Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Ole Schweim

Entwicklung eines Tools für die automatisierte Korrelation von TESS Tracklets mit der NASA Horizons Datenbank

Fakultät Technik und Informatik
Studiendepartment Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and electrical engineering

Ole Schweim

Entwicklung eines Tools für die automatisierte Korrelation von TESS Tracklets mit der NASA Horizons Datenbank

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Elektrotechnik und Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Dr. Marcel Völschow
Zweitgutachter: Prof. Dr. Klaus Jünemann

Eingereicht am: 09. Juli 2024

Ole Schweim

Thema der Arbeit
Entwicklung eines Tools für die automatisierte Korrelation von TESS Tracklets mit der NASA Horizons Datenbank

Stichworte
Korrelation, Datenverarbeitung, Benutzeroberfläche

Kurzzusammenfassung
In dieser Bachelorarbeit wird ein Tool entwickelt, welches automatisiert Übereinstimmungen von gesichteten Objekten mit der National Aeronautics and Space Administration (NASA)-Datenbank ermittelt. Dabei werden verdächtige Objekte über ein bestehendes Programm ermittelt, welches Bilder aus dem Transiting Exoplanet Survey Satellite (TESS) auswertet. Diese Daten müssen dann auf Zusammenhänge untersucht und gefiltert werden. Zu den gefilterten Daten ermittelt das Tool dann passende Objekte aus der NASA-Datenbank und zeigt diese an. In der Oberfläche des Tools soll der Benutzer dann passende Objekte finden, entfernen und eventuell unbekannte Objekte identifizieren können.

Title of the paper
Development of a tool for the automated correlation of TESS tracklets with the NASA Horizons database

Keywords
Correlation, data processing, user interface

Abstract
In this bachelor thesis, a tool is developed that automatically identifies matches between sighted objects and the NASA database. Suspicious objects are identified using an existing programme that evaluates images from the TESS. This data must then be analysed and filtered for correlations. The tool then determines matching objects from the NASA database for the filtered data and displays them. The user should then be able to find and remove matching objects in the tool's interface and identify any unknown objects.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungen

API Application Programming Interface

TESS Transiting Exoplanet Survey Satellite

NASA National Aeronautics and Space Administration

UTC Coordinated Universal Time

RA Rektaszension

DEC Deklination

GUI Graphical User Interface

JD Julianisches Datum

1 Einführung

Im Rahmen der Bachelorarbeit von Elin Thomfohrde-Dammann wurde ein Programm zur Auswertung von TESS Bildern erstellt.[1] Dabei werden die Bilder auf Veränderungen durch Helligkeitsunterschiede untersucht. Diese Helligkeitsunterschiede weisen auf Objekte hin, die das einfallende Licht blockiert haben. Das Programm gibt einen Datensatz aus, in dem diese auffälligen Punkte mit Orts- und Zeitstempel gespeichert sind.
Diese Daten können auf bisher unbekannte Objekte in unserem Sonnensystem hinweisen. Aber auch bereits bekannte Objekte werden mit diesen Verfahren erkannt. Daher ist es wichtig, diese Daten auszuwerten und mit bereits bekannten Daten zu vergleichen. So kann festgestellt werden, ob durch die Bildauswertung neue Objekte gefunden wurden.
Diese auffälligen Veränderungen in den Bildern können dann weiter untersucht werden, um gegebenenfalls neue Objekte zu finden, ihre Umlaufbahn zu bestimmen und sie zu kategorisieren.

1.1 Ziele

Ziel dieser Bachelorarbeit ist es, ein Programm zu entwickeln, das automatisch Korrelationen zwischen den Daten des Bildverarbeitungsprogramms, das die Bilder auswertet, und den von der NASA zur Verfügung gestellten Daten findet. Dazu müssen zunächst die Daten des Bildverarbeitungsprogramms analysiert und gefiltert werden.
Anschließend sollen von der NASA über deren Application Programming Interface (API) Daten angefordert werden, die zur gleichen Zeit im gleichen Koordinatenausschnitt zu sehen waren. Die beiden Datensätze sollen dann auf Korrelationen untersucht werden. Übereinstimmende Objekte sollen identifiziert werden können, um dann im besten Fall noch unbekannte Objekte zu entdecken.
Dazu soll im Rahmen dieser Bachelorarbeit zusätzlich eine GUI programmiert werden, die es einem Benutzer[1] ermöglicht, die notwendigen Arbeitsschritte durchzuführen, um die Datensätze des Bildverarbeitungsprogramms nach unbekannten Objekten zu durchsuchen. Dabei

[1] In dieser Bachelorarbeit sind mit der maskulinen Form alle Geschlechter gemeint

soll es möglich sein, die Datensätze einzubinden, Daten bei der NASA anzufordern und diese zu bearbeiten.

1.2 Struktur der Bachelorarbeit

Diese Bachelorarbeit ist wie folgt aufgebaut:
Zunächst werden in Kapitel 2 die Grundlagen zum Bildverarbeitungsprogramm, die Eigenschaften der Datensätze, der Aufbau der wichtigsten Himmelsmechaniken, sowie die Schnittstelle zur NASA behandelt. Damit soll eine Wissensbasis geschaffen werden, um das Programm zur Umsetzung der Ziele erstellen zu können.
In Kapitel 3 werden dann nacheinander die wichtigsten Grundfunktionen des Programms erklärt. Dabei handelt es sich um einzelne Funktionen, die zusammen die Werkzeuge bilden, um unbekannte Objekte aus den Datensätzen zu extrahieren. Dabei geht es vor allem um das Filtern der bereits vorhandenen Daten, das Anfordern weiterer Daten und das Vergleichen von diesen Datensätzen.
In Kapitel 4 werden diese Funktionen zusammengesetzt. Gleichzeitig wird eine GUI erstellt, die es dem Benutzer ermöglicht, mit den Daten zu arbeiten. Das Hauptziel besteht darin, die GUI so zu gestalten, dass am Ende die Möglichkeit gegeben ist, unbekannte Objekte zu identifizieren.
Das Kapitel 5 fasst die gesamte Implementierung noch einmal zusammen. Es werden die Hauptaspekte dargestellt und ein Ausblick auf zukünftige Arbeiten und Erweiterungen gegeben.

2 Grundlagen

Für die Erstellung des Programms zur automatischen Erkennung von Korrelationen zwischen Datenpunkten müssen zunächst wichtige Grundlagen geklärt werden. Zum einen muss eine Programmierumgebung sowie eine geeignete Programmiersprache ausgewählt werden. Zum anderen müssen die wichtigsten Einheiten in Bezug auf die Datenpunkte betrachtet werden.

2.1 Programmiersprache und Programmierumgebung

Als Programmiersprache wird Python gewählt. Dies hat mehrere Gründe. Zum einen wurde das Programm, welches die Auswertung der TESS Bilder durchführt, ebenfalls in Python geschrieben. Somit könnte dieses Programm und das im Rahmen dieser Bachelorarbeit entstehende Programm leichter zusammengeführt werden. Außerdem ist Python eine höhere Programmiersprache, mit den Vorteilen einer verständlichen Struktur, viel Funktionalität mit wenigen Zeilen Code, komplexen Datentypen und vielem mehr. Der Nachteil einer hohen Programmiersprache ist vor allem der Geschwindigkeitsverlust bei vielen Wiederholungen von Operationen.
Ein weiterer großer Vorteil von Python ist die große Menge an bereits existierenden Bibliotheken, die zur Verfügung stehen. Durch diese lassen sich viele Probleme mit bereits vorhandenen Bibliotheken lösen und ein größerer Programmieraufwand kann so vermieden werden.[2, S.5]

Als Programmierumgebung wird Visual Studio Code auf einem Linux-Betriebssystem verwendet. Grund hierfür ist auch die Kompatibilität zum Bildverarbeitungsprogramm. Dieses ist betriebssystemspezifisch nur auf einem Linux Betriebssystem lauffähig. Um sicherzustellen, dass das resultierende Programm auch auf einem Linux-Betriebssystem lauffähig ist, wird es direkt auf diesem Betriebssystem erstellt, wobei wahrscheinlich keine Linux-spezifischen Funktionen verwendet werden und somit auch andere Betriebssysteme funktionieren würden.

2.2 Datenpunkte

Um mit den Datenpunkten arbeiten zu können, muss zunächst geklärt werden, wie diese gespeichert werden. Jeder Punkt besteht aus einem Orts- und einem Zeitstempel. Insgesamt gibt es drei verschiedene Werte, zum einen die Zeit und zum anderen zwei Koordinaten zur Beschreibung des Ortes.

2.2.1 Julianisches Datum

Der Zeitstempel wird nicht in dem aus dem Alltag bekannten Zeitformat (Tag.Monat.Jahr Stunde:Minute:Sekunde) angegeben. Hierfür wird das sogenannte Julianische Datum verwendet. Das Julianische Datum oder die Julianische Tageszählung ist, wie der Name schon sagt, eine linear fortschreitende Zählung der Tage. Für Bruchteile eines Tages wird eine Dezimalstelle angegeben. Der Tag Null, also der erste Tag des Julianischen Datums, wurde auf den 01.01.4713 v. Chr. festgelegt und ein neuer Tag beginnt immer um 12 Uhr Coordinated Universal Time (UTC). Der Vorteil dieser Zeitangabe besteht darin, dass Besonderheiten wie z.B. unterschiedlich lange Monate oder Schaltjahre nicht berücksichtigt werden müssen.
Allerdings muss aufgepasst werden, denn es gibt auch eine neuere Version des Julianischen Datums, das sogenannte modifizierte Julianische Datum. Dieses funktioniert nach dem gleichen Prinzip, aber das Startdatum ist der 17.11.1858 um 0 Uhr. Die Umrechnung zwischen den beiden Datumsformen sieht also wie folgt aus [3, S.19]:

$$MJD = JD - 2.400.000,5$$

Sowohl das Bildauswertungsprogramm als auch die NASA verwenden jedoch das julianische Datum.

2.2.2 Koordinatensystem

Die Datenpunkte haben neben dem Zeitstempel zwei Positionsangaben. Diese werden im bewegten äquatorialen Koordinatensystem angegeben. Dieses Koordinatensystem beschreibt die Position eines Himmelsobjektes durch seine Rektaszension und Deklination. Diese Koordinaten sind das Äquivalent der Himmelskoordinaten zu den irdischen Längen- und Breitengraden. Allerdings werden die Koordinaten auf ein festes Koordinatennetz am Himmel übertragen. Der Himmelsäquator entspricht dabei der Projektion des Erdäquators auf den Weltraum. Die Deklination kann nun bestimmt werden, indem der Winkel zum Himmelsäquator gemessen wird.

Das Äquivalent zum Nullmeridian wird durch den Stand der Sonne bestimmt, wenn sie den sogenannten Frühlingspunkt einnimmt. Wenn die Sonne am 21. oder 22. März den Himmelsäquator kreuzt, steht sie genau auf der Linie des Himmelsnullmeridians. Wird nun der Winkel zwischen dem Himmelsnullmeridian und dem betrachteten Punkt gemessen, so ergibt sich die Rektaszension.
Eine solche Definition ist notwendig, da sonst der Längengrad am Himmel durch die Erdrotation immer unterschiedlich wäre. Durch diese Definition ist das Koordinatensystem am Himmel jedoch, unabhängig von der Erdrotation, immer gleich. [3, S.9]

3 Funktionen

3.1 Daten einlesen

Das erste Ziel ist das Einlesen der Daten aus dem Bildauswertungsprogramm. Dadurch können diese dargestellt, verglichen und bearbeitet werden. Dies ist die Grundlage, um darauf aufbauend unbekannte Objekte in den Daten lokalisieren zu können.

3.1.1 Struktur analysieren

Um die bereits vorhandenen Daten einlesen zu können, muss zunächst die Struktur der Datei analysiert werden. Die vom Bildauswertungsprogramm ausgegebenen Dateien haben alle den gleichen Aufbau (siehe Abbildung 3.1).

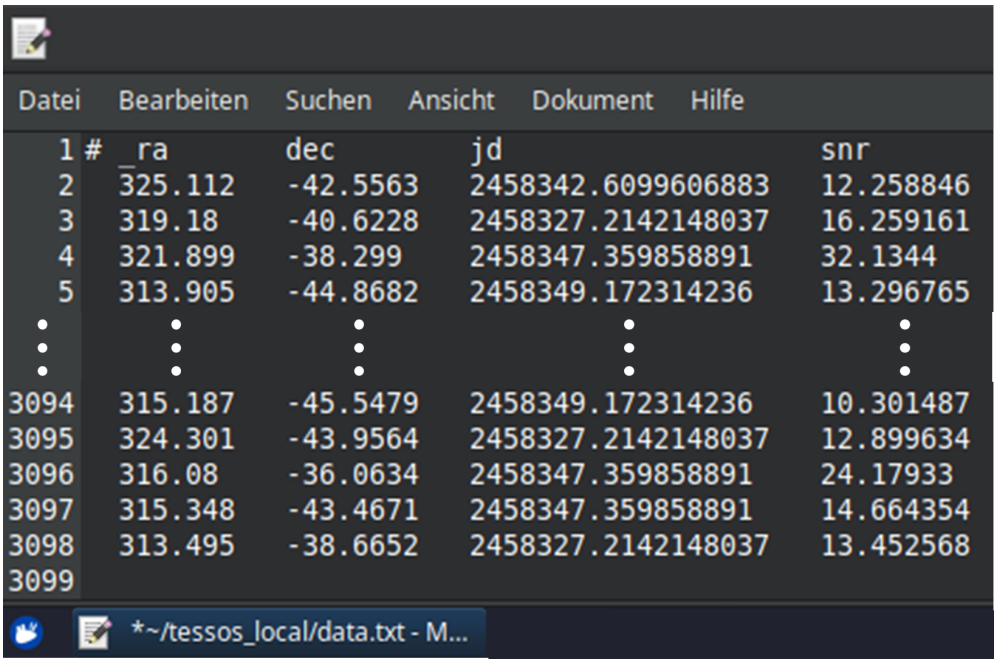

Abbildung 3.1: Auszug aus dem bereitgestellten Datensatz

In der ersten Zeile werden die Daten in den zugehörigen Spalten definiert. Die folgenden Zeilen enthalten dann die entsprechenden Werte, die zusammen einen Punkt mit Zeit- und Ortsstempel ergeben.
Dabei werden folgende Größen angegeben:

- Rektaszension (RA) - Längengrad im astronomischen Koordinatensystem
- Deklination (DEC) - Breitegrad im astronomischen Koordinatensystem
- Julianisches Datum (JD) - Zeitstempel
- Signal-Rausch-Verhältnis (SNR) - Zur Bewertung der Qualität der Auffälligkeit

Das Signal-Rausch-Verhältnis kann vernachlässigt werden, da alle auffälligen Datenpunkte zur Bearbeitung angezeigt werden sollen.

3.1.2 Daten zwischenspeichern

Um die Daten in Python weiterverarbeiten zu können, müssen sie zunächst in einer entsprechenden Variable zwischengespeichert werden. Die Daten bestehen aus mehreren Zeit- und Ortsstempeln. Da es sich um ein sich wiederholendes Schema handelt, bietet sich hierfür eine Liste an.
In einer Liste können in Python mehrere Daten des gleichen Variablentyps gespeichert werden. Um diese Daten zu speichern, wird also eine Liste benötigt, die Listen enthält, die wiederum jeweils drei Gleitkommazahlen speichern. Die Gleitkommazahlen sind jeweils einmal die Koordinate für die RA, die Koordinate für die DEC und der Zeitstempel. Somit entsteht eine Liste von markanten Punkten, die aus zwei Ortsstempeln und einem Zeitstempel bestehen.

