UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

Analysis of dust impacts observed with the Radio Plasma Wave instrument onboard ESA's Solar Orbiter

Alen Ferkic

FYS-3931 Master's thesis in Space Physics - December 2023

Abstract

Dust particles are one of the major constituents of the interplanetary medium in our solar system. They are presumably formed by fragmentation processes of meteoroids and for a certain range of sizes and parameters the dust particles move radially outwards from the Sun. The Radio plasma Wave (RPW) instrument on Solar Orbiter can measure dust particles that impact the spacecraft. By measuring the impacting dust particles we hope to learn about the size and mass distribution of the dust particles formed by the fragmentation processes.

For this thesis we investigate the amplitudes of observed dust impact signals from a set of convolutional neural network processed RPW data. The observations extend from June 2020 to June 2023 and during this period, Solar Orbiter reaches as close as 0.29 AU from the Sun. The model assumption is that the mass and the impact velocity of the dust particle is correlated to the measured voltage amplitude. We searched for systematic variations for recorded dust signals along the orbit of Solar Orbiter and the measured voltage amplitude were divided into three categories. The categories was compared for inbound and outbound trajectories as well as perihelion and aphelion paths. Assuming we have a constant impact velocity the slope of the mass distribution was derived. In addition, under the assumption of a constant dust velocity we infer the ratio of small particles along the orbit.

Many of the results indicate a mass distribution that increases with the distance from the Sun. Further the results showed that the dust impact flux is higher on an inbound trajectory compared to an outbound trajectory. A possible explanation for this is the influence of the relative velocity in the impact velocity and potential changes of the mass distribution.

Acknowledgements

I am sincerely thankful to my supervisor Professor Ingrid Mann for the guidance you provided to me throughout this master's thesis. You encouraged me and kept me on track especially during challenging moments. I would also like to express my gratitude towards Samuel Kočiščák for our helpful discussions and providing help with the data and questions I had about Python.

I would also like to show my appreciation to my co-students for motivating and inspiring me even through the darkest winters in Tromsø. You made the past five years enjoyable and it would not have been the same without you. A big thanks to Amalie Gjelsvik for also proof reading my thesis.

Lastly I would like to thank my family and friends at home for showing me the support and believing in me. Your support has been crucial throughout this academic journey and I am excited for the future that lies ahead.

Contents

List of Figures

List of Tables

1

Introduction

Dust particles play a role in our solar system in forming the zodiacal light and giving a brightness to the night sky. Most of the dust particles originate from comets and asteroids. The majority forms in the fragmentation process of dust - dust collisions and frequently occur in the inner solar system (Mann et al., 2019). The size and the amount of dust that is formed in the fragmentation process by collisions influence the mass distribution of the dust particles that is observed.

Dust particles can be detected by dedicated dust detectors such as the cosmic dust analyzer onboard the Cassini spacecraft (Srama et al., 2004). It is also possible to detect dust impacts with the use of radio instruments during space missions (Meyer-Vernet, 2001). This is possible because a large fraction of the material ionizes when the dust particle hits the spacecraft. The ionized material influences the measurement of the radio instrument. The Radio Plasma Wave (RPW) onboard the Solar Orbiter is one of those instruments. Solar Orbiter is one of the few space missions that explore the inner solar system inside 1 AU (Müller et al., 2020).

In this thesis we use data from the RPW to measure dust impacts that collides with the spacecraft using the process of impact ionization. When a dust particle collides with Solar Orbiter it generates a voltage pulse that the RPW records. Theory suggests that the amplitude of the signal generated varies with the impact speed and the mass of the impacting dust particle. The objective of this thesis is to investigate the recorded dust impact signals along the orbit of Solar

Orbiter so far and search for possible systematic variations. The aim is to infer information on the mass and impact velocity of the impacting dust particles from the measurements.

This thesis starts with an overview of Solar Orbiter in Chapter 2 and the background theory needed for the data product and measurement of the dust impacts. In Chapter 3, the theoretical framework for dust particles is provided. Chapter 4 gives a description of the method used in this thesis, including orbital parameters, voltage amplitude analysis and the calculation of the mass distribution slope. The results of the analysis and calculations are presented in Chapter 5 along with a discussion and interpretation of these results. Finally in Chapter 6, a summary and the conclusion of the results is presented.

2

Solar Orbiter

This chapter gives a overview of the objectives of Solar Orbiter. It also provides the background needed to understand Solar Orbiter's dust measurement technique. In addition we discuss the data product and selection. In section 2.1 we describe the launch and the mission. This is followed by a description of the instrument and the physics of dust measurement in section 2.2. Section 2.3 includes a description of the dust detection algorithm. Finally in chapter 2.4 we describe the data product and selection.

2.1 Launch and mission

ESA's Solar Orbiter was launched 10th of February 2020 using the Atlas V launch vehicle from Cape Canaveral Florida. The spacecraft will study the Sun with a minimum distance of $0.28AU$ and in an orbit with inclination up to $33°$ from the ecliptic plane (Müller et al., 2020). Using several gravity assist flybys from Venus, the spacecraft will adjust it's trajectory and increase the inclination. The first flyby in December 2020 corrected the orbit around the Sun, while subsequent flybys will increase the inclination.

The primary scientific objectives of the Solar Orbiter can be summarized into four questions that describe the overall mission (Mueller et al., 2013):

1. How and where do the solar wind plasma and magnetic field originate

in the corona?

2. How do transients drive heliospheric variability?

3. How do solar eruptions produce energetic particle radiation that fills the heliosphere?

4. How does the solar dynamo work and drive connections between the Sun and heliosphere?

To answer these questions, the Solar Orbiter carries several different instruments and one of those can also be used for detection of dust particles.

2.2 The Radio Plasma Wave (RPW)

The Radio Plasma Wave (RPW) is an instrument that measures and analyzes electric field and magnetic field fluctuations As described in detail in Maksimovic et al. (2020) It consists of three electric antennas that can measure properties of the plasma environment and solar wind. Conducting dust measurements with the RPW is not the main point of the instrument, but it was kept in mind when designing it that it would be able to measure dust impacts. The RPW has a subsystem called Time Domain Sampler (TDS) which provides electric wave forms, which are of interest in this thesis. The TDS records snapshots of the voltage measured when the voltage exceeds a certain threshold (Soucek et al., 2021). A description of the TDS system and the physics behind dust impacts is explained later in section 2.2.2 and 2.3. The antennas are negatively biased with respect to the spacecraft to minimize the potential variations with respect to the plasma at low frequencies (Khotyaintsev et al., 2021). There are two sample rates that the waveforms are sampled with, the first one is 262.1 kHz and the second one is 524.2 kHz. The latter is used when the spacecraft is inward of 0.5 AU.

2.2.1 Spacecraft charging

Spacecrafts are exposed to several charging processes in space which affects the electric potential of the spacecraft. The expression for the currents that work onto the spacecraft is given by:

$$\frac{dq}{dt} = I_{ph} + I_{sw} + I_{sec} + \dots \tag{2.1}$$

Where I_{ph} is the current due to photo electron emission, I_{sw} is the current caused by solar wind particles collection and I_{sec} is the current due to secondary electron emission by electron impact. A further explanation of each term is given in (Zaslavsky, 2015).

The equilibrium potential, also known as the floating potential, can be found by solving the steady state solution of equation 2.1, i.e when $dq/dt = 0$ The solution for the floating potential of the spacecraft is found to be:

$$\phi = T_{ph} ln \left(\frac{J_{ph0} S_{lit}}{e n_e v_e S_{sc}} \right) \tag{2.2}$$

where T_{ph} is the photon electron temperature, J_{ph0} is the photoemission current density, S_{lit} is the illuminated surface of the object, e is the elementary charge constant, n_e is the electron density, v_e is the averaged electron velocity and the S_{sc} is the surface area of the spacecraft. We see that the potential depends on the local plasma parameter n_e which is why it is also called the floating potential. A full description of the derivation is beyond the scope of this thesis and is described in detail in (Zaslavsky, 2015).

