. Freie Universität Berlin, 2015.

[3] Dr. Karl von Frisch. *Tanzsprache und Orientierung der Bienen*. Springer, 1965. ISBN: 978-3-642-94917-3.

[4] *Lock your script*. URL: http://wiki.bash-hackers.org/howto/mutex.

[5] *PostgreSQL Database Page Layout*. URL: http://www.postgresql.org/docs/current/static/storage-page-layout.html.

[6] Christian Tietz. "Entwurf und Implementierung einer Speicherarchitektur für Bildmassendaten zur Verhaltensanalyse von Honigbienenkolonien". Master's thesis. Freie Universität Berlin, 2015.