3.1.3 Übersicht im Programm

Die Auslagerung einzelner Funktionen in eine separate Datei kann zu einer besseren Übersicht im Programm führen. Es bietet sich daher an, das Einlesen der Datei als separate Funktion zu programmieren und diese auszulagern.

```
1 #Funktion zum Einlesen von Elins Daten
2 def data_read_elin(name = ' '):
3     data = []
4     data_sorted = []
5     file_name = name
6     #Fehlerhafte Datei abfangen
7     try:
```

```
8           datei = open(file_name,'r')
9       except:
10          raise InvalidDocument
11
12      line = datei.readline()
13      if line[0] != "#":
14          raise InvalidDocument
15
16      #Jedes Zeile einlesen
17      while True:
18          line = datei.readline()
19          flag = True
20          for char in line:
21              if(char != ' '):
22                  flag = False
23          if(flag==True):
24              break
25          data.append(line.split())
26
27      #Daten abspeichern
28      for line in data:
29          line_sorted = [line[2], line[0], line[1]]
30          data_sorted.append(line_sorted)
31
32      datei.close
33      return data_sorted
```

Listing 3.1: Funktion zum Einlesen der Daten aus dem Bildauswertungsprogramm

Der Programmauszug 3.1 zeigt die entsprechende Funktion, die aus mehreren Teilen besteht. Zuerst wird geprüft, ob die Datei geöffnet werden kann. Im Fehlerfall wird der entsprechender Fehler zurückgegeben. Wenn die Datei geöffnet werden konnte, wird anhand des Inhalts der ersten Zeile überprüft, ob die Datei dem Standard des Bildauswertungsprogramms entspricht. Ist dies der Fall, werden die Daten zeilenweise eingelesen und in der entsprechenden Liste gespeichert. Dazu wird der Inhalt der Zeile in die einzelnen drei Gleitkommazahlen zerlegt. Am Ende wird die erzeugte Liste zurückgegeben.

3.2 Daten verarbeiten

Die Daten müssen in mehrfacher Hinsicht verarbeitet werden. Sie müssen dargestellt werden, um eine Weiterverarbeitung der Daten zu ermöglichen, zusammenhängende Punkte müssen

herausgefiltert werden und die korrelierenden Daten aus der NASA-Datenbank müssen aus dem Datensatz ermittelt werden.

3.2.1 Daten anzeigen

Zur Darstellung der Daten wird die Bibliothek „Matplotlib“ verwendet. Diese kann die Datenpunkte entsprechend in einem Koordinatensystem darstellen und erlaubt eine umfangreiche Bearbeitung.

```
datei_elin = 'data.txt'
data_elin = fnkt.data_read_elin(datei_elin)

xValues_elin = []
yValues_elin = []

for row in data_elin:
    xValues_elin.append(float(row[1]))
    yValues_elin.append(float(row[2]))

plt.scatter(xValues_elin, yValues_elin, s=1)
plt.show()
```

Listing 3.2: Code zur Darstellung der Daten mit der „Matplotlib“ Bibliothek

Das Beispielprogramm 3.2 zeigt, wie die Bibliothek die Darstellung der Daten vereinfacht. Mit wenigen Programmzeilen können die Daten wie folgt angezeigt werden:

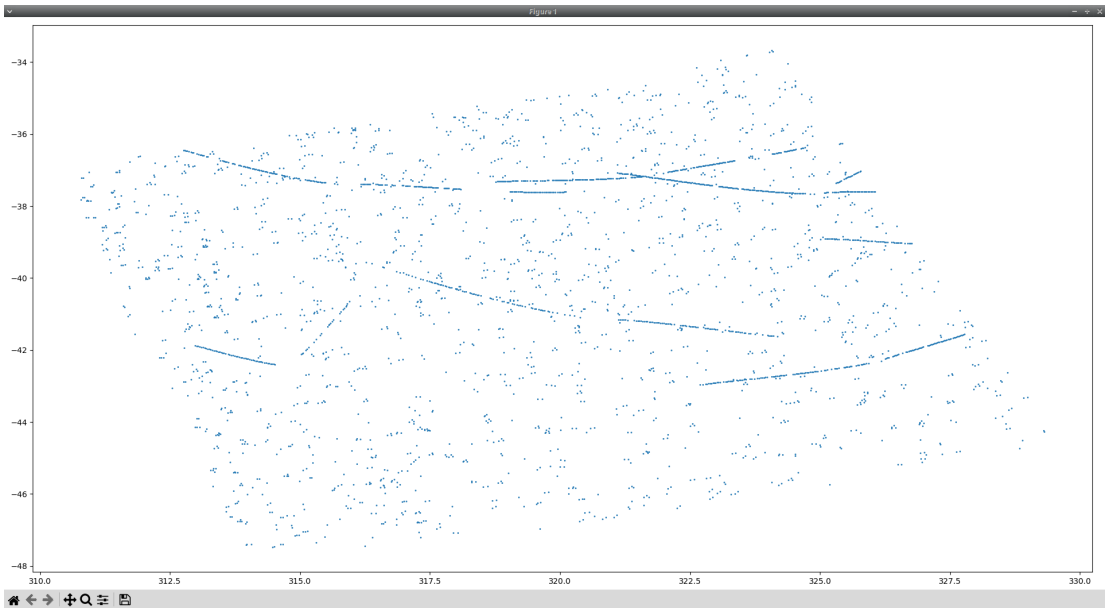

Abbildung 3.2: Einfache Darstellung der Daten aus dem Bildauswertungsprogramm

In der Abbildung 3.2 werden alle auffälligen Objekte dargestellt. Unten links ist zu erkennen, dass das Fenster weitere Funktionen bietet, um die Ansicht zu verändern. Es ist möglich hinein- und herauszoomen, den Ausschnitt verschieben, die Größe des Koordinatensystems anpassen und das Ganze auch speichern. Auf diese Weise kann mit wenigen Zeilen Code eine benutzerfreundliche Oberfläche erstellt werden.

3.2.2 Daten filtern

Die Daten aus dem Bildauswertungsprogramm zeigen viele weit verstreute Punkte. Bei genauerer Betrachtung fallen jedoch auch Verbindungen zwischen den Punkten auf. Dies könnte auf ein Sonnensystemobjekt hindeuten, das sich während der Beobachtungszeit durch den aufgenommenen Bildbereich bewegt hat. In der Abbildung 3.2 ist an mehreren Stellen zu erkennen, dass die Auffälligkeiten eine Linie bilden.
Diese Zusammenhänge gilt es nun herauszufiltern. Dazu müssen die einzelnen Datenpunkte auf ihren Abstand hin untersucht werden und wenn genügend Punkte in einem nahen Abstand gefunden wurden, kann davon ausgegangen werden, dass es sich um dasselbe Sonnensystemobjekt handelt, das durch das Bild gewandert ist.

Für eine programmtechnische Umsetzung bietet sich auch die Auslagerung in eine eigene Funktion an. Diese erhält die zuvor erzeugte Liste und gibt nach der Filterung eine Liste mit den

gefilterten Daten zurück. Zusätzlich können die entsprechenden Filterparameter übergeben werden.

```
#Funktion zum Filtern von Elins Daten
def chain_finder(data_elin, ra_border = 0.05, dec_border = 0.05, min_points = 5):
    data= data_elin.copy()
    finish = []
    #Punkte in Elins Daten miteinander vergleichen
    for point in data:
        data_comp = data
        num = 0
        result = []
        new_point = point
        data_comp.remove(point)
        test = 0

        #ähnliche Punkte finden
        while True:
            flag = False
            for compare in data_comp:
                x = abs(float(new_point[1]) - float(compare[1]))
                y = abs(float(new_point[2]) - float(compare[2]))
                #Wenn der Punkt in den Grenzen ist
                if x < ra_border and y < dec_border:
                    result = result + [new_point]
                    #Gefundene Punkte aus dem Datensatz entfernen und seperat abspeichern
                    data_comp.remove(compare)
                    new_point = compare
                    num = num+1
                    flag = True
                    break
            #Wenn kein neuer Punkt gefunden wurde aber genug Punkte zusammen sind
            if flag == False and num > min_points:
                #Letzten Punkt abspeichern und die eine Suche beenden
                result.append(new_point)
                break
            #Wenn kein neuer Punkt gefunden wurde und noch nicht genug Punkte zusammen sind
            elif flag == False and num <= min_points and num >0:
                #Zum vorherigen Punkt zurückkeheren und andere Punkte suchen
```

```
37                  num = num-1
38                  new_point = result[num-1]
39                  result.remove(result[(num)])
40              #Wenn es keine möglichen Punkte mehr gibt
41              elif flag == False and num == 0:
42                  break
43              #Wenn ein neuer Punkt gefunden wurde dann wird weiter geschaut
44
45          #gefunde Daten werden abgespeichert
46          for data_point in result:
47                  if data_point in data:
48                      data.remove(data_point)
49
50
51          finish = finish + result
52      #Rückgabe des Ergebniss
53      return finish
```

Listing 3.3: Funktion zur Filterung der Daten

Das Prinzip der Filterung bezieht sich auf den maximalen Abstand der Punkte. Wie im Programmausschnitt 3.3 in der zweiten Zeile zu sehen ist, werden der Funktion die entsprechenden maximalen Abstände übergeben. Zusätzlich wird eine minimale Anzahl von Punkten an die Funktion übergeben. Nur wenn mindestens so viele Punkte, wie vorher angegeben, nahe genug beieinander liegen, werden die Punkte von dem Filter berücksichtigt.

Iterative Programmierung

Die Implementierung der Filterung erfolgt durch eine iterative Programmierung. Dabei werden zunächst alle Datenpunkte mittels einer „for-Schleife“ durchlaufen. Für jeden Punkt wird geprüft, ob er mit genügend anderen Punkten einen Zusammenhang bildet. Dazu werden zwei weitere Schleifen verwendet. Die äußere Schleife dient dazu, für den aktuell betrachteten Punkt alle Korrelationen zu finden. Um die Korrelationen zu finden, durchläuft die innere Schleife alle Datenpunkte. Dabei wird der Abstand des Punktes der inneren Schleife mit dem zuletzt betrachteten Punkt verglichen. Liegt dieser nahe genug am betrachteten Punkt, wird dieser zum neuen betrachteten Punkt. Zusätzlich wird der vorherige Punkt in einer separaten Liste gespeichert und aus der ursprünglichen Liste entfernt. Danach beginnt die Suche nach einem weiteren nahegelegenen Punkt von vorne mit dem neuen zu betrachtenden Punkt. Wenn kein

Punkt aus der ursprünglichen Liste mehr nahe genug ist, gibt es verschiedene Möglichkeiten, wie verfahren wird:

1. Wenn durch dieses Verfahren die erforderliche Mindestanzahl von Punkten gefunden wurde, werden diese als gefilterte Punkte gespeichert. Danach wird der nächste Punkt der Hauptschleife betrachtet. Bereits gefilterte Punkte werden aus der Liste entfernt.

2. Wenn jedoch noch nicht genügend Punkte gefunden wurden, wird der aktuell betrachtete Punkt verworfen und der zuvor betrachtete Punkt wird wieder zum aktuell zu betrachtenden Punkt. Auf diese Weise wird sichergestellt, dass die seitlichen Punkte bei einer linearen Häufung nicht mit herausgefiltert werden.

3. Wenn der aktuell betrachtete Punkt der Startpunkt der Hauptschleife ist, wird dieser Punkt verworfen, da von diesem Punkt aus keine Häufungen gefunden werden konnten.

In jedem Fall springt das Programm danach zum nächsten Punkt in der Hauptschleife und das Suchverfahren wird erneut durchlaufen.
Am Ende werden alle gefundenen Punkte als Liste zurückgegeben.

Rekursive Programmierung

Der erste Ansatz bestand darin, die Filterung über eine rekursive Funktion zu implementieren. Das bedeutet eine Funktion, die sich so lange selbst aufruft, bis keine Punkte mehr gefunden werden können. Dabei traten jedoch mehrere Probleme auf, die mit einer iterativen Programmierweise einfacher zu lösen waren.
Zum einen trat der Fehler auf, dass die maximale Rekursionstiefe von 1000 überschritten wurde. In Python kann sich eine Funktion standardmäßig maximal 1000 mal selbst aufrufen [4], wobei hier noch ein weiterer Fehler in der Programmierung vorliegen musste, da bei der Anzahl der Daten eigentlich keine Tiefe von 1000 erreicht werden konnte. Allerdings könnte dieses Problem bei größeren Datensätzen auftreten.
Zum anderen war die Implementierung der erneuten Betrachtung des gleichen Datenpunktes aus dem vorherigen Durchlauf schwieriger umzusetzen. Da nicht, wie üblich, die Funktion eine gewisse Anzahl rekursiv aufgerufen wird und dann die Ebenen wieder linear nach oben gegeben werden, sondern da zwischendurch immer wieder die Ebenen gewechselt werden müsste.
Somit war die Umsetzung durch eine iterative Programmierweise für diesen Anwendungsfall effizienter.

Gefilterte Daten anzeigen

Die gefilterten Daten können nun genauso angezeigt werden wie die ursprüngliche Liste. Dazu muss der Code 3.2 um folgende Zeilen erweitert werden:

```
1  data_elin_flt = fnkt.chain_finder(data_elin, 0.08, 0.08, 10)
2
3  xValues_elin_flt = []
4  yValues_elin_flt = []
5
6  for row in data_elin_flt:
7      xValues_elin_flt.append(float(row[1]))
8      yValues_elin_flt.append(float(row[2]))
9
10 plt.scatter(xValues_elin_flt, yValues_elin_flt,s=1, color = 'green')
```

Listing 3.4: Zusätzlich benötigter Code um ebenfalls die gefilterten Daten anzuzeigen

In dem Beispiel aus dem Programmausschnitt 3.4 sind die Filterparameter auf einen maximalen Abstand von 0,08 Grad in beiden Achsen und eine minimale Anzahl von 10 Punkten eingestellt.

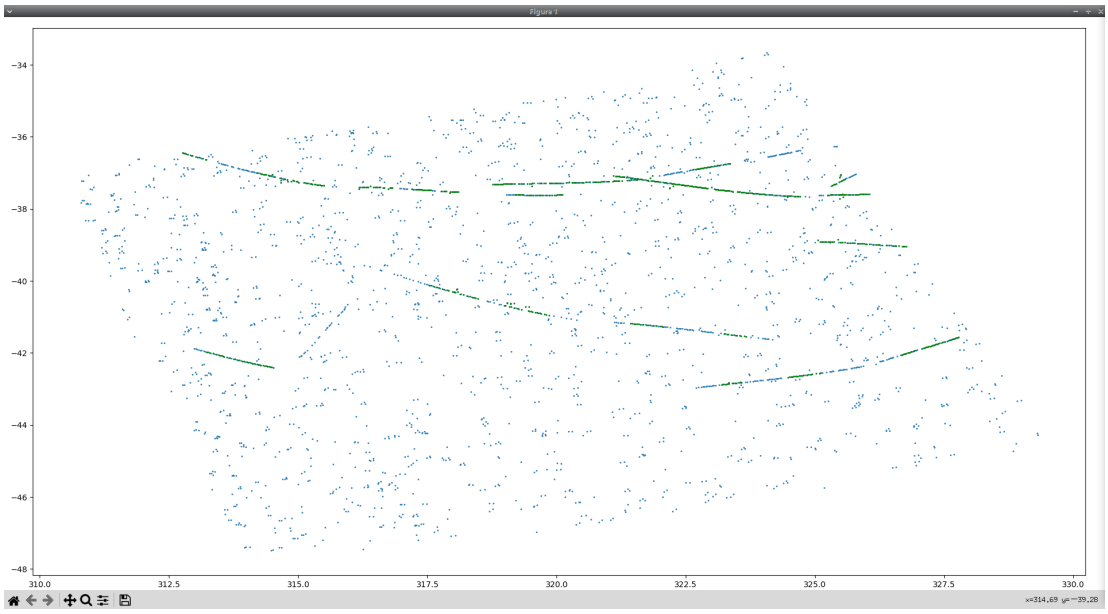

Abbildung 3.3: Darstellung der gefilterten Daten

In der Abbildung 3.3 sind die gesamten Datenpunkte in blau und die gefilterten Daten in grün dargestellt. Es ist zu erkennen, dass ein Großteil der linear zusammenhängenden Punkte heraus-

gefiltert und hervorgehoben wurde. Allerdings ist im Koordinatensystem bei $ra(x - Achse) \approx 315$ und $dec(y - Achse) \approx 41$ noch eine weitere lineare Anhäufung von Punkten zu sehen, die nicht gefiltert wurde. Um besagte Punkte ebenfalls herauszufiltern, müssten die Filterbedingungen angepasst werden. Entweder müsste ein größerer Abstand zwischen den Punkten gewählt werden oder es müssten weniger Punkte benötigt werden. Eine ungenauere Filterung führt jedoch auch dazu, dass nichtlineare Häufungen von Punkten ebenfalls herausgefiltert werden. Diese werden dann bei der weiteren Verarbeitung ebenfalls berücksichtigt, was zu längeren Verarbeitungszeiten führen kann.