2.2.2 Dust measurement

The RPW instrument can be used to measure dust impacts through a process called impact ionization. When a dust particle hits the spacecraft, the dust particle is destroyed and a portion of the material ionizes. The process generates a cloud of free electrons and ions that are attracted or repelled from the spacecraft depending on the electric potential of the spacecraft. The charge Q that is generated by the impact can be described by the equation below (Auer and Sitte, 1968):

$$Q = \xi m^{\kappa} v^{\gamma} \tag{2.3}$$

where Q is given in Coloumb, m is the mass of the impactor given in kg and v is the impact velocity given in km/s. The constant ξ is a proportionality constant and parameters κ and γ are from experimental data that depends on the impactor and target composition (Mann et al., 2019).

Electrons are much lighter than ions which makes them quick compared to the ions, assuming that they have similar temperature which was shown to be the case (Collette et al., 2016). The Solar Orbiter is positively charged and when the dust particles hit the spacecraft the electrons are quickly collected while the ions are repelled in a short amount of time (μs timescale) before the spacecraft reaches its floating potential again. We assume that the potential of an antenna in monopole mode ϕ_{ant} is roughly constant during the process. The term monopole will be explained in section 2.4. This enables us to link the charge Q produced by the impact with the peak of our voltage pulse through the following equation: (Zaslavsky et al., 2021)

$$Q(m, v) \simeq \frac{C_{sc} V_{peak}}{\Gamma} \tag{2.4}$$

where C_{sc} is the capacitance of the spacecraft and Γ is an attenuation factor. expressed as:

$$\Gamma = C_{ant}/(C_{ant} + C_{stray})$$

Using equations 2.3 and 2.4 we can relate the measured voltage with the properties of the dust particle such as mass and velocity. The resulting equation is:

$$V_{peak} = \frac{\xi m^{\kappa} v^{\gamma} \Gamma}{C_{sc}} \tag{2.5}$$

2.2.3 Signal shape

The electric potential measured between an antenna and the spacecraft is recorded as a waveform, which can be studied. The combination of electrons being collected and ions being repelled after impact, gives rise to a pulse in the signal before the spacecraft relaxes to its equilibrium potential. The pulse is defined as the perturbation in the equilibrium potential given by $\delta\phi$ in equation 2.2.

The shape of the signals depend on the electric potential of the spacecraft. As mentioned, electrons are quicker than ions which means that they get attracted or repelled, respectively in the case of a positively or a negatively charged spacecraft. The antennas can also affect the signal; however since we measure in monopole mode and we assume that the antenna's potential are roughly constant, the antennas does not affect the shape in our case (Zaslavsky et al., 2021). a thing to note is that the location of the impact can affect which antenna reads the largest pulse - it is the antenna with the greatest amplitude that the impact location is closest to. Generally the antennas give the same shape when examined individually, but the amplitude difference is in the location of the impact. This allows us to study body and antenna effect separately.

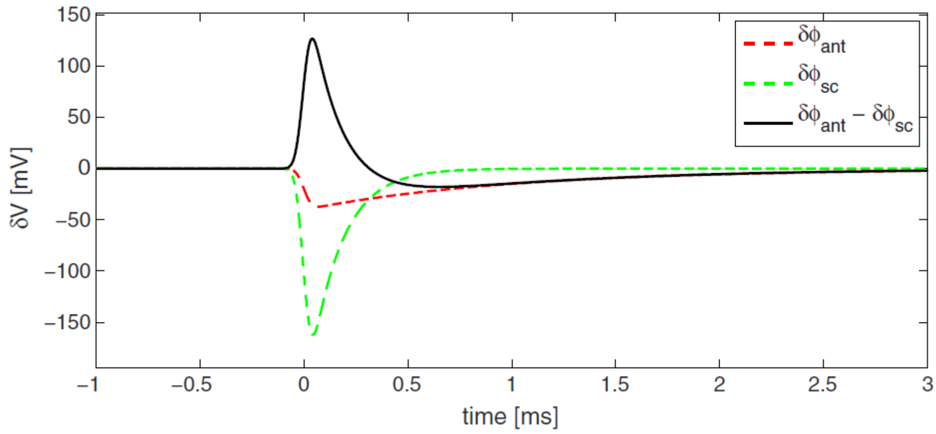


Figure 2.1: Simulation of an ideal dust impact signal. The red line corresponds to the antenna's potential, the green line corresponds to the spacecraft's potential and black line is the difference between the potentials. (Taken from Zaslavsky, 2015)

Figure 2.1 shows an ideal dust impact signal. The red line is the potential of the antenna ϕ_{ant}, the green line is the potential of the spacecraft ϕ_{sc} and the black line is the difference between ϕ_{ant} and ϕ_{sc}, which is the waveform we measure. We see that the potential difference gives rise to a voltage pulse. The green line decreases significantly compared to the red line and this is due to the spacecraft having a larger surface area which enables it to collect more of the free charges after the impact (Zaslavsky, 2015). Dust impacts can occur on the solar panel, but we do not use them as a collecting area due to them being electrically isolated on the front side. Impacts on the solar panels can produce charges that could be collected by conductive parts of the spacecraft but we neglect this effect (Zaslavsky et al., 2021)

The antenna has a saturation effect where it can cause inconsistent measurements of the amplitude. Large voltages amplitudes (> 200) mV may not be reliable as we are not sure if the measurement has been saturated. We do not know for certain what affects the saturation and at what limit an amplitude becomes saturated, but from inspection of the signals it has an variation but is never below 200 mV. Figure 2.2 shows a saturated dust impact, which is particularly evident in the third panel from the top where we can see that after the impact the signal decays exponentially before it breaks and then returns to the equilibrium.

Other effects that can alter the signal are high energy electrons that escape the potential of the spacecraft, which causes a negative spike in the signal due to

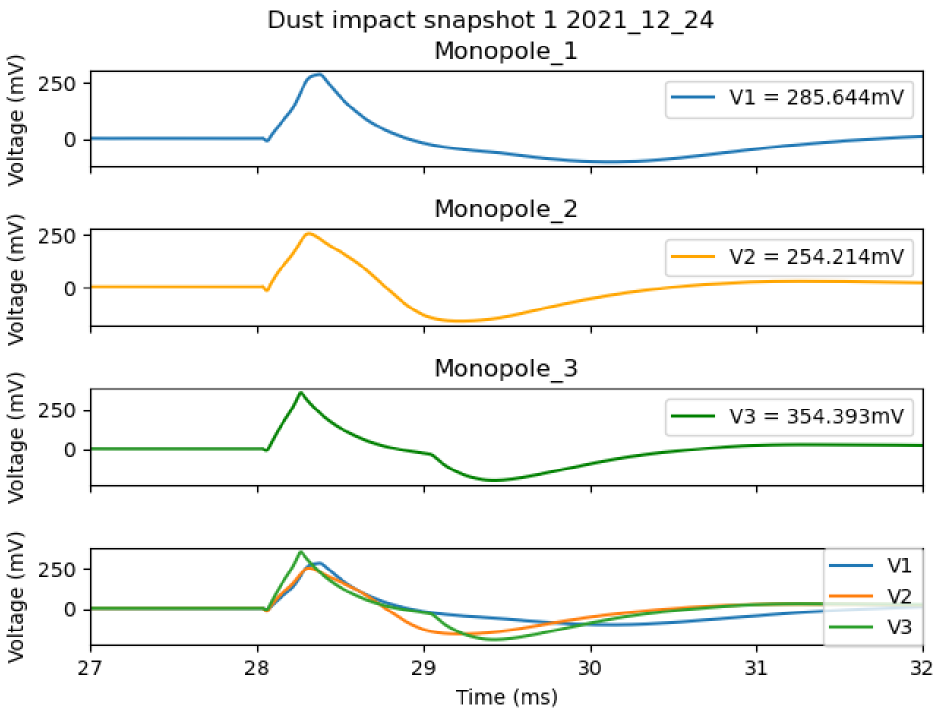


Figure 2.2: Shows a saturated dust impact on the three antennas. The uppermost panel shows antenna 1, followed by antenna 2 and 3. The last panel shows them together in one Figure.

the spacecraft becoming more positively charged. Another effect which we have to be aware of is ions that are repelled can influence the signal if they come too close to the antenna. One can think of it almost as ions are interfering with the antennas in a way that the antennas reads them twice. This leads to the antennas measuring positive charges which causes a secondary peak after the dust impact's primary peak. This secondary peak sometimes exhibits a greater amplitude than the primary peak. (Kočiščák et al., 2023b)

In this thesis I will focus on the amplitude of the dust impacts and not on the shape of the signals. A portion of the amplitude information to saturated dust impacts may be retrieved with signal processing techniques. I will not look into that as that is beyond the scope of this thesis, however this could be studied in future work.

2.3 Dust detection algorithm

The TDS has a detection algorithm on-board which allows the detection of dust impacts using an voltage amplitude threshold. The instrument takes a short snapshot of 1/16s of the waveform with a rate of 16384 every second. If the snapshot exceeds the minimum amplitude threshold it is recorded and processed by the software onboard the TDS. A detailed explanation of the detection algorithm can be found in the work of Soucek et al., 2021. The duty cycle of the TDS is between 1/32 and 1/16 which means that even under normal conditions it is only online for about 3% or % of a second.

The convolutional neural network (CNN) is a machine learning detection algorithm. Although The TDS has its own detection algorithm, The CNN detection method is used in this work. This is due to the CNN having a 96% ± 1% overall classification precision and 94% ± 2% dust detection accuracy while the on-board dust classifier has a 85% overall precision and 75% detection accuracy. A description of how the CNN detection method works can be found in the work of Kvammen et al., 2022. In some of the results we use a 5% error on the classification since we assume that it is related to the dust detection accuracy to CNN.

2.4 Data product and selection

The Data files used in this thesis can be downloaded from the link in the Appendix, In this thesis we have used data from June 2020 to June 2023. The data format is in ".cdf" and one ".cdf" file contains 24 hours of data that the

RPW instrument measures. From the website we use Data product level 2 (L2) and select tds_wf_e, this is TDS electric field waveform data which contains our voltage measurements. We are interested in the Triggered snapshot waveform (TSWF) which are the snapshots recorded when the TDS algorithm was triggered. The antennas have three configuration modes:

- SE1 mode: three monopoles, $V1$, $V2$ and $V3$
- XLD1 mode: two antennas measures dipole $(V1 - V3)$, $(V2 - V1)$ and one channel is antenna $V2$ against spacecraft $(V2 - V_{sc})$
- DIFF1 mode: three dipole, $(V1 - V3)$, $(V2 - V1)$ and $(V3 - V2)$

Figure 2.3 shows the configuration of the antennas. Monopole is the antenna's electric potential against the spacecraft's electric potential and dipole is the electric potential difference between two antennas. In SE1 mode the three channels are monopoles which is shown as the orange color in Figure 2.3. The blue color is XLD1 mode which has two channels in dipole and one channel in monopole. Meanwhile, DIFF1 mode has three channels in dipole colored in green. Maksimovic et al., 2020

XLD1 mode is used in this thesis as all of our data (both CNN and TDS) are in this mode. It is also the most practical since it can be adapted to monopole channels and the length between two antennas in dipole provides an advantage for plasma measurements. $DIFF1$ mode is not useful in our case since we want monopole measurement for simplicity. Not being able to adapt it to monopole makes the calculation of charge production more complicated since one would need to know the impact location with respect to the antennas (Zaslavsky et al., 2021).

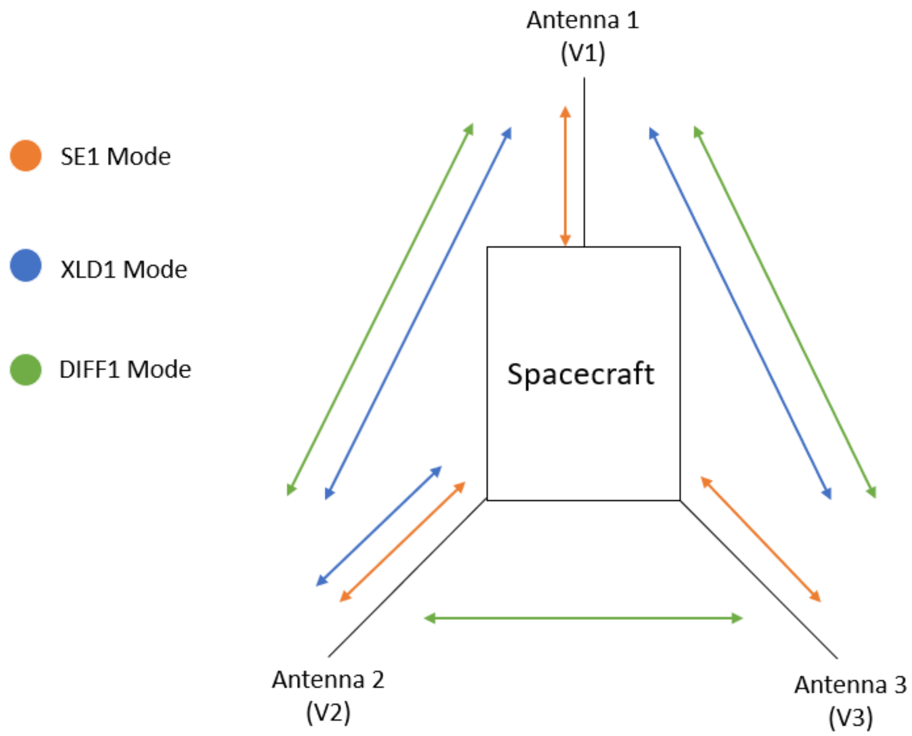


Figure 2.3: Sketch of the spacecraft and antenna configurations. SE1 Mode (orange) has only monopole channels shown with orange arrows. XLD1 Mode (blue) has 2 dipole and 1 monopole shown with blue arrows. DIFF1 Mode (green) has only dipole shown with green arrows.

In our case we use data that is recorded in $XLD1$ and the channels are shown below:

$$
\begin{aligned}
\mathrm{ch}_1 &= \phi_1 - \phi_3 \\
\mathrm{ch}_2 &= \phi_2 - \phi_1 \\
\mathrm{ch}_3 &= \phi_2 - \phi_{sc}
\end{aligned}
$$

where ϕ_1, ϕ_2, ϕ_3 are the antennas electric potential and ϕ_{sc} is the spacecraft potential. We can arrange the channels on this form ($V_i \equiv \phi_i - \phi_{sc}$) so that we measure in monopole mode:

$$
\begin{aligned}
V_1 &= \mathrm{ch}_3 - \mathrm{ch}_2 \\
V_2 &= \mathrm{ch}_3 \\
V_3 &= \mathrm{ch}_3 - \mathrm{ch}_2 - \mathrm{ch}_1
\end{aligned}
$$

V_1, V_2 and V_3 is now our new voltage in monopole. As mentioned the signal can be affected by a secondary peak which affects the amplitude. Upon observing the signals of dust impacts, it seems that it is only one of the antennas that shows the secondary peak effect most of the time, while the other antennas have more or less equal amplitude. There is no preferred antenna that shows the effect. In the data processing, the effect (of a secondary peak) is removed by only considering the signal from the antenna with the lowest peak amplitude, and disregarding the signals from the other two antennas. This method allows us to exclude the secondary peak while maintaining an accurate representation of the amplitude. The offset of each channel is removed by subtracting the average.

3

Background considerations on dust particles

The majority of the dust in the solar system originates from the fragmentation of celestial objects such as asteroids and comets. These large objects are known as the parent objects of the dust. The produced dust particles from the fragmentation process are initially in orbits with inclination similar of their parents body (Mann et al., 2004). Large dust particles on the scale of micrometer are in Keplerian orbits around the Sun with velocities and number density increased close to the Sun. (Mann et al., 2019).

The forces that act on a dust particle are the gravitational pull from the Sun, the Solar radiation pressure and the Lorentz force. If the charge to mass ratio of the dust particle is small, the Lorentz force can be neglected and in this case the Lorentz force is neglected. The dynamics of dust particles can be described by the Solar radiation pressure and the gravity from the Sun. These forces are radial forces working in opposite directions which makes their ratio a useful way to describe dust trajectories. This ratio is known as the β-value of the dust particles and is expressed in the equation below.

$$\beta = \frac{F_{rad}}{F_g} \tag{3.1}$$

The β-value together with the initial conditions describes the dust trajectories. The large dust particles are dominated by the gravity force and have a $\beta -$ value between $0 < \beta < 0.5$. Small dust particles are often in a unbound hyperbolic orbit and have a $\beta -$ value of $\beta \geq 0.5$, these particles are known as β-meteoroids. β-meteoroids are pushed radially outwards from the Sun due to the Solar radiation pressure. It is assumed that Solar Orbiter collides with the β-meteoroids. The β-value depends on the size and composition of the dust particle and are discussed below.