3.3 NASA-Daten anfordern

Entsprechend dem Datensatz aus dem Bildverarbeitungsprogramm müssen die korrelierenden Daten bei der NASA angefordert werden. Dazu wird die API Schnittstelle verwendet, die NASA auf ihrer Website kostenlos zur Verfügung stellt [5]. Dabei wird die Anfrage durch Anpassung des aufgerufenen Links erstellt, wodurch ein entsprechenden Ergebnisse erhalten wird.
Die Abfrage liefert Positionsdaten von Objekten zu einem bestimmten Zeitpunkt in einem bestimmten Himmelsausschnitt. Der Datensatz aus dem Bildauswertungsprogramm enthält jedoch Daten, die über einen längeren Zeitraum aufgenommen wurden. Daher müssen mehrere Anfragen für diesen Zeitraum durchgeführt werden, um die Historie der Objekte zu erhalten.

3.3.1 Die Schnittstelle

Die „Small-Body Identification API“ der NASA ist die Schnittstelle, über die diese Daten angefordert werden können. Dazu werden die gewünschten Attribute nach einer bestimmten Syntax an den Link „https://ssd-api.jpl.nasa.gov/sb_ident.api‘ angehängt. Dieser wird dann aufgerufen und die gewünschten Daten werden zurückgegeben. Es gibt verschiedene Attribute, die die Suche nach Objekten einschränken können. Für die Anfrage nach den korrelierenden Daten zum Bildauswertungsprogramm werden die folgenden Attribute verwendet [5]:

- Location defined by geodetic coordinates:
  - geodetic latitude of observer (Link:„lat=0“)
  - geodetic longitude of observer (Link:„lon=0“)
  - altitude above the reference ellipsoid (Link:„alt=0“)
- Observation Constraint Parameters

  - date/time of the observing night (Link:„obs-time= [1]“)
  - visual magnitude threshold (Link:„vmag-lim= [1] “)
- Field of View Parameters
  - right-ascension values of the edges of the FOV (Link:„fov-ra-lim= [1]“)
  - declination values of the edges of the FOV (Link:„fov-dec-lim= [1]“)

Die Koordinaten des Standortes des TESS ändern sich je nach Zeitpunkt der Aufnahme, da sich der TESS um die Erde dreht. Die Änderung des Winkels auf das betrachtete Gebiet ist jedoch vernachlässigbar. Daher wird der Standort vereinfacht auf der Erdoberfläche festgelegt. Die Koordinaten sind dann $lat = 0°N$, $lon = 0°O$ und eine Höhe von $alt = 0m$.

Die Parameter zur Einschränkung der Observation setzen sich aus der Zeit und der scheinbaren Helligkeit zusammen.
Die Zeit wird aus den Zeitstempeln des Datensatzes des Bildverarbeitungsprogramms bestimmt. Es werden mehrere Anfragen an die NASA benötigt, die zeitlich zwischen dem minimalen und dem maximalen Zeitpunkt liegen. Daher ist eine gleichmäßige Unterteilung des Zeitintervalls sinnvoll.
Die scheinbare Helligkeit wiederum beschreibt die Helligkeit, die Objekte mindestens auf den Beobachter gehabt haben müssen, damit die NASA sie auf Anfrage ausgibt. Der TESS kann Objekte mit einer minimalen Helligkeit von $12mag$ [6] erkennen. Bei der Bildverarbeitung werden jedoch Helligkeitsunterschiede ausgewertet. Daher können auch dunklere Objekte in den Daten erkannt worden sein, vorausgesetzt sie dabei hellere Objekte verdeckt. Sehr dunkle Objekte sollten jedoch trotzdem herausgefiltert werden, da sie mit hoher Wahrscheinlichkeit zu klein sind, um einen Einfluss auf die Daten gehabt zu haben. Daher ist die Begrenzung der scheinbaren Helligkeit eine nützliche Ergänzung, um die Anzahl der möglichen Himmelskörper zu reduzieren.

Die letzten Parameter dienen der Begrenzung des Beobachtungsfeldes. Dazu müssen die maximale und minimale Rektaszension und Deklination angegeben werden. Diese ergeben sich aus dem Datensatz des Bildauswertungsprogramms.

[1] An diesen Stellen wird je nach Anfrage ein anderer Wert eingesetzt

3.3.2 Umsetzung im Programm

Das Erfassen der Daten von NASA, die mit den Daten aus dem Bildauswertungsprogramm korrelieren, wird ebenfalls in eine separate Funktion ausgegliedert. Diese bekommt als übergebende Variablen einmal den Dateinamen von den Daten des Bildauswertungsprogrammes, eine Genauigkeit für die Zeitspanne, sowie die gewünschte scheinbare Helligkeit übergeben.

Minima und Maxima

Zunächst müssen die Daten aus dem Bildauswertungsprogramm in einer Liste gespeichert werden, was mit der Funktion 3.1 geschieht. Aus dieser Liste müssen dann jeweils die Minima und Maxima der Zeit, der Rektaszension sowie der Deklination ermittelt werden.

```
#Maxima und Minima finden
for point in data_elin:
    if point[0] < time_min:
        time_min = point[0]
    elif point[0] > time_max:
        time_max = point[0]

    if float(point[1]) < ra_min:
        ra_min = float(point[1])
    elif float(point[1]) > ra_max:
        ra_max = float(point[1])

    if float(point[2]) < dec_min:
        dec_min = float(point[2])
    elif float(point[2]) > dec_max:
        dec_max = float(point[2])
```

Listing 3.5: Code zur Ermittelung der Minima und Maxima

Zur Ermittlung der Minima und Maxima werden alle Punkte durch eine „for-Schleife“ verglichen. Ist ein Wert kleiner/größer, wird dieser als neues Minimum/Maximum gespeichert.

Anpassung des Formates

Die gespeicherten Werte sind jedoch noch im falschen Format. Die Werte für die Rektaszension müssen im 24-Stunden-Format (hh-mm-ss.[ss]) und die Werte für die Deklination im Grad-Minuten-Sekunden-Format (dd-mm-ss.[ss]) angegeben werden. Außerdem müssen alle negativen Vorzeichen durch ein großes „M“ ersetzt werden, da die API diese sonst nicht erkennt.

```
#Min. RA ins richtige Format formatieren
ra_min_angle = Angle(str(ra_min)+"d")
ra_min_angle = ra_min_angle.to_string(unit=un.hour, sep='-')
if ra_min_angle[0] == '-' and ra_min_angle[2] == '-':
    ra_min_angle = ra_min_angle.replace("-","M0",1)
elif ra_min_angle[0] == '-':
    ra_min_angle = ra_min_angle.replace("-","M",1)
elif ra_min_angle[1] == '-':
    ra_min_angle = '0'+ra_min_angle
```

Listing 3.6: Code zur Anpassung des Formates der Werte

Die Formatanpassung wird über die „astropy“ Bibliothek realisiert. Zunächst wird der Winkel im bestehenden Format der Bibliothek gespeichert. Dabei wird der Wert der minimalen Rektaszension, sowie das Format der Eingabe übergeben. Das Format der Eingabe ist die Einheit Grad, was durch ein kleines „d“ angegeben wird. Anschließend wird der Winkel mit der Funktion „to_string“ in das gewünschte Format konvertiert. Im Falle der Rektaszension ist dies das 24-Stunden-Format, das mit „un.hour“ angegeben wird. Für die Deklination wird genauso verfahren, nur das Format wird geändert. Es handelt sich um das Format „Grad Minuten Sekunden“, das mit „un.degree“ angegeben wird.
Zusätzlich müssen, falls vorhanden, negative Vorzeichen durch das große „M“ ersetzt werden.

Anzahl der Anfragen

Um die Anzahl der Anfragen zu bestimmen, muss die Länge des Zeitraums bekannt sein, der von den Datensätzen des Bildauswertungsprogramms abgedeckt wird.

```
#Anfragenanzahl berechnen
time_min = round(float(time_min))
time_max = round(float(time_max))
time_period = time_max-time_min

#Bestimmte Anzahl an anfragen druchführen
for day in list(range(int(time_period*(1/time_accuracy)))):
    time = time_min+(day*time_accuracy)
```

Listing 3.7: Code für die Anzahl an Anfragen

Der Zeitraum ergibt sich aus der Subtraktion der minimalen Zeit von der maximalen Zeit. Da die Zeiten im Julianischen Datum angegeben sind, kann dies durch die Rechenoperation wie in dem Code 3.7 durchgeführt werden.
Die an die Funktion übergebene Genauigkeit wird in Tagen angegeben. Da das julianische

Datum ebenfalls tageweise hochgezählt wird, kann über eine prozentuale Berechnung ermittelt werden, wie viele Anfragen gestellt werden.
Sollen z.B. für jeden zweiten Tag Anfragen gestellt werden, so wird die Anzahl der Tage des Zeitraums mit dem Kehrwert der Genauigkeit multipliziert. Mit einer „for-Schleife“ wird nun die Anfrage entsprechend oft gestellt. Dabei wird die jeweilige Zeit für die Anfrage in dem entsprechenden Durchlauf berechnet. Dazu wird die Anzahl der Durchläufe mit der Genauigkeit multipliziert und zur minimalen Zeit addiert. Die Zeit steigt also linear mit der Anzahl der Anfragen.

Anfrage und übergebene Daten

Die Anfrage wird durch den Aufbau des Links realisiert. Dieser wird dann aufgerufen und die Antwort gespeichert.

```
1 #Link aus allen Bestandteilen zusammensetzen
2 link = "https://ssd-api.jpl.nasa.gov/sb_ident.api?lat=0&lon=0&alt=0&obs-time
      ="+str(time)+"&fov-ra-lim="+ra_min_angle+","+ra_max_angle+"&fov-dec-lim=
      "+dec_min_angle+","+dec_max_angle+"&vmag-lim="+str(vmag)
3
4 #HTML Anfrage senden
5 response = requests.get(link)
6 result = response.json()
```

Listing 3.8: Code zur Erstellung des Links und der entsprechenden Anfrage

Der Link für die Anfrage setzt sich aus den in Kapitel 3.3.1 genannten Parametern zusammen. Der erste Teil besteht aus der Adresse der NASA-API und den Standortdaten des Beobachters, da diese immer gleich sind. Danach folgen die Zeit, das Beobachtungsfeld und die scheinbare Helligkeit durch abwechselnden Gebrauch von Variablen für die Werte und Texte, um diese zu verbinden.
Der Link wird dann mit Hilfe der „requests-Bibliothek“ aufgerufen.

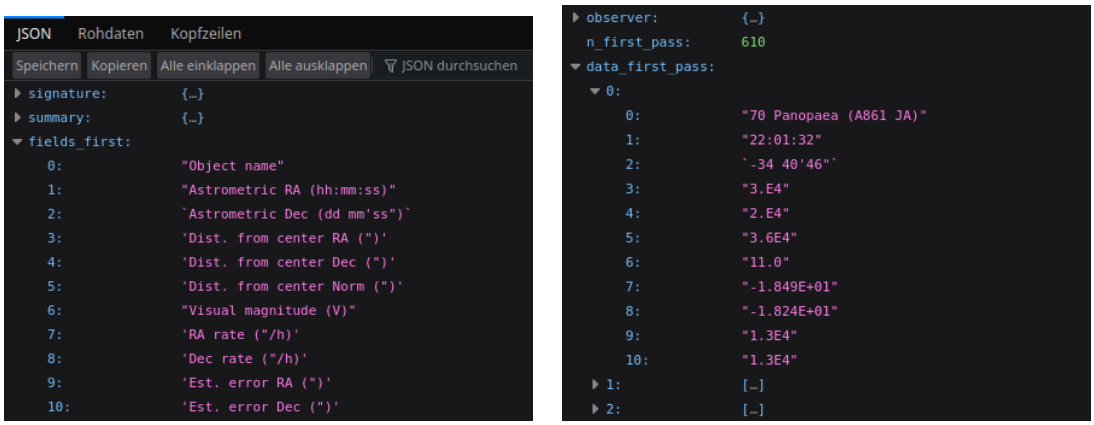

Abbildung 3.4: Beispiel Auszug aus den NASA-Daten

Die Antwort der NASA-API ist ein Text im JSON-Format. Mit Hilfe der „JSON-Bibliothek“ können die Daten einfach in Python ausgelesen werden. Im JSON-Format sind die Informationen in Ebenen unterteilt, die aus einer Beschreibung und einem Inhalt bestehen, ähnlich wie bei einer Liste mit Unterpunkten. Dabei kann der Inhalt auch eine weitere Ebene mit einer Beschreibung und zugehörigem Inhalt sein. Die Antwort der NASA-API besteht aus den folgenden sechs übergeordneten Elementen:

- signature - Enthält Informationen über die API
- summary - Enthält die Parameter, die für die Anfrage angegeben wurden
- fields_first - Enthält die Beschreibung der verschiedenen Felder für die folgenden Objekte
- observer - Enthält Informationen zum Beobachtungsstandort
- n_first_pass - Enthält die Anzahl der gefundenen Objekte
- data_first_pass - Enthält die verschiedenen Objekte mit den entsprechenden Informationen

Daten auslesen

Für die Auswertung der Daten sind vor allem die konkreten Objektdaten relevant. Darüber hinaus ist dem Feld „fields_first“ zu entnehmen, an welcher Stelle sich die benötigten Objektdaten befinden. Aus der Abbildung 3.4 können die Beschreibungen und die Objektdaten eines Beispiels entnommen werden. Bei mehreren Anfragen ist der Objektname wichtig, da

die Positionsdaten dem jeweiligen Objekt zugeordnet werden müssen. Außerdem wird der Zeitstempel benötigt, der bei der Anfrage angegeben wurde.

```
#Daten auswerten
count = result['n_first_pass']
count_list = list(range(count))
#Jedes Objekt durchgehen
for number in count_list:

    #Daten vom Objekt im richtigen Format zwischenspeichern
    name = result['data_first_pass'][number][0]
    RA = Angle(result['data_first_pass'][number][1], un.hour)
    RA = RA.to_value(un.degree)
    DEC = result['data_first_pass'][number][2]
    DEC = DEC.replace(" ", "°")
    DEC = Angle(DEC, un.degree)
    DEC = DEC.to_value(un.degree)

    counter = 0
    flag = False

    #Daten in einer Liste abspeichern
    #Data list: [name, [[time, RA, DEC],[...]],]
    #Nur passende Daten abspeichern
    if DEC > dec_min and DEC < dec_max and RA > ra_min and RA < ra_max:
        for point in data_list:
            #Nach passenden Objekt suchen
            if point[0] == name:
                data_list[counter][1].append([time, RA, DEC])
                flag = True
            else:
                counter = counter+1
        #Falls das Objekt noch nicht existiert wird ein neues angelegt
        if flag == False:
            data_list.append([name, [[time, RA, DEC]]])
```

Listing 3.9: Code zur Zwischenspeicherung der angefragten Daten

Die Implementierung ist im Code 3.9 dargestellt. Mit der Angabe im Feld „n_first_pass“ wird die Anzahl der Objekte bestimmt, die mit der „for-Schleife“ betrachtet werden sollen.
In der Schleife werden zunächst die notwendigen Daten des aktuell betrachteten Objekts zwischengespeichert. Dazu gehören der Name, die Rektaszension und die Deklination. Anschließend werden die Koordinaten wieder in Grad umgerechnet.
Da es bei mehreren Tests auch fehlerhafte Objekte gab, die nicht im gesuchten Koordinatenbe-

reich lagen, werden diese zunächst überprüft. Wenn die Koordinaten innerhalb der minimalen und maximalen Grenzen liegen, wird in der Liste für die Datenspeicherung geprüft, ob der Name des Objektes bereits existiert. Ist dies der Fall, wird das Koordinatenpaar mit dem entsprechenden Zeitstempel hinzugefügt. Wenn das Objekt noch keinen Eintrag hat, wird ein neuer Eintrag mit dem Namen und dem entsprechenden Koordinatenpaar mit Zeitstempel hinzugefügt. Die somit erstellte Liste ist wie folgt aufgebaut:

[[Name,[Zeit, RA, DEC],[Zeit, RA, DEC],...][Name,[Zeit, RA, DEC],[Zeit, RA, DEC],...],...]