Since the dust particles have mass they feel the gravitational force from the Sun. This force can be described as:

$$F_g = -\frac{GM_{\odot}m_d}{|r|^3}r$$

Where G is the gravitational constant, $M_{\odot}$ is the Solar mass and m_d is the mass of the dust particle. r is position vector of the dust particle relative to the Sun.

The Sun emits photons in all directions, propagating outward at the speed of light. The photons carries momentum which enables them to scatter of objects transferring the momentum. When a photon collides with a dust grain it scatters off the dust particle exerting a force onto it. This is the Solar radiation pressure expressed as:

$$F_{rad} = \frac{A_d Q_{pr} S_0}{|r|^3}r$$

Where A_d is the cross section of the dust particle, Q_{pr} is the efficiency factor of the radiation pressure weighted by the solar spectrum and S_0 is the solar flux weighted for the distance to the Sun. A detailed explanation of this parameter is given in Sterken et al., 2012.

Figure 3.1 shows us the β-value as a function of the mass in grams for two dust species. In the original work from Wilck and Mann (1996), there are two more curves that are similar to asteroidal dust, which are old cometary and interstellar dust. It is safe to assume that our dust grains follow the β-value for asteroidal dust and not young cometary dust. This is because the analysis previously done shows trajectories that can be explained with β-values around 0.5. (Kočiščák et al., 2023b).

If we consider a dust particle, we notice that F_g is dependent on the mass and F_{rad} is dependent on the cross section. Both are related to the size as mass can be described as $m = \rho V$. Assuming we have a spherical dust particle, We can see that F_g scales with r^3_{dust} and F_{rad} scales with r^2_{dust}, which causes the gravity force to dominate. This is the explanation of the right hand side of the peak in Figure 3.1.

On the other hand we see that for low mass the β-value is small. If the dust particle is significantly smaller than the wavelength of the photon, the photon will pass through it and not be able to scatter off the dust particle. This results in not pushing it outwards as there is no momentum transfer. The dust particles that has a size much smaller than the typical wavelength of the solar spectra does not scatter the sunlight photons effectively.

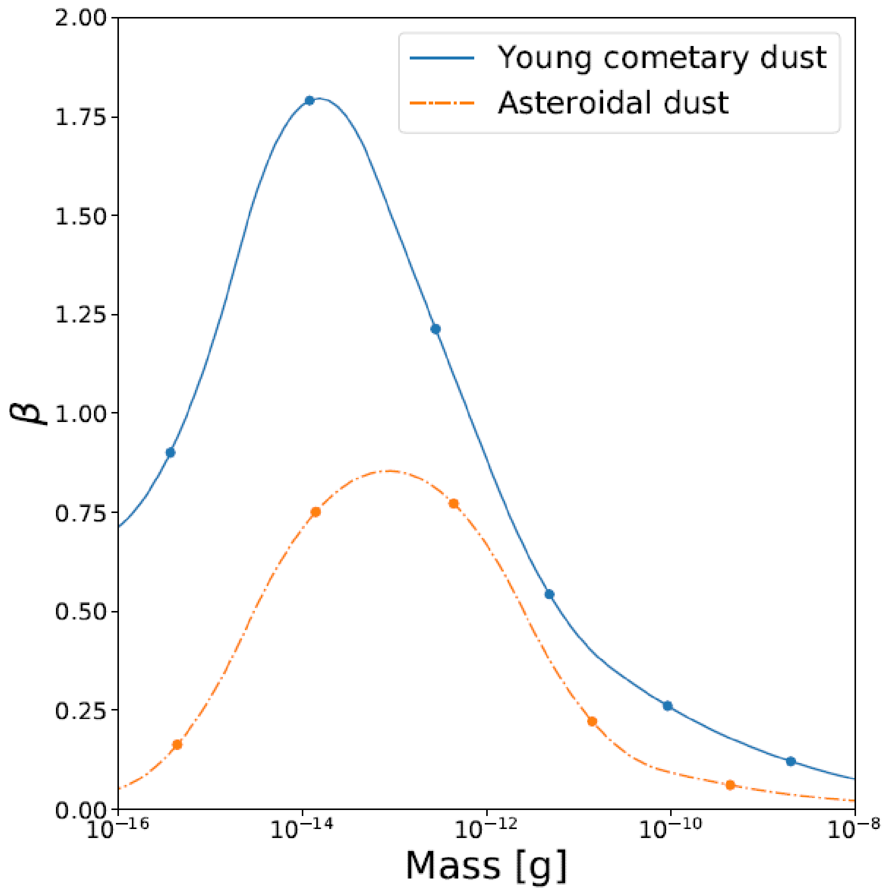


Figure 3.1: Shows the β-value as a function of the mass for young cometary dust and asteroidal dust. This Figure is taken from Henriksen, 2020 in which he adapted it from Wilck and Mann (1996).

4
Methods

In this chapter we discuss the methods used to derive the results. A brief look of the orbital parameters is introduced in section 4.1, following a method description of the voltage amplitude measurements in section 4.2. In section 4.3 a derivation of the slope to the mass distribution is included as well as the calculation of a charge production threshold in section 4.4.

4.1 Orbital Parameters

Orbital parameters were calculated by generating an ephemeris for Solar Orbiter. It can be generated with the link in the Appendix. Figure 4.1 shows the radius of Solar Orbiter and the radial velocity with respect to the Sun. It has data from 15^{th} of June 2020 to 14^{th} June 2023 which is the same amount of data in days as the measurements.

The closest point to the Sun is 0.29 AU and the furthest is 1.01 AU. The radial velocity ranges from -23.6 to $+23.6$ km/s. On the 27^{th} of November 2021 Solar Orbiter did an Earth flyby to adjust the trajectory of the spacecraft, which can be seen in the radial velocity panel where the curve shows a sudden change around 8^{th} of December.

The coordinate system used is Heliocentric Aries Ecliptic (HAE) which has the Z-axis normal to the ecliptic, X-axis points towards the first point of Aries on the Vernal Equinox and Y-axis completes the system forming a right-handed coordinate system. A description of the coordinate system can be found in *Henriksen* (2022).

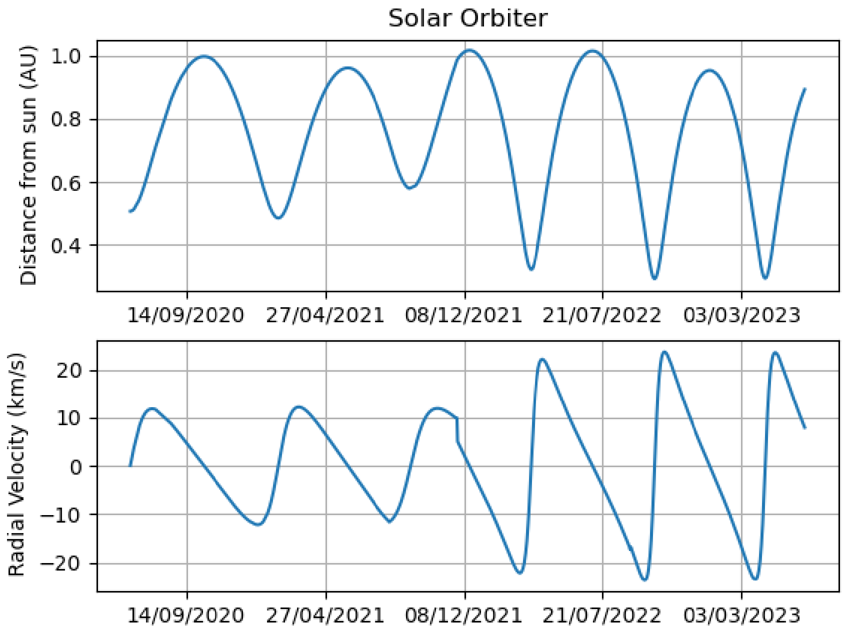


Figure 4.1: The heliocentric distance and radial velocity of Solar Orbiter from 15th of June 2020 to 14th June 2023.

4.2 Voltage amplitude analysis

We are interested in the voltage peak amplitude as they are related to the mass and velocity of the impacting dust particles. Voltage measurements can have a low amplitude, resulting in an uncertainty in whether the signal is a dust impact, or if it is a potential perturbation that triggers the TDS algorithm. On the other hand we can have dust impacts with saturated voltage amplitudes which does not reveal the true amplitude. Due to this effects we categorize the voltage amplitude into three groups. *weak* corresponds to dust impacts with low voltage amplitudes, *mid* corresponds to voltage amplitudes we are certain of and *strong* corresponds to dust impacts with voltage amplitudes that may be saturated. Figure 4.2 shows the normalized density of the peak voltages of dust impacts measured from April - November 2020. This Figure is adapted from a publication by Zaslavsky et al. (2021). We see in the Figure that there are dust impacts that deviates from the straight line both at the lower amplitudes and higher amplitudes.

Due to these deviating points we want to categorize the amplitudes in a way that represents the groups accurately. In the section 5.1 we describe the chosen values and the reason behind it.

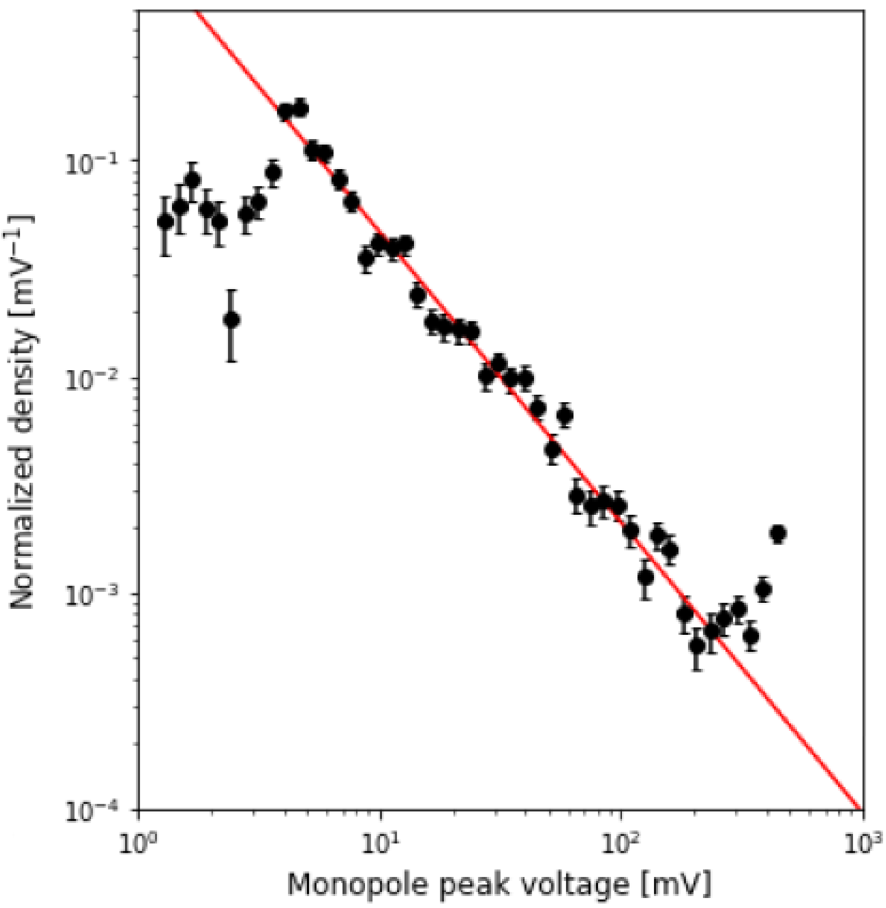


Figure 4.2: Normalized density distribution of peak voltage with logarithmic scales from April - November 2020. Red line shows a power-law least-square fit with a slope of -1.37 ± 0.07. Errorbars are computed from $\sqrt{N_{bin}}/(\Delta V_{bin} N_{tot})$ where N_{bin} is the number of counts in the bin, ΔV_{bin} the bin width in mV, and N_{tot} the total number of events from which the histogram is computed. This Figure is taken from the work of Zaslavsky et al. (2021).

4.3 Derivation of the slope of the mass distribution

The voltage peak distribution follows a power law behaviour as can be seen in Figure 4.2. The slope that they found as a result of using a power-law least-square fit has an value of $a \approx 1.34$ (Zaslavsky et al., 2021). In this section we show a new method on how to derive the slope value a by looking at the charge production between two interval ranges. Based on assumptions we can infer the mass distribution of the dust impacts. The results are presented in section 5.4.

The intervals ranges are represented below:

$$Q_{lo} < Q < Q_{hi}$$
$$Q_{hi} < Q$$

We assume we have a probability density function (PDF) of the masses:

$$f_m(m) = c \cdot m^a \tag{4.1}$$

Equation 4.1 integrates to

$$\int_{lo}^{hi} f_m(m)\, dm = c \left[\frac{m^{a+1}}{a+1} \right]_{lo}^{hi} = c \frac{hi^{a+1} - lo^{a+1}}{a+1}$$

Where lo and hi are lower and upper boundaries respectively, later in the thesis we explain our selection for lo and hi.

We assume that the impact velocity is constant within the time periods of the intervals in question, which enables us to have a charge production that depends only on the mass $Q(m)$. The proportionality constant is included in v, we then have:

$$Q = mv$$

$$\frac{Q}{v} = m \implies Q \propto m$$

This implies that:

$$f_m(m) = c \cdot m^a \implies f_Q(Q) = c' \cdot Q^a$$

The slope of $f_Q(Q)$ is the same as the slope of $f_m(m)$. Using measured values of Q we can infer the slope of the mass distribution, assuming that the mass distribution does not change over the time period of the measurements. Considering equation 2.3, we see that a higher mass m implies a higher charge Q given that the velocity v is constant. The masses for our boundaries lo and hi are:

$$m_{lo} = \frac{Q_{lo}}{v}$$

$$m_{hi} = \frac{Q_{hi}}{v}$$

where Q_{lo} and Q_{hi} is the charge production related to the mass m_{lo} and m_{hi}, respectively.

From that we predict the number of dust grains between the two masses, m_{lo} and m_{hi} using the PDF:

$$N_{Q\in(Q_{lo}:Q_{hi})} = \int_{m_{hi}}^{m_{lo}} f_m(m)\, dm = \tag{4.2}$$

$$N_{Q\in(Q_{lo}:Q_{hi})} = \int_{\frac{Q_{lo}}{v}}^{\frac{Q_{hi}}{v}} f_m(m)\, dm \tag{4.3}$$

Then we integrate equation 4.3:

$$N_{Q\in(Q_{lo}:Q_{hi})} = c\frac{\left(\frac{Q_{hi}}{v}\right)^{a+1} - \left(\frac{Q_{lo}}{v}\right)^{a+1}}{a+1} \tag{4.4}$$

$$= c\frac{\frac{Q_{hi}^{a+1}}{v'} - \frac{Q_{lo}^{a+1}}{v}}{a+1} \tag{4.5}$$

$$= \left(\frac{c}{v^{(a+1)}(a+1)}\right)(Q_{hi}^{a+1} - Q_{lo}^{a+1}) \tag{4.6}$$

For the other interval we integrate:

$$N_{Q>Q_{hi}} = \int_{\frac{Q_{hi}}{v'}}^{\infty} f_m(m)\,dm$$

We note that it is strictly that $a < -1$ which enables the infinite term to converge to 0. Integrating we get:

$$N_{Q>Q_{hi}} = \int_{\frac{Q_{hi}}{v'}}^{\infty} f_m(m)\,dm = \left(\frac{c}{v'^{(}a+1)(a+1)}\right)(-Q_{hi}^{a+1}) \tag{4.7}$$

We relate the expressions 4.6 and 4.7:

$$\frac{N_{Q>Q_{hi}}}{N_{Q\in(Q_{lo};Q_{hi})}} = R = \frac{-Q_{hi}^{a+1}}{Q_{hi}^{a+1} - Q_{lo}^{a+1}} \tag{4.8}$$

Where R is the ratio between the counts N of the intervals

We invert equation 4.8 to get a:

$$a = \frac{-\ln(Q_{hi}) + \ln(Q_{lo}) + \ln\left(\frac{R}{R+1}\right)}{\ln(Q_{hi} - \ln(Q_{lo})} \tag{4.9}$$

Equation 4.9 was found using Wolfram Mathematica which is a computational intelligence software. The equation implies that R has to be positive number. The values for Q_{hi} and Q_{low} are discussed later in the thesis.