Daten abspeichern

Die Abfrage der Daten dauert aufgrund der Menge der Anfragen recht lange. Es kann bis zu 30 Sekunden dauern, bis eine Anfrage von der NASA-API beantwortet wird. Wenn eine hohe Genauigkeit gewünscht wird, können es mehr als 100 Anfragen sein, wodruch die Zeit erheblich verlängert wird.
Dementsprechend müssen die von der NASA-API erhaltenen Daten für eine spätere Verarbeitung in einem Dokument gespeichert werden, da sonst die Anfragen wiederholt werden müssten.

```
#Daten in einer Datei abspeichern
#Data write: name1; time RA DEC; ... \n name2;
#Zeitstempel erstellen
current_time = dt.datetime.now()
current_time_str = str(current_time.strftime("%Y-%m-%d_%H:%M:%S"))

#Dateinamen erstellen
filepart = datei_name.split('/')
filepart.remove(filepart[-1])
filepart.append('nasa_data_'+current_time_str)
delimiter = '/'
file_write_name = delimiter.join(filepart)

#Datei öffnen und schreiben
datei = open(file_write_name+".txt",'w')
datei.write("Nasa data (RA: ("+str(ra_min)+")-("+str(ra_max)+"); DEC: ("+str
    (dec_min)+")-("+str(dec_max)+"); time: ("+t_min_link +")-("+t_max_link+"
    ); time accuracy: "+str(time_accuracy)+"; visual magnitude: "+str(vmag)+
    ")\r")
for point in data_list:
    datei.write(point[0])
    for combi in point:
```

```
20          if combi != point[0]:
21              for info in combi:
22                  datei.write(";")
23                  for num in info:
24                      datei.write(" ")
25                      datei.write(str(num))
26      datei.write("\r")
27  datei.close()
```

Listing 3.10: Code zur Speicherung der angefragten Daten

Bei dem Speichern wird eine neue Datei angelegt. Der Dateiname setzt sich aus dem Text „nasa_data_“ und dem aktuellen Datum mit Uhrzeit zusammen. In der ersten Zeile werden allgemeine Informationen zur Abfrage beschrieben. Es werden die Koordinaten, der Zeitraum und die scheinbare Helligkeit angegeben.
In den folgenden Zeilen werden alle Koordinaten und Zeiten eines Objektes je Zeile gespeichert. Zuerst wird der Name mit einem Semikolon als Trennzeichen gespeichert. Danach folgt der erste Zeitstempel mit den zugehörigen Koordinaten, wobei die Werte mit Leerzeichen getrennt sind. Danach folgt wieder ein Semikolon als Trennzeichen. Dieses Schema wird fortgesetzt, bis alle Werte des Objekts gespeichert sind. Danach folgt in der nächsten Zeile das gleiche Prinzip für das nächste Objekt.

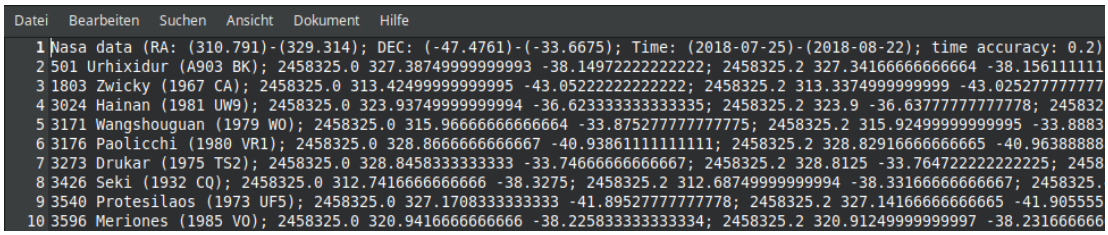

Abbildung 3.5: Beispielhafter Auszug aus den abgespeicherten NASA-Daten

Die Abbildung 3.5 zeigt nur einen Ausschnitt der gespeicherten Daten. Im eigentlichen Dokument sind wesentlich mehr Zeilen mit Objekten und wesentlich mehr Koordinaten und Zeitstempel gespeichert.

3.3.3 Überprüfen der erhaltenden Daten

Um sicherzustellen, dass die erhaltenen Daten mit den entsprechenden Daten in der NASA-Datenbank übereinstimmen, werden sie manuell überprüft. Die Überprüfung kann über das von der NASA zur Verfügung gestellte Tool erfolgen. Das besagte Tool bietet die Möglichkeit,

direkt auf deren Website eine Anfrage durchzuführen. Die entsprechenden Suchparameter müssen manuell eingegeben werden. Als Ergebnis wird eine Liste mit den entsprechenden gefundenen Objekten ausgegeben.[7]
Zur Überprüfung wird nun getestet, ob die Anzahl der gefundenen Objekte mit der Anzahl der Objekte in der Liste annähernd übereinstimmt. Bei der Anfrage durch das Programm kann es durch das Herausfiltern von Objekten, die nicht im Koordinatenbereich liegen, zu geringfügigen Abweichungen kommen. Zusätzlich werden die Elemente stichprobenartig verglichen. Es ist festzustellen, dass sowohl die Anzahl als auch die Stichproben bei beiden Abfragen übereinstimmen.

3.4 NASA-Daten filtern und anzeigen

Für die Weiterverarbeitung der gespeicherten Daten werden nun zwei Funktionen benötigt. Zum einen eine Funktion, die die Daten aus dem gespeicherten Dokument wieder in eine Liste umwandeln kann. Damit ist eine Anzeige der Rohdaten möglich. Und zum anderen eine Funktion, die die Daten der NASA mit denen des Bildauswertungsprogramms vergleicht und anhand bestimmter Parameter nach Korrelationen sucht.

3.4.1 Darstellen der NASA-Daten

Die Funktion zur Konvertierung der Daten in eine Liste ist ähnlich der Funktion 3.1. Auch hier wird zunächst das Dokument aufgerufen und dann Zeile für Zeile ausgelesen, um die Daten einzulesen. Der Unterschied liegt in der Art und Weise, wie die Zeilen eingelesen werden.

```
#Daten aus jeder Zeile herauslesen
while True:
    line = datei.readline()
    if line != "":
        save = line.split(";")
        for point in save:
            if point != save[0]:
                data.append(point.split())
    else:
        break
    counter = counter + 1

datei.close
#Daten zurückgeben
```

```
15 return data
```

Listing 3.11: Auszug aus der Funktion um Nasa-Daten einzulesen

Bei den NASA-Daten muss zunächst der Inhalt einer Zeile in Elemente zerlegt werden. Anschließend wird das erste Element, der Name des jeweiligen Objekts, verworfen. Diese werden für die Darstellung der Rohdaten nicht mehr benötigt. Alle weiteren Elemente der Liste werden erneut aufgeteilt, so dass die drei Elemente (Zeit, RA, DEC) voneinander getrennt werden. Auf diese Weise entsteht die gewünschte Liste, in der sich mehrere Listen befinden, die jeweils die Werte enthalten. Nachdem dieser Vorgang für alle Zeilen durchgeführt wurde, wird die vollständige Liste mit allen Elementen übergeben.
Diese Liste kann nach dem gleichen Prinzip wie in dem Code 3.2 angezeigt werden.

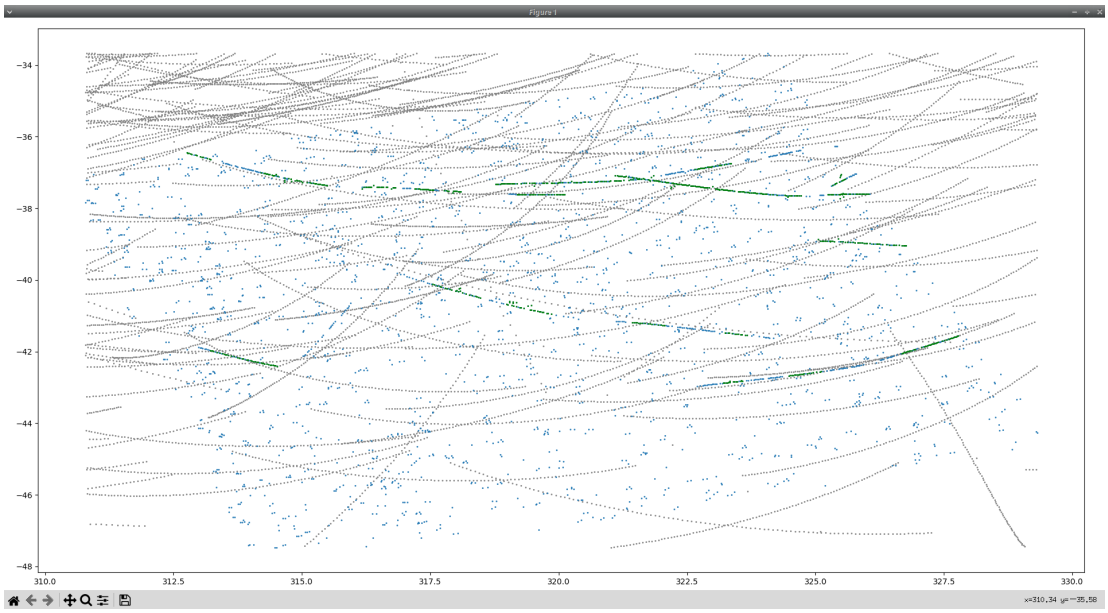

Abbildung 3.6: Darstellen der NASA-Daten

In der Abbildung 3.6 werden in grau die rohen NASA-Daten, die mit einer Genauigkeit von 0,25 Anfragen pro Tag und einer erforderlichen scheinbaren Helligkeit von 18,5 mag angefordert wurden, dargestellt. Es ist zu erkennen, dass es sehr viele Objekte gibt, die über den gesamten Koordinatenbereich verteilt sind. Der nächste Schritt ist nun, diese Daten zu filtern, um Zusammenhänge zu erkennen und gegebenenfalls Objekte aus dem Datensatz des Bildauswertungsprogramms zu identifizieren, die der NASA noch unbekannt sind.

3.4.2 NASA-Daten filtern

Für die Filterung der NASA-Daten müssen Korrelationen zwischen den NASA-Daten und den gefilterten Daten des Bildauswertungsprogramms gefunden werden. Dazu muss eine bestimmte Anzahl von Punkten, bestehend aus Zeit- und Ortsstempel, übereinstimmen. Durch Ungenauigkeiten des Bildauswertungsprogramms oder durch ungenaue Angaben bei der Anfrage der NASA-Daten kann es zu Abweichungen zwischen den Punkten kommen. Als Filterparameter dienen daher die maximal zulässigen Abweichungen für Rektaszension, Deklination und Zeit. Zusätzlich kann als weiterer Parameter festgelegt werden, wie viele Punkte eines Objektes innerhalb der Abweichungen liegen müssen, damit es für die Filterung in Frage kommt.

NASA-Daten einlesen

Anders als bei der Anzeige der Rohdaten ist es für die weitere Bearbeitung notwendig, den Namen und die Zugehörigkeit der Punkte zu kennen. Daher werden die Daten in der Filterfunktion zunächst ausgelesen und in einer Liste gespeichert.

```
#Datei auslesen und Daten zwischenspeichern
while True:
    line = datei.readline()
    if line != "":
        save = line.split(";")
        for point in save:
            if point == save[0]:
                database_list = database_list + [[point]]
            else:
                database_list[(counter)].append(point.split())
        counter = counter +1
    else:
        break
```

Listing 3.12: Einlesen der Nasa-Daten für die Filterung

Der Unterschied zwischen dem Einlesen der Daten in 3.12 und dem Einlesen der Rohdaten in 3.11 ist der Aufbau der Liste. Beim Einlesen der Daten zur Weiterverarbeitung wird die Liste so aufgebaut, wie die Daten im Dokument gespeichert wurden. Das Ergebnis ist eine Liste von Objekten. Diese Objekte werden durch eine Liste repräsentiert, die aus einem Namen als erstes Element und aus Zeit- und Ortsstempeln als folgenden Elementen besteht. Daraus ergibt sich folgende Liste:

[[Name;[Zeit; RA; DEC];[Zeit; RA; DEC];[...];...];[Name;[Zeit; RA; DEC];...];...]

Filterung der NASA-Daten

Die Filterung basiert auf dem Vergleich der Werte für Zeit, Rektaszension und Deklination. Dabei werden der Funktion maximal zulässige Abweichungen übergeben.

```
#Nasadaten mit Elinsdaten vergleichen und zusammenhänge finden
number = 0
#Für jedes Element der Nasadaten
for sb in database_list:
    number += 1
    counter = 0
    #Für jeden Punkt im Element
    for element in sb:
        #Wird jeder Punkt in Elins Daten verglichen
        for point in suspects:
            if element != sb[0]:
                time_diff = abs(float(element[0])-float(point[0]))
                if time_diff < time_diff_max:
                    RA_diff = abs(float(element[1])-float(point[1]))
                    if RA_diff < RA_diff_max:
                        DEC_diff = abs(float(element[2])-float(point[2]))
                        if DEC_diff < DEC_diff_max:
                            counter = counter+1
                            break
    #Wenn genug passende Punkte gefunden wurden, wird das Element
    abgespeichert
    if counter >= points:
        result_list.append(sb)

"""
for element in result_list:
    for point in element:
        if point != element[0]:
            result.append(point)
"""
#Die gefunden Elemente werden zurückgegeben
return result_list
```

Listing 3.13: Einlesen der Nasa-Daten für die Filterung

Für die Filterung wird mit einer „for-Schleife“ jedes Objekt der NASA-Daten betrachtet. Mit einer weiteren „for-Schleife“ wird dann jeder Punkt eines Objektes betrachtet. Jeder Punkt wird dann mit den gefilterten Daten aus dem Bildauswertungsprogramm verglichen. Der Vergleich bezieht sich zunächst auf die Zeit, dann auf die Rektaszension und schließlich auf die Deklination. Wenn ein Punkt innerhalb des zeitlichen und räumlichen Bereichs liegt, wird ein Zähler inkrementiert. Wenn genügend Punkte für ein Objekt gefunden wurden, wird dieses Objekt mit allen Punkten in der Ergebnisliste gespeichert. Am Ende, wenn alle Objekte betrachtet wurden, wird die Ergebnisliste zurückgegeben.