4.4 Simplified mass distribution

In this section we use a constant mass threshold to derive a charge production threshold. This enables us to investigate the mass of the impacting particles. From the voltage measurements we calculate the charge production of the dust impacts and see if it exceeds the charge threshold. This will tell us if the dust impact had a mass larger or smaller than the fixed mass. The charge production threshold is calculated using the relation between equation 2.3 and equation 2.4. The peak voltage amplitude is related to the charge production with equation 2.5. We assume that the impact velocity is equal to the velocity of the dust particle and the radial velocity of the spacecraft given in the equation below:

$$v_{imp} = v_{dust} + v_{rad}^{solO} \tag{4.10}$$

We assume that $v_{dust} = 50$km/s since Zaslavsky et al. (2021) derived that as the average dust velocity in his work. It is also assumed that it remains constant during the orbits, seeing as in Figure 1 in the work of Kočiščák et al. (2023a) it remains roughly constant for different β-values at different intervals of distances.

Since the radial velocity of Solar Orbiter varies daily, we calculate a daily charge production threshold as a function of the radial velocity of the spacecraft. In this derivation, we consider a high charge yield, leading to the parameters in equation 2.3 being assigned values of $\xi = 0.7$, $\kappa = 1$, and $\gamma = 3.5$ (McBride and McDonnell, 1999).

The fixed mass used in this work is $m = 1.0 \times 10^{-17}$ kg, the reason being that panel 2 of Figure 7 in the work of Zaslavsky et al. (2021), shows that the mass is on the order of 1×10^{-17} kg. The dust impacts that has a charge production that exceeds the threshold are then compared against the dust impacts that do not exceed the threshold. This will tell us about the mass of the dust particles.

5

Results and discussion

This Chapter provides the result and the discussion, we first take a look at the voltage distribution and determine our categories in section 5.1. A comparison of the inbound and outbound trajectory is conducted in section 5.2 to search for variations along the orbit. This is followed by mass threshold comparison along the orbit with the aim of seeing a trend for the mass distribution in section 5.3. Section 5.4 we calculate the slope value of the voltage amplitude distribution which is related to the mass distribution of the dust particles. Finally in section 5.5 we take a look at the impact velocity at two different trajectories that corresponds to the perihelion and aphelion to search for variation due to impact velocity.

5.1 Choice of the voltage amplitude categories

Figure 5.1 shows the normalized density of the peak voltages of all the measurements. We see that the majority of the measurements follow a linear fit on logarithmic scales, especially during the *mid* section. A portion of the *weak* voltages deviates from the line at around 10^0mV, there is also two peaks in the *weak* section but as mentioned the *weak* section can contain noise signals that has triggered the TDS algorithm. On the *strong* we see that there are measurements that do not follow the line and are scattered around. The measurements are in agreement with the results reported by Zaslavsky et al. (2021), we see a similar trend of amplitudes following a linear fit as seen in Figure 4.2.

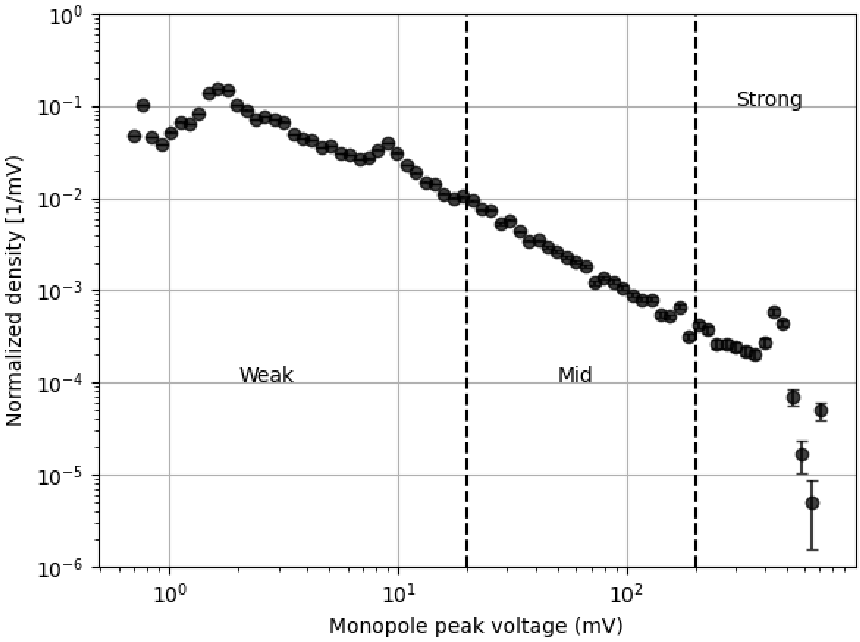


Figure 5.1: Normalized density of the peak voltages with logarithmic scales. The dashed lines are located at $20mV$ and $200mV$.

Based on Figure 5.1 I choose my *weak*, *mid* and *strong* values. We see that there are inconsistencies in following a line at the start of the *weak* section. There is also the two peaks which are hard to explain but as mentioned this section can be dust impacts or noise that has triggered the TDS algorithm. For these reasons a value of amplitudes greater than 20 mV has been chosen for the *weak* category.

The *mid* section follows a consistent line from 20 mV until the 200 mV mark before the amplitudes start to scatter. Therefore the *mid* category contains amplitude values between 20 and 200 mV. The remaining *strong* group is then amplitude values greater than 200 mV, we also see that a portion of these points are scattered which can be a result of saturated amplitudes. The categories are represented in table 5.1 as well as the counts of them throughout all of the measurements.

Name	Amplitude	Counts
Weak	20 mV > V	5808
Mid	20 mV < V < 200 mV	2737
Strong	200 mV < V	1076

Table 5.1: Voltage amplitude categorized into three categories. *Weak* are amplitudes less than 20 mV, *mid* are amplitudes between 20 mV and 200 mV and *strong* are amplitudes greater than 200 mV.

5.2 Comparison of inbound and outbound trajectories

Figure 5.2 shows the ratio between the groups *weak*, *mid* and *strong* for inbound and outbound trajectories. The inbound is showed in dashed lines and outbound are fully drawn lined. Error bars are calculated with a $\sqrt{N}$, where N is the counts for one group in each distance point for inbound and outbound respectively. In addition there is a 5% classification error included due to CNN's classification precision. The distance 0.3 AU is not included because the ratios between inbound and outbound trajectory at this point is significantly different.

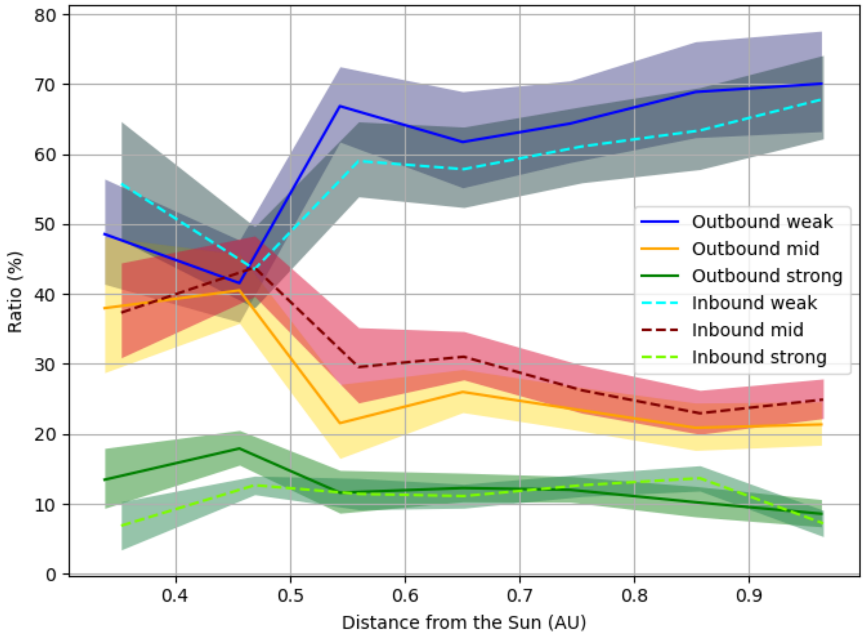


Figure 5.2: Amplitude ratio of the three categories, shows the inbound and outbound orbits. Fully drawn lines are outbound and dashed lines are inbound. Errors are filled in around the lines. All of the inbound sums up to 100 % and all of the outbound sums up to 100 %.