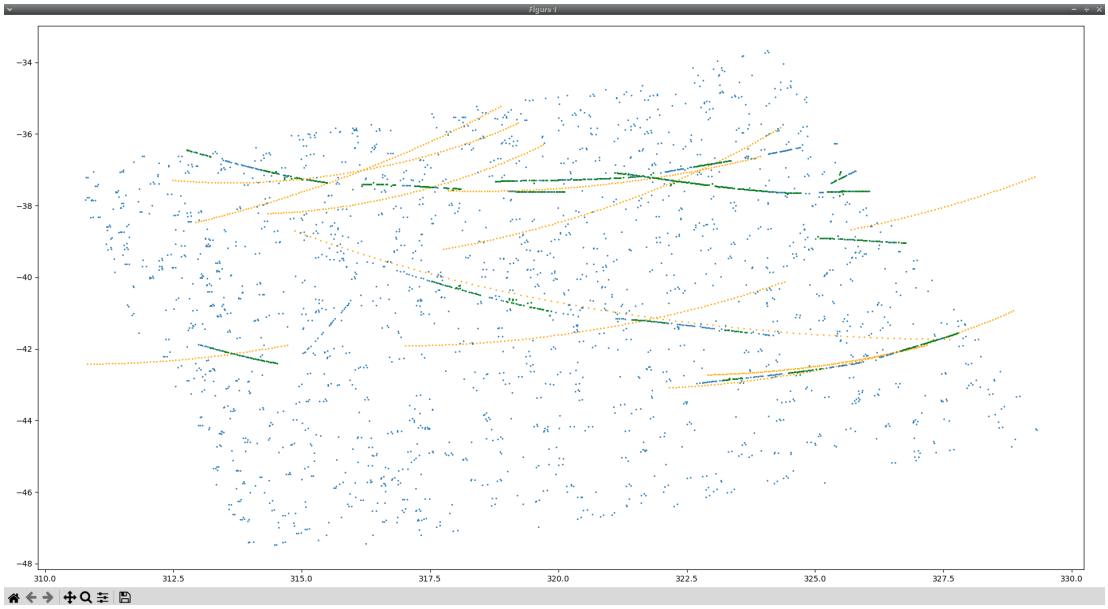

Abbildung 3.7: Gefilterte NASA-Daten

In der Abbildung 3.7 sind beispielhaft gefilterte NASA-Daten in gelb dargestellt. Die Daten aus der Abbildung 3.6 wurden mit folgenden Parametereinstellungen gefiltert:

- Benötigte Punkte: 14
- Genauigkeit der Zeit: 1 Tag
- Genauigkeit der Rektaszension: 0,6 Grad
- Genauigkeit der Deklination: 0,6 Grad

Die Parameter können verändert werden, um unterschiedliche Ergebnisse zu erhalten. Wird die Anzahl der benötigten Punkte reduziert oder die erlaubten Abweichungen erhöht, so werden

mehr passende Objekte gefunden. Es gibt zwei sinnvolle Vorgehensweisen. Die Erste ist die Anzahl der benötigten Punkte stark zu reduzieren, dafür aber die geforderte Genauigkeit sehr präzise einzustellen. Bei der Zweiten wird, wie in diesem Beispiel, eine hohe Ungenauigkeit zuzulassen, dafür werden aber die benötigten Punkte erhöht. So wird eine übersichtliche Auswahl von Objekten ausgegeben, die nur eine geringe Abweichung zu den gefilterten Daten des Bildauswertungsprogramms haben.

4 Grafische Benutzeroberfläche

Für die sinnvolle Anwendung der geschriebenen Funktionen muss eine geeignete Benutzeroberfläche, auch GUI genannt, erstellt werden. Diese soll es ermöglichen, die Datensätze des Bildauswertungsprogramms und die Daten der NASA entsprechend darzustellen, zu filtern und zu bearbeiten. Ziel ist es, dass ein Anwender bei der Auswertung der korrelierten Daten fehlende Objekte seitens der NASA entdecken und diesen nachgehen kann. Möglicherweise können so bisher unbekannte Objekte entdeckt werden.

4.1 Kriterien

Die GUI muss bestimmte Kriterien erfüllen, um eine sinnvolle Arbeit mit den Daten zu ermöglichen. Die wichtigsten Aspekte sind:

- Flexibles Koordinatensystem zur Darstellung der Daten
- Hinzufügen und Löschen von Daten
- Filtern der Daten mit flexibler Eingabe der Parameter
- Editieren der Daten zum Identifizieren auffälliger Objekte
- Neue Daten von NASA anfordern mit entsprechender Parametereingabe
- Speichern des Arbeitsfortschritts für flexibles Arbeiten

Die GUI und die Bedienung sollten intuitiv sein. Falsche Benutzereingaben sollten abgefangen werden und entsprechende Warnungen und weitere Informationen sollten dem Benutzer angezeigt werden.

4.2 Fenster und Elemente

Für die Erstellung des Fensters wird die Bibliothek „tkinter“ verwendet. Diese bietet die Möglichkeit, ein Fenster mit verschiedenen Elementen zu erstellen. Das Fenster kann benannt

werden und bestimmte Eigenschaften, wie z.B. die Größe, können festgelegt werden.
Die zur Verfügung stehenden Elemente reichen von Buttons über Texte bis hin zu Texteingabefeldern.

```
#Hauptfenster
root = tk.Tk()
root.title("TESS-NASA-Dataploter")
root.protocol("WM_DELETE_WINDOW", root.quit)
root.minsize(1616, 909)
```

Listing 4.1: Code zur Erstellung des Fensters

Im Programmausschnitt 4.1 wird das Hauptfenster erstellt und benannt. Außerdem wird die Funktion hinzugefügt, das bei dem Schließen des Fensters das Programm beendet wird. Es wird auch eine minimale Fenstergröße festgelegt, die wichtig wird, sobald alle Elemente hinzugefügt wurden.

Es gibt drei verschiedene Möglichkeiten, Elemente hinzuzufügen:

- „.pack“ - Bei dieser Variante werden die Elemente einfach untereinander gepackt. Es besteht wenig Kontrolle über die genaue Positionierung der Elemente.
- „.grid“ - Bei dieser Variante wird das Fenster in ein Raster unterteilt, in dem die Elemente positioniert werden können. Durch das Raster ist eine genauere Positionierung möglich. Das Raster selbst kann jedoch kaum weiter bearbeitet werden.
- „.place“ - Bei dieser Variante werden die Elemente nach ihren Pixeln angeordnet. Dies ist die genaueste, aber auch die umständlichste, Möglichkeit.

Es ist nicht möglich, innerhalb eines Fensters verschiedene Arten der Platzierung von Elementen zu verwenden. Es ist jedoch möglich, einen Frame zu erstellen, der selbst mit einem Typ platziert wird, und die Elemente innerhalb des Frames mit einem anderen Typ zu platzieren. Für die hier zu erstellende GUI bietet sich eine Kombination aus „.pack“ für die übergeordneten Frames und mit „.grid“ ein Raster innerhalb der Frames an. So können verschiedene Bereiche erzeugt werden, in denen die jeweiligen Elemente mit dem Raster angeordnet werden.

4.2.1 Koordinatensystem

Das Koordinatensystem zur Darstellung der Datenpunkte ist ein weiteres Element, das hinzugefügt werden muss. Dies wird durch die Bibliothek „matplotlib“ ermöglicht. Mit dem Zusatzmodul „FigureCanvasTKAgg“ kann das bereits in Kapitel 3 verwendete Koordinatensystem in ein Fenster eingebettet werden.

```
fig, ax = plt.subplots()
fig.set_size_inches(12,8)

#Diagramm Fenster
interact_frame = tk.Frame(root)
info_frame = tk.Frame(root)

#Übergeordnete Frames
diagram_frame = tk.Frame(interact_frame)
canvas = FigureCanvasTkAgg(fig, master = diagram_frame)
toolbar = NavigationToolbar2Tk(canvas)
canvas.get_tk_widget().pack()
```

Listing 4.2: Code um das Koordinatensystem einzubinden

Das Koordinatensystem muss dabei in einen separaten Frame ausgelagert werden, da dieses und die Toolbar nur über „.pack“ eingebunden werden können. Die anderen Elemente im „interact_frame“ werden jedoch für eine intuitive Anordnung mit dem Raster platziert.
Die anderen Elemente werden benötigt, um mit den Daten im Koordinatensystem arbeiten zu können. Sie werden in Kapitel 4.2.3 behandelt.

4.2.2 Daten einbinden

Es muss nun möglich sein, beliebige Daten aus dem Kapitel 3 einzubinden. Dazu werden entsprechende Buttons und Eingabefelder für die Filterparameter benötigt.

```
#Beispielhafte Elemente
elin_label = tk.Label(info_frame, text="Elins Daten auswählen und filtern:")
elin_button = tk.Button(info_frame, text="Elins Daten hinzufügen", command=
    add_plot_elin)
elin_flt_ra_input = tk.Entry(info_frame, bd=5, width=5)
elin_flt_ra_input.insert(0, 0.07)
empty_space_2 =  tk.Label(info_frame, height= 1)

elin_label.grid(row= 2, column= 0)
elin_button.grid(row= 2, column= 1)
elin_flt_ra_input.grid(row= 3, column= 1)
empty_space_2.grid(row= 4, column=0)
```

Listing 4.3: Beispielhafter Code um Elemente einzubinden

Diese Elemente werden zunächst wie in dem Code 4.3 angelegt. Dabei erhalten sie ihre Eigenschaften wie das zugehörige Fenster, eingebettete Texte oder die auszuführende Funktion. Im

zweiten Schritt werden diese Elemente dann an der gewählten Stelle im Raster platziert. Folgende Elemente werden für die Datenverarbeitung benötigt:

- Ein beschreibender und ein variabler Text, um Statusmeldungen anzuzeigen
- Ein Text und Button, um Daten aus dem Bildauswertungsprogramm hinzuzufügen
- Mehrere Texte, drei Eingabefelder (RA, DEC, Punkte) und ein Button, um die Daten aus dem Bildauswertungsprogramm zu filtern
- Mehrere Texte, vier Eingabefelder (RA, DEC, Punkte, Tage) und ein Button, um NASA-Daten einzufügen und zu filtern
- Ein Beschreibungstext, ein Fortschrittsbalken und ein variabler Text, um den Fortschritt anzuzeigen

Den Buttons ist jeweils eine Funktion zugeordnet, die bei Betätigung ausgeführt wird. Die Eingabefelder erhalten einen Standardwert, der beim Start eingetragen wird. Dieser ist ein Richtwert für eine sinnvolle Eingabe.
Werden alle Elemente zusammengefügt, entsteht ein Fenster mit einem Koordinatensystem und darunter mit den einzelnen Texten, Buttons und Eingabefeldern. Dabei ist der Aufbau der Elemente für ein intuitives Arbeiten mit der GUI zeilenweise realisiert.

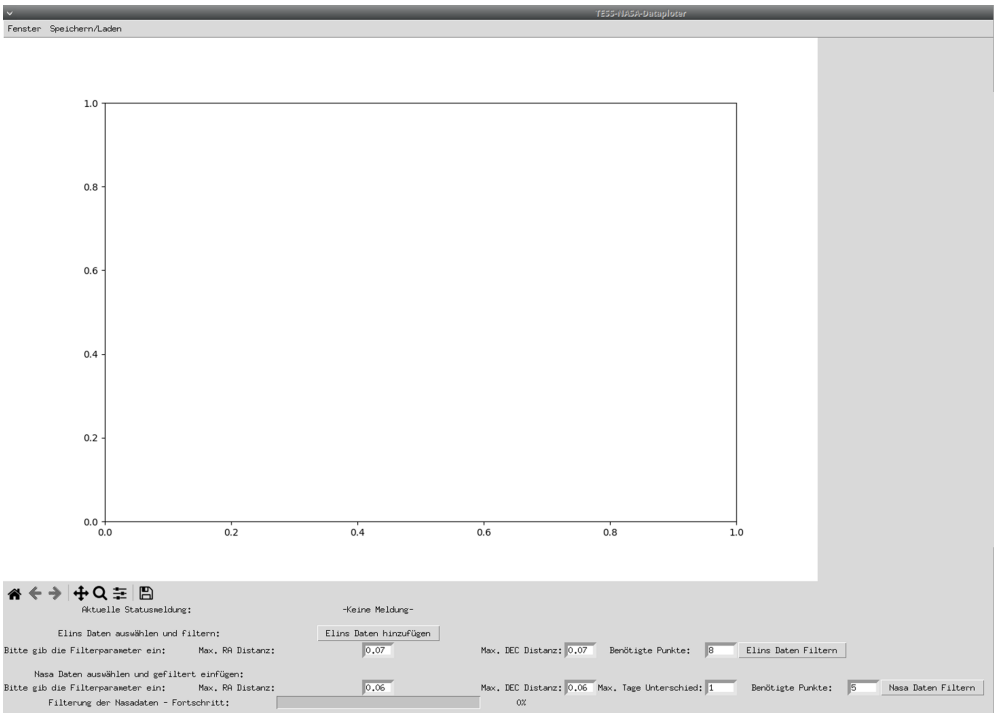

Abbildung 4.1: GUI mit Koordinatensystem und den Elementen um Daten hinzuzufügen

Daten aus dem Bildauswertungsprogramm

Um die Daten aus dem Bildauswertungsprogramm einzufügen, muss die entsprechende Datei mit den Daten ausgewählt werden. Anschließend werden die Datenpunkte mit der Funktion 3.1 ausgelesen und können in das Koordinatensystem eingefügt werden.

```
#Funktion zum Hinzufügen von Daten von Elins Auswertung
def add_plot_elin():
    try:
        file_name = select_file()
    except:
        infotext_info.config(text="Datei konnte nicht aufgerufen werden!")
        return -1

    try:
        data_elin = fnkt.data_read_elin(file_name)
    except:
        infotext_info.config(text="Datei entspricht nicht Elins-Standard!")
        return -1

    #ggf alte Daten entfernen und neue Daten abspeichern
```

```
16      all_data.clear()
17      liste.delete(0,'end')
18      all_data.append(data_elin)
19
20      draw_data()
```

Listing 4.4: Funktion um Daten des Bildauswertungsprogrammes einzufügen

Zuerst wird die gewünschte Datei aufgerufen. Dazu wird die Funktion „select_file()“ verwendet, die in 4.5 beschrieben ist. Die ausgewählte Datei wird mit der Funktion aus 3.1 gelesen. In einem Fehlerfall wird die Funktion beendet und der Infotext angepasst. Wurden die Daten erfolgreich ausgelesen, werden sie in einer separaten Liste mit allen anzuzeigenden Daten gespeichert. Zuletzt wird die Funktion „draw_data“ aufgerufen, die in 4.6 erklärt wird.

Datei auswählen

Um die Dateiauswahl intuitiv zu gestalten, wird die Bibliothek „tkinter.filedialog“ verwendet. Diese ruft ein separates Dateiverwaltungsfenster auf, in dem die entsprechende Datei ausgewählt werden kann.

```
1  #Funktion zum Auswählen einer Datei
2  def select_file():
3      filetypes = (
4          ('Nasa files', '*.txt *.tnd'),
5          ('All files', '*.*')
6      )
7
8      filename = fd.askopenfilename(
9          title='Open a file',
10         initialdir=file_dir,
11         filetypes=filetypes)
12
13     replace_dir(filename)
14
15     return filename
16
17 #Funktion zur Abspeicherung des Dateipfades
18 def replace_dir(filename):
19     filepart = filename.split('/')
20     filepart.remove(filepart[-1])
21     filepart.append('')
22     delimiter = '/'
23     global file_dir
```

```
24      file_dir = delimiter.join(filepart)
```

Listing 4.5: Funktionen für die Auswahl einer Datei

Die möglichen Datentypen werden über die Variable „filetypes“ angegeben. Die entsprechende Funktion wird dann über die Bibliothek aufgerufen. Dabei werden ein Titel, der Dateipfad und die Datentypen übergeben.
Wählt der Benutzer nun eine Datei aus, gibt die Funktion den kompletten Dateipfad der ausgewählten Datei zurück. In einer separaten Funktion wird nun der Name aus dem Dateipfad entfernt und der Pfad wird gespeichert. So wird bei dem nächsten Aufruf der Funktion „select_file“ direkt der vorherige Dateipfad geöffnet.
Am Ende wird der vollständige Name zurückgegeben und kann entsprechend weiterverarbeitet werden.

Daten zeichen

Um beliebige Daten darstellen zu können, wird das Zeichnen dieser Daten in eine separate Funktion ausgelagert. Somit kann immer die gleiche „draw_data“ Funktion aufgerufen werden, unabhängig davon, welche Daten hinzugefügt wurden.

```
1  #Funktion um die Daten darzustellen
2  def draw_data(keep_axis = False):
3      xlim_min = ax.get_xlim()[0]
4      xlim_max = ax.get_xlim()[1]
5      ylim_min = ax.get_ylim()[0]
6      ylim_max = ax.get_ylim()[1]
7
8      ax.clear()
9      count = 0
10
11     #Daten aus der Liste nacheinander darstellen
12     for point in all_data:
13         xValues = []
14         yValues = []
15
16         for row in point:
17             xValues.append(float(row[1]))
18             yValues.append(float(row[2]))
19         try:
20             ax.scatter(xValues, yValues,s=1, color = color_order[count])
21         except:
22             ax.scatter(xValues, yValues,s=1, color = 'blue')
```

```
23          count += 1
24
25      #ggf alten Bildausschnitt beibehalten
26      if(keep_axis == True):
27          ax.set_xlim(xmin=xlim_min, xmax=xlim_max)
28          ax.set_ylim(ymin=ylim_min, ymax=ylim_max)
29
30      canvas.draw()
```

Listing 4.6: Funktionen zur Darstellung der Daten

Der Funktion 4.6 kann ein Parameter übergeben werden, der den aktuell ausgewählten Bildausschnitt speichert und bei der nächsten Grafikausgabe wiederherstellt. Dazu werden zunächst die Daten des aktuellen Ausschnitts gespeichert und alle vorherigen Daten werden gelöscht. Nun wird mit einer „for-Schleife“ jedes Element von „all_data“ betrachtet, um die Daten darzustellen. Das erste Element in der Liste der gesamten Daten ist der Datensatz aus dem Bildauswertungsprogramm. Danach folgen die gefilterten Daten des Bildauswertungsprogramms. Als letztes Element werden die NASA-Daten gespeichert.
Im nächsten Schritt werden mit einer weiteren „for-Schleife“ die jeweiligen Koordinaten der Objekte aus den entsprechenden Datensätzen extrahiert. Diese werden dann nacheinander angezeigt. Die Farbe wird dabei aus einer vordefinierten Reihenfolge entnommen.
Abschließend wird ggf. der zuvor gespeicherte Bildausschnitt wiederhergestellt. Anschließend werden mit der Funktion „canvas.draw“ die neu gespeicherten Daten in der GUI dargestellt.

Gefilterte Daten des Bildauswertungsprogramm

Um nun die Daten aus dem Bildauswertungsprogramm zu filtern und in die GUI einzufügen, werden mehrere Elemente in der GUI und eine entsprechende Funktion benötigt. Für die Filterung werden zusätzlich Eingabefelder für die Filterparameter benötigt. Durch Betätigung des entsprechenden Buttons wird die entsprechende Funktion aufgerufen.

```
#Funktion um die Filterung von Elinsdaten hinzuzufügen
def add_flt():
    #Überprüfen aller Filterparamenter
    try:
        ra = float(elin_flt_ra_input.get())
        dec = float(elin_flt_dec_input.get())
        point = int(elin_flt_point_input.get())
    except:
        infotext_info.config(text="Fehlerhafte Eingabe der Filterparameter!"
    )
```

```
10          return
11      if(all_data == []):
12          infotext_info.config(text="Zunächst müssen Daten von Elin geladen
    werden!")
13      elif(ra <= 0):
14          infotext_info.config(text="Der Filterparameter RA muss größer Null
    sein!")
15      elif(dec <= 0):
16          infotext_info.config(text="Der Filterparameter DEC muss größer Null
    sein!")
17      elif(point <= 0):
18          infotext_info.config(text="Die benötigten Datenpunkte müssen größer
    Null sein!")
19      else:
20          #Wenn die Bedingungen passen wird die Filterung berechnet
21          data = all_data[0]
22          if(len(all_data) < 2):
23              all_data.append(fnkt.chain_finder(data, ra, dec, point))
24          else:
25              all_data.clear()
26              all_data.append(data)
27              all_data.append(fnkt.chain_finder(data, ra, dec, point))
28              liste.delete(0,'end')
29          #Filterung wird dargestellt
30          draw_data()
```

Listing 4.7: Funktionen zur Darstellung der Daten

Das Wichtigste an der Funktion 4.7 ist, dass mögliche Fehler abgefangen werden. Zunächst müssen die Eingaben in den Feldern überprüft werden. Bei den Koordinaten muss es sich um eine Gleitkommazahl und bei den Punkten um eine ganze Zahl handeln. Außerdem müssen alle Eingaben größer als Null sein. Zudem müssen die Daten erst aus dem Bildauswertungsprogramm geladen werden, bevor sie gefiltert werden können.
Sind alle Bedingungen erfüllt, wird die Funktion 3.3 aufgerufen. Dabei werden die Daten aus der Variable „all_data“ entnommen und zusammen mit den eingelesenen Filterparametern an die Funktion übergeben. Die gefilterten Daten, die die Funktion zurückgibt, werden als zweites Element in der Liste mit allen Daten gespeichert. Abschließend wird die Funktion „draw_data‘ aufgerufen, um auch die neuen Daten darzustellen.

Gefilterte NASA-Daten

Bei den noch fehlenden Daten handelt es sich um die korrelierenden NASA-Daten. Diese Daten müssen in Bezug auf die Daten, die bereits hinzugefügt wurden, gefiltert werden. Für die entsprechenden Filterparameter sind ebenfalls Eingabefelder in der GUI vorgesehen. Hier können der maximal erlaubte Abstand von Rektaszension und Deklination, die maximal entfernte Zeitspanne sowie die minimal benötigten Punkte eingegeben werden.

```
#Nasadaten auswählen
try:
    datei_name = select_file()
except:
    infotext_info.config(text="Datei konnte nicht aufgerufen werden!")
    return -1
#Filterung der Nasadaten
try:
    data_nasa = fnkt.data_compare(all_data[1], datei_name, point, day, ra,
    dec, progressbar_funk)
except:
    infotext_info.config(text="Datei entspricht nicht dem Standard!")
    return

#gefilterte Nasadaten abspeichern
nasa_data.clear()

data_nasa_raw = []
for objekt in data_nasa:
    nasa_data.append(objekt)
    for row in objekt:
        if row != objekt[0]:
            data_nasa_raw.append(row)

#Nasadaten in der Liste darstellen
liste.delete(0,'end')
list_add_nasa()

if(len(all_data) < 3):
    all_data.append(data_nasa_raw)
else:
    all_data[2] = data_nasa_raw

draw_data()
```

Listing 4.8: Ausschnit aus der Funktionen zur Filterung und Anzeige der NASA-Daten

In dem Funktionsausschnitt 4.8 ist nicht dargestellt, dass zunächst, wie bei der Funktion 4.7, die eingegebenen Parameter überprüft werden. Es wird geprüft, ob eine Gleitkommazahl (für RA, DEC, Tage) bzw. eine Ganzzahl (für die Punkte) eingegeben wurde. Außerdem müssen diese Werte alle größer Null sein.
Im nächsten Schritt muss die Datei mit den NASA-Daten ausgewählt werden. Dazu wird wiederum die Funktion „select_file“ verwendet. Nach Auswahl der Datei wird die Funktion aus Kapitel 3.3.2 mit den Filterparametern aufgerufen. Die von der Funktion zurückgegebenen Daten können jedoch nicht von der Funktion „draw_data“ verarbeitet werden, da die Listenstruktur noch eine andere ist. Daher müssen zunächst alle Punkte ohne Objektzugehörigkeit gespeichert werden.
Anschließend wird die Funktion „list_add_nasa“ aufgerufen, die in Kapitel 4.2.3 erläutert wird. Als letzter Schritt werden die NASA-Daten gespeichert und die Daten neu gezeichnet.

Anzeige des Fortschrittes

Da die Filterung der NASA-Daten einige Zeit in Anspruch nehmen kann, ist es sinnvoll, dem Benutzer den zeitlichen Fortschritt anzuzeigen. Ansonsten könnte der falsche Schluss gezogen werden, dass das Programm abgestürzt ist. Für die Implementierung wird ein Text und ein Fortschrittsbalken verwendet. Für die Anzeige wird der in „tkinter“ integrierte Fortschrittsbalken verwendet. Dieser wird mit der Funktion „tkk.Progressbar“ erzeugt.

```
1  #Elemente zur Anzeige des Fortschritts
2  progress_label = tk.Label(info_frame, text="Filterung der Nasadaten -
       Fortschritt:")
3  progressbar = ttk.Progressbar(info_frame, orient="horizontal", length=300,
       mode="determinate")
4  progress_status_label = tk.Label(info_frame, text = "0%")
5  #Variable zur Übergabe der Daten für den Fortschritt
6  progressbar_funk = [progressbar, root, progress_status_label]
7
8  progress_label.grid(row=7, column= 0)
9  progressbar.grid(row= 7, column= 1)
10 progress_status_label.grid(row= 7, column= 2)
```

Listing 4.9: Code um die Elemente zum Anzeigen des Fortschritts einzubinen

Um den Statustext und den Fortschrittsbalken zu ändern, müssen die Parameter der erstellten Elemente geändert werden. Dazu müssen die Elemente der Funktion per Call-by-Reference übergeben werden. Dies kann über eine Liste der Elemente realisiert werden. Dadurch erhält die Funktion die Möglichkeit, die Eigenschaften der Elemente zu verändern. Außerdem muss

das Hauptfenster übergeben werden, da dieses aktualisiert werden muss, um den Fortschritt anzuzeigen.
Für die Implementierung in der Funktion 3.13 wird die Schleife in Zeile 4 wie folgt geändert:

```
1 for sb in database_list:
2         #Fortschritt anzeigen
3         if(process != []):
4             process[0]['value'] = (number/len(database_list))*100
5             process[2].config(text= str(round((number/len(database_list))
      *100,2))+"%")
6             process[1].update_idletasks()
7         number += 1
```

Listing 4.10: Code für die Anzeige des Fortschritts

Die Liste „progressbar_funk“ wird der Funktion übergeben. Innerhalb der Schleife wird der Wert für den Fortschrittsbalken berechnet und verändert. Der Wert setzt sich aus dem Durchlauf der Schleife und der Anzahl der benötigten Schleifendurchläufe zusammen. Ebenso wird der Statustext angepasst, der den prozentualen Fortschritt in Textform angibt. Zuletzt wird mit „update_idletasks“ das Fenster neu geladen und der Fortschritt wird somit sichtbar gemacht.
Am Ende der Funktion 4.8 müssen nun die Eigenschaften der Fortschrittsanzeige zurückgesetzt werden. Dadurch kann eine erneute Filterung der NASA-Daten angezeigt werden.

4.2.3 Daten bearbeiten

Um mit den hinzugefügten Daten sinnvoll arbeiten zu können, werden verschiedene Funktionalitäten benötigt. Die Daten müssen editierbar gemacht werden, damit die auffälligen Punkte hervorgehoben werden können.

Datensätze entfernen

Eine der wichtigsten Funktionen ist das Löschen von bereits hinzugefügten Datensätzen. Dies betrifft die Daten aus dem Bildauswertungsprogramm, die gefilterten Daten sowie die NASA-Daten. Daher werden drei Buttons benötigt, die jeweils einen der Datensätze entfernen. Diesen Buttons wird eine entsprechende Funktion hinterlegt, die den entsprechenden Datensatz löscht. Dabei ist zu beachten, dass jeder Datensatz, der von einem anderen abhängig ist, ebenfalls gelöscht wird. Dies bedeutet wenn die Filterung gelöscht wird, müssen die NASA-Daten ebenfalls entfernt werden und wenn die Daten des Bildauswertungsprogramms gelöscht werden, müssen sowohl die Filterung als auch die NASA-Daten ebenfalls entfernt werden.

```
1 #Funktion zum Entfernen der Datensätze
```

```
2  def remove_data(data):
3      #Nasadaten sollen immer entfernt werden
4      try:
5          del_data = all_data[2]
6          all_data.remove(del_data)
7      except:
8          if data == 2:
9              infotext_info.config(text="Nasadaten konnten nicht entfernt
       werden.")
10     if(data < 2):
11         #ggf auch Elins gefilterte Daten entfernen
12         try:
13             del_data = all_data[1]
14             all_data.remove(del_data)
15         except:
16             if data == 1:
17                 infotext_info.config(text="Gefilterte Daten konnten nicht
       entfernt werden.")
18         if(data < 1):
19             #ggf auch Elins Datensatz entfernen
20             try:
21                 del_data = all_data[0]
22                 all_data.remove(del_data)
23             except:
24                 infotext_info.config(text="Elins Daten konnten nicht
       entfernt werden.")
25     #Liste anpassen und neu zeichnen
26     liste.delete(0,'end')
27     draw_data()
```

Listing 4.11: Funktion zur Entfernung von Datensätzen

Da Daten immer einen weiteren Datensatz bedingen, können diese linear nacheinander gelöscht werden. Somit ist egal welcher Button gedrückt wird, die NASA-Daten müssen immer entfernt werden. Da eventuell noch keine NASA-Daten hinzugefügt wurden, muss dies mit einer „try: - except:“ Anweisung geprüft werden. Falls keine Daten gelöscht werden konnten und der Button zum Löschen der NASA-Daten gedrückt wurde, wird eine Warnung ausgegeben. Für die nächsten beiden Datensätze wird zunächst abgefragt, ob diese ebenfalls gelöscht werden sollen. Ist dies der Fall, wird mit ihnen genauso verfahren wie mit den NASA-Daten.
In dem letzten Schritt wird das Koordinatensystem neu gezeichnet und damit die gewünschten Daten gelöscht.

Liste der NASA-Daten

Ein weiterer wichtiger Aspekt für die Arbeit mit den Daten ist die Handhabung der NASA-Daten. Es muss erkennbar sein, welche Objekte aus den NASA-Daten hinzugefügt wurden. Die Zugehörigkeit muss erkennbar sein und ausgewählte Objekte müssen gelöscht werden können. Zur Identifizierung und Zuordnung der Objekte bietet sich eine Liste aller Elemente der NASA-Daten an. Dazu kann die im „tkinter“ integrierte Listbox verwendet werden. Diese wird mit der Funktion „liste = tk.Listbox(interact_frame, height= 50)“ erzeugt und wie die anderen Elemente im Raster platziert.
Dieser Liste müssen nun die Objektnamen aus den NASA-Daten hinzugefügt werden. Die Funktion hierfür wird von der Funktion 4.8 aufgerufen. Damit werden die Namen zusammen mit den NASA-Daten entsprechend hinzugefügt.

```
#Funktion zum Hinzufügen von Elementen zur Liste
def list_add_nasa():
    for point in nasa_data:
        liste.insert("end", point[0])
```

Listing 4.12: Funktion um Objekte in die Liste hinzuzufügen

In der Variable „nasa_data“ werden alle NASA-Daten zwischengespeichert. Von jedem Objekt wird das erste Element, welches der Objektname ist, der Liste hinzugefügt.
Im nächsten Schritt muss die Zugehörigkeit der NASA-Daten zur Liste dargestellt werden. Dazu muss das ausgewählte Element der Liste in dem Koordinatensystem hervorgehoben werden. Hierfür muss die Auswahl eines Elementes in der Liste mit einer Funktion verknüpft werden. Dies ist über „liste.bind(«<ListboxSelect»", list_show)“ möglich.

```
#Funktion um die aktulle Auswahl darzustellen
def list_show(event = []):
    try:
        seltect = liste.curselection()[0]
    except:
        return

    #Zunächst die Daten neu darstellen und den Bildausschnitt beibehalten
    draw_data(True)

    #Nun die ausgewählten Daten zusätzlich darstellen
    xValues = []
    yValues = []
    for row in nasa_data[seltect]:
        if row != nasa_data[seltect][0]:
```

```
16                 xValues.append(float(row[1]))
17                 yValues.append(float(row[2]))
18         ax.scatter(xValues, yValues,s=1, color = 'red')
19
20         canvas.draw()
```

Listing 4.13: Funktion zur Hervorhebung des ausgewählten Objektes der Liste

Wird ein neues Element ausgewählt, wird zunächst geprüft, welches Element ausgewählt wurde. Da die Anordnung innerhalb der Liste die gleiche ist wie in den zwischengespeicherten NASA-Daten, kann so festgestellt werden, welches Objekt in der Liste ausgewählt ist. Um ggf. eine alte Auswahl zu entfernen, werden die Daten zunächst mit der Funktion 4.6 neu gezeichnet. Dabei wird der Bildausschnitt beibehalten. Anschließend werden die Datenpunkte des markierten Objekts in roter Farbe darüber gezeichnet. Dadurch entsteht eine Hervorhebung des ausgewählten Objektes.