We see that the ratios between the groups are somewhat similar for inbound and outbound at the aphelion ≈ 1.0 AU. As mentioned the distance point 0.3 AU is not included due to the ratios between the groups being substantially different for both trajectories. At the perihelion and aphelion this should not be the case since the spacecraft is more or less at same point regardless of an inbound or outbound trajectory. The spacecraft also spends less days at the perihelion since the velocity is higher closer to the Sun. This means that it does not get as many dust impacts at the perihelion compared to the aphelion, which could explain the difference. We clearly see that there is an increase of dust impacts with a *mid* amplitude class from 0.5 AU and beyond for inbound compared to the *mid* class at outbound. Error bars do overlap when looking at 0.65 AU.

We know that the relative velocity between Solar Orbiter and the dust grains are different for inbound and outbound trajectory. My interpretation is that on the inbound orbit the relative velocity is greater compared to the outbound resulting in a higher impact velocity. This generates a voltage pulse with an amplitude high enough that classifies a *weak* dust impact on outbound orbit to *mid* on inbound orbit. One would also expect the *strong* amplitude impacts to increase in correlation with the *mid* amplitude impacts. However this seems not to be the case.

A further look, shows that *weak* amplitudes decreases at 0.45 AU and beyond on the inbound compared to the outbound. In table 5.2 and 5.3, we see that the total numbers of impacts are greater for inbound compared to the outbound for every distance interval which is expected since the dust moves radially outwards from the Sun. This might indicate that, an increase in relative velocity on inbound orbit produces higher charge that results in more dust impacts.

We do see that the outbound *strong* impacts are greater than the inbound for 0.35 – 0.5 AU, looking at the numbers in 5.2 and 5.3 we see that the counts of *strong* in this interval are greater for the outbound path but it is not a recurring pattern for the other intervals. A possible explanation is that for the outbound *strong* the mass had a greater influence on the charge production compared to the impact velocity for these dust impacts. Other than that it is hard to draw a conclusion for the increase in *strong* at this interval.

In general, we do see an higher flux of impacting dust particles on the inbound trajectory compared to the outbound trajectory, this is sensible as the impact velocity is greater on an inbound path compared to an outbound path which can result in more dust impacts being detected. We see that it is the *weak* and *mid* categories that shows most variation between the two trajectories while the *strong* shows small variation.

Table 5.2: Shows the inbound numbers for *weak*, *mid*, *strong* and the total at different distances.

Distance (AU)	Weak	Mid	Strong	Total
0.3 – 0.4	348	233	43	624
0.4 – 0.5	189	190	55	434
0.5 – 0.6	449	225	87	761
0.6 – 0.7	520	279	100	899
0.7 – 0.8	482	207	100	789
0.8 – 0.9	519	188	112	819
0.9 – 1.01	760	279	81	1120

Table 5.3: Shows the outbound numbers for *weak*, *mid*, *strong* and the total at different distances.

Distance (AU)	Weak	Mid	Strong	Total
0.3 – 0.4	202	158	56	416
0.4 – 0.5	160	156	69	385
0.5 – 0.6	363	117	63	543
0.6 – 0.7	508	214	101	823
0.7 – 0.8	316	116	59	491
0.8 – 0.9	343	104	51	498
0.9 – 1.01	538	164	66	768

Distance (AU)	Number of impacts
0.2 – 0.3	251
0.3 – 0.4	1040
0.4 – 0.5	819
0.5 – 0.6	1304
0.6 – 0.7	1722
0.7 – 0.8	1317
0.9 – 1.01	1888

Table 5.4: Shows the number of impacts at different distance intervals.

5.3 Mass ratio of a constant mass

Figure 5.3 shows the ratio of dust impacts that has a mass larger or smaller than 1.0×10^{-17} kg at 0.1 AU distance intervals. The distance intervals are the sum of dust impacts that are between the current point and the preceding interval step. For example the interval 0.3 shows the dust impacts within the range $0.2 - 0.3$, while the 0.4 point includes impacts between 0.3 and 0.4 and so on. Because of this, an offset of 0.05 AU has been added to represent the average between two intervals. The ratio is then calculated from the total between the intervals. The error bars are produced assuming that events near the mass threshold have been miss categorized. Near the threshold corresponds to a lower limit of $0.75Q_{threshold}$ and an upper limit of $1.5Q_{threshold}$.

We see a clear trend that the ratio of masses smaller than 1.0×10^{-17} increases with decreasing distance to the Sun. A possible explanation is that we have smaller dust particles closer to the Sun. Since we do not know the mass distribution of the dust particles throughout the solar system nor the mass distribution of fragmentation of larger objects it is hard to conclude anything at this point. Nevertheless we know that the β-value from Figure 3.1 is low for either very small dust particles that does not get affected by solar radiation pressure or too large dust grains where the gravitational pull dominates. One assumption that is made is that the dust particles are of same composition since this can affect the β-value. Due to the fact that Figure 5.3 shows small masses closer to the Sun, this could indicate that we are on the left side of the β-value peak as shown in figure 3.1 where we have small particles.

Calculations of the charge production threshold was also attempted with a fixed mass of 1.0×10^{-16} kg however this became only one-sided and gave us nothing to compare with. This tells us that given the average $v_{dust} = 50km/s$, there are no dust impacts with masses larger than 1.0×10^{-16} kg. For better comparison one would do intervals of masses and see how they change along the distance, however this was not done due to the limited amount of time.

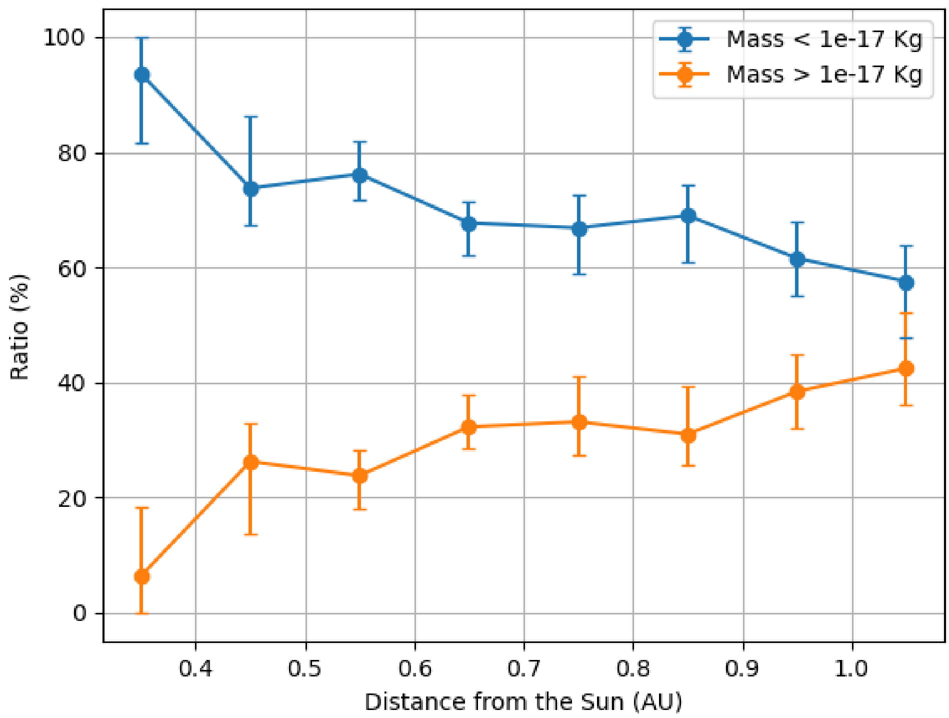


Figure 5.3: The mass ratio between two masses as a function of the distance. Blue line shows dust impacts with masses larger than 10^{-17} kg and Orange line shows masses smaller than 10^{-17} kg. Error bars are constructed with upper and lower bounds of charge production threshold each day. A offset of 0.05 AU is included as well.