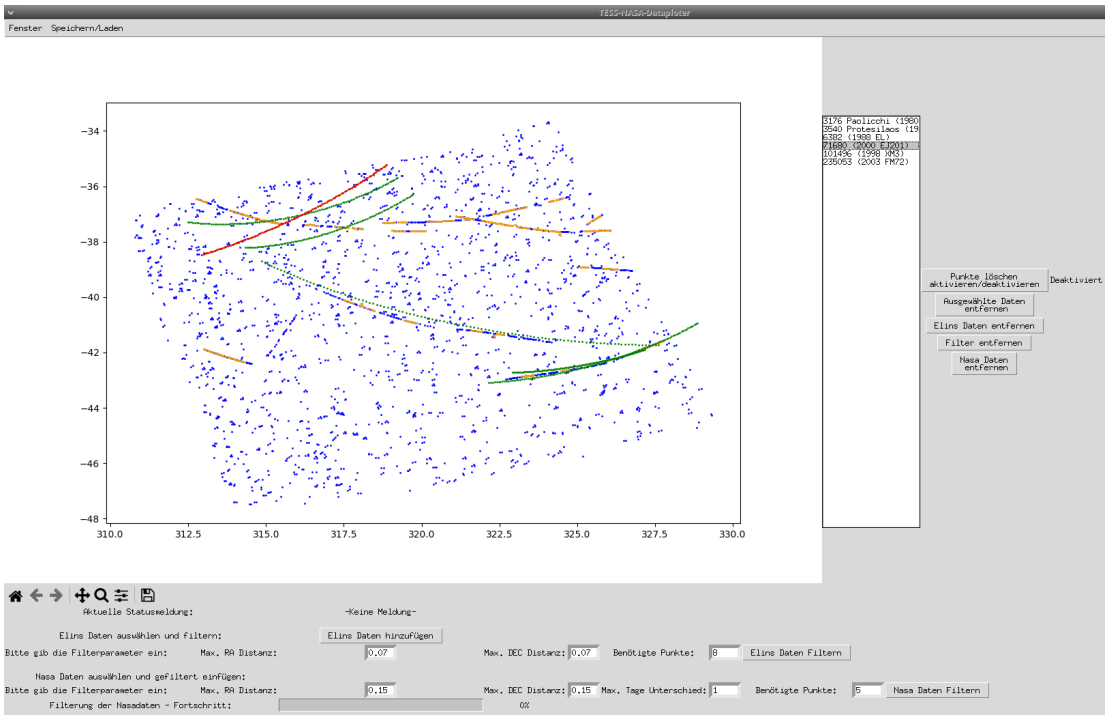

Abbildung 4.2: Liste mit ausgewähltem und hervorgehobenem Objekt

Neben der Hervorhebung von Objekten in den NASA-Daten muss es möglich sein, Objekte aus der Liste und dem Koordinatensystem zu entfernen. Dazu wird ein Button mit einer

entsprechenden Funktion erstellt, mit der das ausgewählte Objekt entfernt werden kann. Besagter Button ist in der Abbildung 4.2 rechts zu sehen.

```
#Funktion zum entfernen der ausgewählten Nasadaten
def remove_nasa_sel():
    try:
        seltect = liste.curselection()[0]
    except:
        infotext_info.config(text="Es sind aktuell keine Daten ausgewählt.")
        return

    #Element aus der Liste und aus den abgespeicherten Daten entfernen
    remove_data = nasa_data[seltect]
    liste.delete(liste.get(0, 'end').index(remove_data[0]))
    nasa_data.remove(remove_data)
    remove_data.remove(remove_data[0])

    #Punkte aus dem gesammten Datensatz entfernen
    for element in remove_data:
        all_data[2].remove(element)
    draw_data(True)
```

Listing 4.14: Funktion um das ausgewählten Objektes zu löschen

Um das hervorgehobene Element zu löschen, muss zunächst erneut das ausgewählte Element aus der Liste ermittelt werden. Das entsprechende Objekt wird dann mit allen zugehörigen Punkten aus der Variable mit allen NASA-Daten ausgelesen und in einer Löschvariable gespeichert. Über den Namen und dem daraus resultierenden Index wird das Objekt aus der Liste entfernt. Aus der Variable mit den gesamten NASA-Daten wird das Objekt mit der Funktion „.remove“ entfernt. Um die Datenpunkte aus der Variable mit allen Daten zu löschen, muss zuerst der Name des Objekts aus der Löschvariable entfernt werden. Danach muss mit einer „for-Schleife“ jeder Punkt des Objekts nacheinander gelöscht werden. Zuletzt wird das Koordinatensystem neu gezeichnet und das ausgewählte Objekt gelöscht.

Punkte entfernen

Die korrelierenden NASA-Daten können nun identifiziert und gegebenenfalls entfernt werden. Es fehlt jedoch noch die Möglichkeit, Datenpunkte aus dem Bildbearbeitungsprogramm zu entfernen, so dass im besten Fall schrittweise die für die NASA noch unbekannten Objekte ermitteln werden können. Dabei ist es sinnvoll, dass über die Position des Cursors ermittelt wird, welcher Punkt entfernt werden soll. Dazu wird aus der „matplotlib“ die „MouseButton“

Bibliothek hinzugefügt. Außerdem wird mit „plt.connect('button_press_event', on_click)“ die Funktion „on_click“ an das Drücken der Maustaste gebunden.

```
#Funktion zum Entfernen der angeklickten Datenpunkte
def on_click(event):
    if event.inaxes and delete_activ[0] == True and all_data != []:
        RA = event.xdata
        DEC = event.ydata
        delete_point = all_data[0][0]
        for point in all_data[0]:
            RA_diff_cur = abs(float(delete_point[1])- RA)
            DEC_diff_cur = abs(float(delete_point[2])- DEC)
            diff_cur = sqrt(RA_diff_cur**2+DEC_diff_cur**2)

            RA_diff_new = abs(float(point[1])- RA)
            DEC_diff_new = abs(float(point[2])- DEC)
            diff_new = sqrt(RA_diff_new**2+DEC_diff_new**2)
            if(diff_new < diff_cur):
                delete_point = point
        all_data[0].remove(delete_point)
        draw_data(True)
```

Listing 4.15: Funktion, um den Punkt zu löschen, der dem Cursor am nächsten liegt

Zuerst wird geprüft, ob sich der Mauszeiger bei gedrückter Maustaste innerhalb des Koordinatensystems befindet. Weiterhin wird geprüft, ob das Löschen aktiviert ist, was in der Funktion 4.16 erklärt wird. Außerdem müssen bereits Daten vom Bildauswertungsprogramm eingefügt worden sein.
Wenn alle diese Bedingungen erfüllt sind, werden die Koordinaten des Mauszeigers gespeichert. Anschließend wird mit Hilfe einer „for-Schleife“ der nächstgelegene Punkt ermittelt. Anschließend wird dieser Punkt aus der Variable mit allen Daten entfernt und das Koordinatensystem neu gezeichnet.

Damit die Punkte nur zu den von dem Benutzer gewünschten Zeitpunkten gelöscht werden, muss diese Funktion über eine separate Funktion aktivierbar und deaktivierbar sein. Dazu wird ein Button und ein Text erstellt. Der Button wechselt zwischen den Zuständen „Aktiviert“ und „Deaktiviert“ und der zugehörige Text kennzeichnet diesen Zustand.

```
#Funktion zum Aktivieren/Deaktivieren der Entfernenfunktion
def activ_del():
    if(delete_activ[0] == False):
        delete_activ[0] = True
```

```
5           delete_status.config(text="Aktiviert")
6       else:
7           delete_activ[0] = False
8           delete_status.config(text="Deaktiviert")
```

Listing 4.16: Funktion zur Aktivierung/Deaktivierung der Entfernenfunktion

Hier wird bei dem Anklicken des Buttons zunächst geprüft, welcher Status gerade vorherrscht. Je nachdem ändert sich der Wert der Variable auf Wahr oder Falsch und der Statustext wird ebenfalls angepasst.
Auf diese Weise kann der Anwender nach Bedarf Datenpunkte aus den Daten des Bildverarbeitungsprogramms entfernen.

Speichern und Laden

Um die Arbeit mit den Daten zeitweise unterbrechen zu können, ist es auch wichtig, die Daten speichern und laden zu können. Für die Umsetzung ist zu klären, welche Daten gespeichert werden müssen, um zu einem späteren Zeitpunkt mit den gleichen Daten weiterarbeiten zu können.
Zunächst werden die drei Datensätze der Variable „all_data“ benötigt. Diese entsprechen einmal den Daten aus dem Bildauswertungsprogramm, den daraus gefilterten Daten und den hinzugefügten NASA-Daten. Zusätzlich werden die Informationen aus der Variable „nasa_data“ benötigt. Diese speichert die Zugehörigkeit der Punkte aus den NASA-Daten in „all_data“ zu den einzelnen Objekten der NASA-Daten.
Für eine intuitive Arbeitsumgebung bietet sich das Menüband am oberen Rand für die Buttons der Funktionen an.

```
1  #Menüleiste
2  menubar = tk.Menu(root)
3  window_menu = tk.Menu(menubar, tearoff=0)
4  save_menu = tk.Menu(menubar, tearoff=0)
5
6  #Fenster - Menüleiste
7  window_menu.add_command(label="Diagramm", command=lambda:[nasa_req_frame.
       pack_forget(),build_diagramm_screen()])
8  window_menu.add_command(label="Nasa", command=lambda:[del_diagramm_screen(),
       nasa_req_frame.pack(side=tk.TOP, fill='x')])
9
10 #Speichern/Laden - Menüleiste
11 save_menu.add_command(label="Speichern", command=speichern)
12 save_menu.add_command(label="Laden", command=laden)
```

```
13
14 menubar.add_cascade(label="Fenster", menu=window_menu)
15 menubar.add_cascade(label="Speichern/Laden", menu=save_menu)
16 root.config(menu=menubar)
```

Listing 4.17: Code zur Einrichtung der Menüleiste

Die Menüleiste enthält einmal ein Register mit „Speichern/Laden“ und ein Register zum Wechseln zwischen den Fenstern, was in Kapitel 4.2.4 erläutert wird. Im Register „Speichern/Laden“ gibt es dann die beiden Menüpunkte „Speichern“ und „Laden“ mit den jeweiligen Funktionen.

```
1  #Funktion zum Abspeichern der Daten
2  def speichern():
3      if file_dir != '/':
4          #Dateiname mit Zeitstempel erstellen
5          current_time = dt.datetime.now()
6          current_time_str = str(current_time.strftime("%Y-%m-%d_%H:%M:%S"))
7          file_name = (file_dir+'TND-Data_'+current_time_str+'.tnd')
8
9          #Datei erstellen und Daten abspeichern
10         datei = open(file_name,'w')
11         datei.write('TESS-NASA-Dataploter_Speicherstand_'+current_time_str+'
       \r')
12         #Alle Datenpunkte abspeichern
13         for data in all_data:
14             for point in data:
15                 for element in point:
16                     datei.write(element+' ')
17                 datei.write(';')
18             datei.write('\r')
19         datei.write('NASA-DATEN:\r')
20         #Nasadaten abspeichern
21         for objekt in nasa_data:
22             for point in objekt:
23                 if point == objekt[0]:
24                     datei.write(point)
25                 else:
26                     for element in point :
27                         datei.write(element+' ')
28                 datei.write(';')
29             datei.write('\r')
30         datei.close
31         infotext_info.config(text="Datei wurde erfolgreich abgespeichert.")
32     else:
```

```
33          infotext_info.config(text="Keine Daten zum abspeichern vorhanden.")
34          return
```

Listing 4.18: Funktion, um die aktuellen Daten zu speichern

Zum Speichern der Daten wird zunächst geprüft, ob bereits Daten vorhanden sind und ob das richtige Dateiverzeichnis hinterlegt ist. Im hinterlegten Verzeichnis wird eine neue Datei angelegt, dessen Name setzt sich aus dem Text „TND-Data_“, dem aktuellen Datum mit Uhrzeit und der Dateiendung „.tnd“ zusammen.
Die Datei wird nun aufgerufen und beschrieben. Eine Beschreibung und der Zeitstempel werden in die erste Zeile geschrieben. Danach folgen in den nächsten ein bis drei Zeilen (je nach Anzahl der hinzugefügten Daten) die Datensätze aus der Variable mit allen Daten. Die einzelnen Punkte innerhalb einer Zeile werden durch Semikolons getrennt und die Werte innerhalb der Punkte werden durch ein Leerzeichen getrennt.
Um die NASA-Daten mit der Objektzugehörigkeit zu speichern, werden die folgenden Zeilen verwendet. Zunächst wird zur Trennung der Text „NASA-DATEN:“ in die nächste Zeile geschrieben. In jeder folgenden Zeile wird ein Objekt aus den NASA-Daten mit Namen und zugehörigen Punkten gespeichert. Dazu wird der Name an den Anfang der Zeile geschrieben. Es folgt ein Semikolon als Trennzeichen. Danach folgen die zum Objekt gehörenden Punkte, ebenfalls durch Semikolons getrennt. Die Werte der Punkte werden, wie bei den Datensätzen, durch Leerzeichen getrennt. Sobald alle Objekte mit allen Punkten gespeichert sind, wird die Datei geschlossen und eine entsprechende Statusmeldung ausgegeben.

Um die Daten wieder laden zu können, müssen die Variablen nun mit den Informationen aus der Datei beschrieben werden.

```
1  #Funktion zum Laden von abgespeicherten Daten
2  def laden():
3      #Datei aufrufen
4      try:
5          datei_name = select_file()
6          datei = open(datei_name, 'r')
7      except:
8          infotext_info.config(text="Datei konnte nicht aufgerufen werden!")
9          return
10
11     #Datei auf Standard überprüfen
12     line = datei.readline()
13     if line.find("TESS-NASA-Dataploter_Speicherstand") != 0:
```

```
14          infotext_info.config(text="Datei entspricht nicht dem TND Standard!"
    )
15          return
16      else:
17          #Aktuelle Daten entfernen und gespeicherte Daten laden
18          all_data.clear()
19          nasa_data.clear()
20          liste.delete(0,'end')
21          #Datenpunkte auslesen
22          while True:
23              line = datei.readline()
24              if line.find('NASA-DATEN:') != 0:
25                  line_split = line.split(";")
26                  data = []
27                  for element in line_split:
28                      if len(element.split()) > 2:
29                          data.append(element.split())
30                  all_data.append(data)
31              else:
32                  break
33          #Nasadaten auslesen
34          while True:
35              line = datei.readline()
36              if line != '':
37                  line_split = line.split(";")
38                  data = []
39                  for element in line_split:
40                      if element == line_split[0]:
41                          data.append(element)
42                      else:
43                          if len(element.split()) >= 1:
44                              data.append(element.split())
45                  nasa_data.append(data)
46              else:
47                  break
48          #Liste und Darstellung erstellen
49          list_add_nasa()
50          draw_data()
```

Listing 4.19: Funktion, um einen alten Speicherstand zu laden

Um die Daten aus der Datei zu laden, muss diese zunächst ausgewählt werden. Dazu wird ebenfalls die Dateiauswahlfunktion verwendet. Es wird geprüft, ob die ausgewählte Datei eine frühere Speicherung ist. Dazu wird der Inhalt der ersten Zeile geprüft. Ist die Datei in Ordnung,

werden die Daten ausgelesen. Mit einer „while-Schleife“ werden weitere Zeilen eingelesen. Solange der Umbruch zu den NASA-Daten noch nicht ausgelesen wurde, wird der Inhalt der Zeile in seine Bestandteile zerlegt. Diese werden dann an den entsprechenden Stellen in der Variable „all_data“ gespeichert. Sobald der Umbruch erfolgt ist, wird die Schleife abgebrochen und eine neue Schleife wird ausgeführt um die Objekte der NASA-Daten einzulesen. Auch hier wird zeilenweise gelesen und geteilt. Die Objekte mit den zugehörigen Punkten werden dann in der Variable „nasa_data“ gespeichert. Sobald die Zeilen keinen Inhalt mehr haben, wird die Schleife unterbrochen.
In dem letzten Schritt werden die Objekte der Liste hinzugefügt und die eingelesenen Datensätze gezeichnet.

4.2.4 NASA-Daten anfragen

Um alle wichtigen Funktionen abzudecken, muss es in der GUI auch möglich sein, NASA-Daten über die NASA-API anzufragen.

Ansicht im anderen Fenster

Aus Gründen der Übersichtlichkeit ist es sinnvoll, die Funktion zum Anfragen der NASA-Daten in ein eigenes Fenster auszulagern. Dazu wird in der Menüleiste ein Reiter „Fenster“ mit den Unterpunkten „Diagramm“ und „Nasa“ angelegt. Diese Menüpunkte führen jeweils Funktionen aus, die Elemente aus dem jeweils anderen Fenster entfernen bzw. von dem gewünschten Fenster einfügen.

```
#Funktion zum entfernen des Diagramm Fensters
def del_diagramm_screen():
    info_frame.pack_forget()
    interact_frame.pack_forget()

#Funktion zum erstellen des Diagramm Fensters
def build_diagramm_screen():
    interact_frame.pack(side=tk.TOP, fill='x')
    info_frame.pack(side=tk.TOP, fill='x')
```

Listing 4.20: Funktionen zum Wechseln der Ansichten

Für das Hinzufügen und Entfernen von Elementen des NASA-Fensters werden nur die Befehle „nasa_req_frame.pack(side=tk.TOP, fill='x')“ und „nasa_req_frame.pack_forget()“ benötigt. Diese Befehle werden zusammen mit den Funktionen 4.20 direkt über den Button aufgerufen.

Dadurch werden die Elemente des anderen Fensters entfernt und die Elemente des gewünschten Fensters angezeigt.

Anfrage über die GUI

Für die Anfrage von NASA-Daten werden verschiedene weitere Elemente benötigt. Dazu gehören Texte, Eingabefelder zur Festlegung der Parameter, ein Button und eine Fortschrittsanzeige, da die Anfrage viel Zeit beanspruchen kann.

Abbildung 4.3: Ansicht umNASA-Daten anzufragen

Die beiden Eingabefelder werden zum einen für die zeitliche Genauigkeit der angefragten Daten und zum anderen für die Mindesthelligkeit, die die Objekte aufweisen müssen, benötigt. In dem Infotext können die einzelnen Anfragen dargestellt werden und der Fortschritt kann wiederum über den entsprechenden Balken angezeigt werden.
Durch Anklicken des Buttons wird die entsprechende Funktion aufgerufen.

```
#Funktion um Nasadaten anzufragen
def nasa_request():
    #Überprüfung aller Filterparamenter
    try:
        day = float(nasa_req_day_input.get())
    except:
        nasa_req_progress_funk[2].config(text= "Fehlerhafte Eingabe der
    Genauigkeit!")
        return
    try:
        vmag = float(nasa_req_vmag_input.get())
    except:
        nasa_req_progress_funk[2].config(text= "Fehlerhafte Eingabe der
    visuellen Magnitude!")
        return
    if(day <= 0 ):
        nasa_req_progress_funk[2].config(text= "Die Genauigkeit in Tagen
    muss größer Null sein!")
    elif(vmag <= 0):
        nasa_req_progress_funk[2].config(text= "Die viuelle Magnitude muss
    größer Null sein!")
```

```
18      else:
19          #Nasadaten auswählen
20          try:
21              datei_name = select_file()
22          except:
23              nasa_req_progress_funk[2].config(text= "Die Datei konnte nicht
    aufgerufen werden!")
24              return
25          #Nasadaten anfragen
26          try:
27              nasa.get_nasa_data(datei_name, day, vmag, nasa_req_progress_funk
    )
28          except:
29              nasa_req_progress_funk[2].config(text= "Die Datei entspricht
    nicht Elins Standard!")
30              return
```

Listing 4.21: Funktionen um NASA-Daten über die Oberfläche anzufragen

Zunächst müssen die eingegebenen Parameter überprüft werden, um fehlerhafte Eingaben abzufangen. Beide Parameter müssen eine Gleitkommazahl größer als Null sein. Ist dies der Fall, wird der Benutzer aufgefordert, die gewünschte Datei mit den Daten des Bildauswertungsprogramms auszuwählen. Diese wird zusammen mit den anderen Parametern an die Funktion aus dem Kapitel 3.3.2 übergeben. Zusätzlich wird die Funktion so erweitert, dass sie auch einen Fortschritt anzeigen kann. Die Implementierung erfolgt analog zu demm Funktionsausschnitt 4.10. Eine weitere Änderung ist, dass im Infotext die Anzahl der Anfragen und der entsprechende Link angezeigt werden.

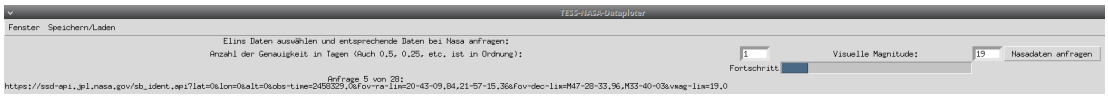

Abbildung 4.4: Ansicht während einer Anfrage

4.3 Arbeiten mit der Benutzeroberfläche

Die GUI bietet nun alle notwendigen Funktionen, um in den Daten des Bildauswertungsprogramms nach unbekannten Objekten zu suchen. Die Hauptaufgabe des Benutzers besteht darin einzelne Punkte zu entfernen, um Objekte zu identifizieren, die der NASA unbekannt sind.

4.3.1 Daten erstellen und NASA-Daten anfragen

Der erste Schritt bei der Arbeit mit der GUI ist die Erstellung der Daten mit Hilfe des Bildauswertungsprogramms. Dazu wird der gewünschten Bilddatensatz des TESS ausgewählt und auswertet, so dass eine Textdatei mit den auffälligen Datenpunkten erstellt wird. Diese Datei kann nun verwendet werden, um die NASA-Daten anzufragen, die im gleichen Bildausschnitt und zur gleichen Zeit wie die auffälligen Daten sichtbar waren.
Die Wahl der Parameter für die Anfrage der NASA-Daten ist vor allem für die Bearbeitungszeit entscheidend. Je genauer die zeitliche Auflösung gewählt wird, desto mehr Anfragen müssen gestellt werden, was wiederum mehr Zeit in Anspruch nimmt. Außerdem werden mehr Punkte der einzelnen Objekte dargestellt, so dass die Suche nach Zusammenhängen zwischen den NASA-Daten und den verdächtigen Daten länger dauert. Durch eine höhere Auflösung können die Korrelationen jedoch besser automatisch erkannt werden.
Die Wahl der scheinbaren Helligkeit bleibt dem Anwender überlassen. Durch die Wahl einer höheren scheinbaren Helligkeit, so dass auch dunklere Objekte erkannt werden können, kann ausgeschlossen werden, dass mögliche korrelierende Objekte am Ende fehlen. Allerdings wird dadurch der NASA-Datensatz größer, wodurch sich die Verarbeitungszeiten erhöhen können.

4.3.2 Daten einfügen

Die beiden Datensätze können nun in das Hauptfenster integriert werden, um unbekannte Objekte zu identifizieren. Zuerst müssen die Daten des Bildauswertungsprogramms eingefügt werden.
Diese müssen dann gefiltert werden. Dabei ist die Wahl der Parameter entscheidend. Je nach Datensatz muss geprüft werden, wie nah die zusammenhängenden Punkte beieinander liegen. In dem beispielhaft generierten Datensatz liegen die Tupel maximal 0,1 Grad auseinander, daher sollte der maximale Abstand ähnlich gewählt werden. Die minimale Anzahl der benötigten Punkte kann verändert werden, um die gewünschten zusammenhängenden Punkte herauszufiltern. Der Filtervorgang wird nun mit jeweils leicht angepassten Parametern wiederholt, bis die gewünschte Filterung erreicht ist.
Nun können die zuvor generierten NASA-Daten integriert werden. Dazu müssen wiederum Parameter gewählt werden, nach denen die NASA-Daten mit den gefilterten Daten verglichen werden. Es bietet sich an, den maximal zulässigen Abstand der Punkte zunächst ähnlich groß zu wählen wie bei der Filterung der Daten des Bildauswertungsprogramms. Die Genauigkeit der Tage sollte nicht kleiner gewählt werden als die Auflösung, mit der die Anfragen der NASA-Daten erstellt wurden. Ansonsten kann nicht gewährleistet werden, dass tatsächlich

zusammengehörige Punkte richtig zugeordnet werden, da der Zeitstempel zu weit entfernt ist. Die benötigten Punkte sollten verändert werden. Grundsätzlich gilt: Je großzügiger der maximal zulässige Abstand gewählt wird, desto mehr Punkte sollten übereinstimmen. Nun wird das Einfügen der NASA-Daten mehrmals wiederholt, wobei die Parameter jeweils angepasst werden. Wenn mehrere Objekte gefunden wurden, die übereinstimmen, werden diese beibehalten und es wird zu der Bearbeitungsphase übergegangen.

4.3.3 Daten bearbeiten

Nun müssen zunächst die nicht passenden Objekte aus den NASA-Daten entfernt werden. Dazu werden diese über die Liste ausgewählt und die nicht passenden gelöscht. Nachdem alle unpassenden Objekte entfernt wurden, gilt es, die bereits bekannten Objekte aus den Daten des Bildauswertungsprogramms zu entfernen. Dazu wird die Funktion um Punkte zu löschen aktiviert und alle Punkte, die mit einem der Objekte aus den NASA-Daten übereinstimmen, werden entfernt. Anschließend wird das Objekt aus der Liste gelöscht. Dieser Vorgang wird solange wiederholt, bis alle Objekte der NASA-Daten und die dazugehörigen Punkte gelöscht sind.
Nun kann entsprechend neu gefiltert werden, wenn dadurch weitere zusammenhängende Punkte sichtbar werden. Im Anschluss daran erfolgt erneut das Hinzufügen von NASA-Daten und das entsprechende Löschen der zugehörigen Punkte.
Werden bei diesem Vorgang zusammenhängende Punkte gefunden, die kein passendes zugehöriges Objekt in den NASA-Daten enthalten, besteht die Möglichkeit, ein der NASA unbekanntes Objekt entdeckt zu haben. In diesem Fall sollte noch einmal explizit für diese Koordinaten und Zeitstempel geprüft werden, ob doch ein passendes Objekt bekannt ist, um mögliche Fehler auszuschließen. Danach wäre es möglich, die weitere Flugbahn des Objektes zu bestimmen und in weiteren Bilddaten o.ä. genau danach zu suchen.

5 Fazit

5.1 Zusammenfassung

In dieser Bachelorarbeit wurde ein Programm zur automatischen Erkennung von Korrelationen zwischen den Daten eines Bildauswertungsprogramms für TESS-Bilder und den Daten der NASA-Datenbank vorgestellt. Das Programm beinhaltet verschiedene Funktionen, die für die Erkennung von unbekannten Objekten in den Bilddaten benötigt werden.
Besagte Funktionen reichen von der Filterung der Daten über die Anfrage korrelierender Daten über die NASA-API bis hin zur Aufbereitung der Daten, um Abweichungen aufzuzeigen, die durch die NASA-Daten nicht erklärt werden können.
Der Schwerpunkt liegt dabei auf einer Oberfläche, die dem Benutzer eine einfache und intuitive Möglichkeit bietet, nach unbekannten Objekten in den Datensätzen des Bildauswertungsprogramms zu suchen. Alle für die Umsetzung notwendigen Funktionen sowie die Werkzeuge innerhalb der Oberfläche wurden vorgestellt. Eine generelle Vorgehensweise zum Auffinden unbekannter Objekte wurde ebenfalls aufgezeigt.

5.2 Zukünftige Arbeit

Dieses Projekt bietet noch viele Möglichkeiten zur Weiterentwicklung.
Zum einen kann die Oberfläche weiter ausgebaut werden, so dass die Daten noch flexibler und präziser bearbeitet werden können. Es können weitere Funktionen hinzugefügt werden, um z.B. einen einfacheren Zugriff auf die NASA-Daten in der Datei zu ermöglichen. Die automatische Erkennung könnte noch verbessert und gegebenenfalls eine automatische Löschung von Datenpunkten eingebaut werden, wenn sich aus den NASA-Daten ein korrelierendes Objekt ergibt.
Es wäre auch denkbar, die Bildverarbeitung der Satellitenbilder von TESS ebenfalls in die Oberfläche zu integrieren. Dies würde die Suche nach unbekannten Objekten noch weiter vereinfachen, da alle notwendigen Werkzeuge in einer GUI zusammengefasst werden.
Auch die Anfragen an die NASA können optimiert werden. Um die benötigte Zeit deutlich zu

reduzieren, würde sich eine parallele Verarbeitung anbieten. So könnten die einzelnen Anfragen, die jeweils ca. 20-30 Sekunden benötigen, parallel ausgeführt werden. Es wäre jedoch zu prüfen, wie viele Anfragen gleichzeitig an die API der NASA gestellt werden können, ohne dass es zu Problemen kommt.
Die angeforderten NASA-Daten können noch genauer werden. Dazu könnten in einer weiteren Arbeit die Positionsdaten des TESS für einen bestimmten Zeitpunkt ermittelt werden. Mit einem solchen Verfahren könnten die Positionsdaten bei der Anfrage der NASA-Daten exakt übergeben werden, wodurch die Abweichungen zwischen den NASA-Daten und den Daten aus dem Bildauswertungsprogramm verringert werden könnten.
Zukünftige Arbeiten könnten auch Bilddaten eines anderen Satelliten oder Teleskops auswerten. Mit angepassten Standortdaten könnte das Programm auch auf andere Daten angewendet werden und somit universeller eingesetzt werden.

Bei Bedarf kann der Anhang (Quellcode, Beispieldateien) bei dem Erstprüfer eingesehen werden.

Literatur

[1] E. Thomfohrde-Dammann, *An Algorithm for the Discovery of Variable Stars and Transients in the Transiting Exoplanet Survey Satellite data archive*, Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg, 2023.

[2] C. Schäfer, *Schnellstart Python, Ein Einstieg ins Programmieren für MINT-Studierende*. Springer Spektrum, 2019.

[3] A. Hanslmeier, *Einführung in Astronomie und Astrophysik*. Springer Spektrum, 2020, Bd. 4.

[4] R. E. Team, „How to Fix RecursionError in Python“, März 2023, Last accessed 04.07.2023. Adresse: `https://rollbar.com/blog/python-recursionerror/#`.

[5] „Small-Body Identification API“, Version 1.1, Dez. 2021, Last accessed 04.07.2023. Adresse: `https://ssd-api.jpl.nasa.gov/doc/sb_ident.html`.

[6] G. Schilling, „Transiting Exoplanet Survey Satellite: the mission to find new worlds“, Apr. 2018, Last accessed 06.07.2023. Adresse: `https://www.skyatnightmagazine.com/space-missions/tess-to-impress`.

[7] „Small-Body Identification Tool“, Last accessed 04.07.2023. Adresse: `https://ssd.jpl.nasa.gov/tools/sb_ident.html#/`.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 09. Juli 2024 Ole Schweim