5.4 Slope of the mass distribution

The intervals we consider in equation 4.9 are the intervals for the *mid* and *strong* categories. This means that we compare the number of *mid* versus the number of *strong*, so our Q values in equation 4.9 are Q_{lo} which corresponds to a voltage amplitude of 20 mV and Q_{hi} which corresponds to a voltage amplitude of 200 mV. Figure 5.4 shows the slope value a calculated with equation 4.9. Each point represents the sums of *mid* and *strong* individually in a weekly interval where the impact velocity is assumed constant within each week. The distance is the average of all the days within the week. The red line is constructed by a least square polynomial fit of the points and the mean value of all the points are $a \approx -1.56 \pm 0.18$. The error is the standard deviation of all the points.

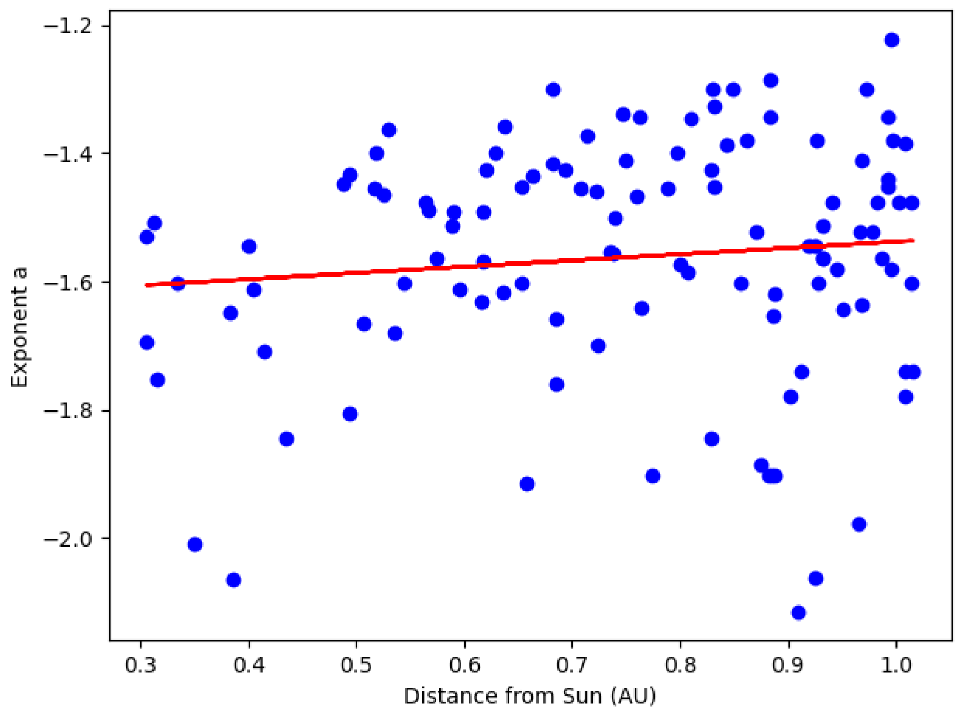


Figure 5.4: Shows the slope value a from weekly intervals for all the days recorded as a function of distance. The red line is a least-square fit of all the points.

In the work of Zaslavsky et al. (2021), the authors reported that their results of linear fitting showed a steeper curve at the perihelion compared to the aphelion, this can be seen in Figure 4.2. They also mention that it is hard to draw a conclusion of this result and it is sensible to wait for more data. By studying Figure 5.4 it seems that the slope trend continues with extended data. The slope is steeper closer to the Sun compared to far away. The slope value they found using a least square fit of the voltage distributions is $a = -1.34 \pm 0.07$. With the method used in this thesis, the value of a is calculated to be $a = -1.56 \pm 0.18$. This seems to be reasonable in agreement with the authors considering we have different amount of data. The data Zaslavsky et al. (2021) used is from 1^{st} of April to 30^{th}, during this period, Solar Orbiter have not done its first gravity assisted flyby. This means that the perihelion is at approximately 0.5 AU while the perihelion for this data is at approximately 0.29 AU.

If we consider Figure 5.4 we see that smaller slope value a results in a steeper curve along the distance. Since we are using the ratio of counts between the *mid* group and *strong* group, this indicates that there are fewer strong classified amplitudes compared to *mid* classified amplitudes. As mentioned before there is a possibility that there are smaller dust particles closer to the Sun compared to far away. One could argue that the slope supports this explanation since there are fewer *strong* amplitude dust impacts than *mid* amplitude dust impacts at this distance given that they have the same impact velocity.

Some of the differences between the slope values for (Zaslavsky et al., 2021) and this investigation, are the method, the data and the classification method. The authors did a linear fitting of the voltage distribution in Figure 4.2 where the voltage ranged from $4-206$ mV (Zaslavsky, personal communication 2023). The same assumption they used is also used in this thesis, that we assume the velocity is independent of the mass and that the mass is proportional to the charge production. However, we find the slope value by finding the ratio of the counts between the measured amplitude categories *mid* and *strong* that has a voltage amplitude range of $20-200$mV. As mentioned Zaslavsky et al. (2021) used data from 1^{st} of April to 30^{th} with the TDS algorithm while the data used in this work is from June 2020 to June 2023 with the CNN data.

As previously discussed, the trend of a steeper a closer to the Sun appears to be in agreement of what has formerly been shown. There are some factors that could distinguish the similarities between my report and previous work, nonetheless as the slope values a are derived using the same assumption of a constant velocity, it seems plausible.

5.5 Comparison of perihelion and aphelion impacts

We want to investigate how the voltage amplitude is influenced by the impact velocity v_{imp}. An attempt is to compare the categories for different impact velocities at trajectories for the perihelion and the aphelion to see if there is a noticeable difference for the categories. Figure 5.5 illustrates the trajectories we want to compare. The shaded area corresponds to the perihelion trajectory, assuming Solar Orbiter travels from minimum v_{rad} to maximum v_{rad} where v_{rad} is the radial velocity of the spacecraft. The aphelion is then the remaining trajectory of the orbit from maximum v_{rad} to minimum v_{rad}. We also assume that the dust particles traveling radially outwards from the Sun has a velocity of v_{dust} = 50 km/s. The impact velocity is then:

$$v_{imp} = v_{rad} + v_{dust} \tag{5.1}$$

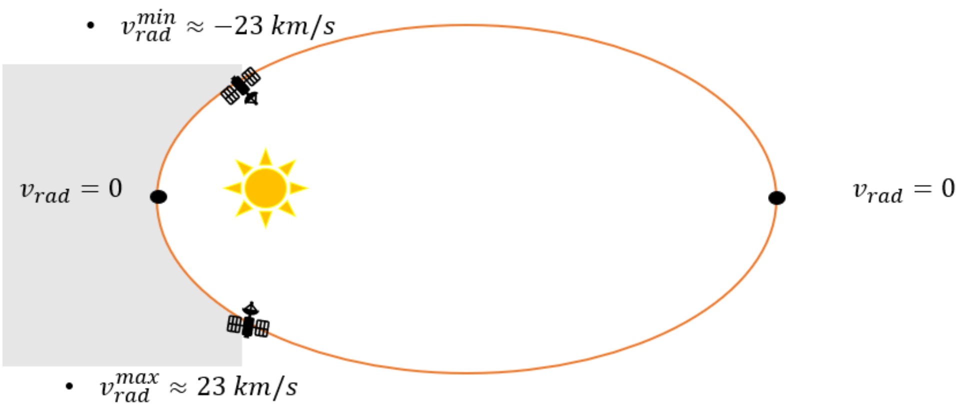


Figure 5.5: Shows the perihelion side and aphelion trajectory. Shaded area corresponds to the perihelion and the white area corresponds to the aphelion side.

Figure 5.6 shows the ratio of *weak*, *mid* and *strong* categories as a function of the impact velocity for the perihelion side (**a**) and aphelion side (**b**). Error bars are calculated using $\sqrt{N}$ where N is the number of counts in each point, along with a 5% CNN classification uncertainty. The counts are determined by examining intervals of the impact velocity range, ranging from 20 to 80km/s, with interval steps of 10km/s. The points are then located at the average velocity of the intervals. The dashed vertical line is located at the 50km/s mark. This is where the Solar Orbiter has v_{rad} = 0 which is at the perihelion and aphelion respectively. The counts and the total of all the categories for perihelion and aphelion sides is shown in table 5.5 and 5.6 respectively.

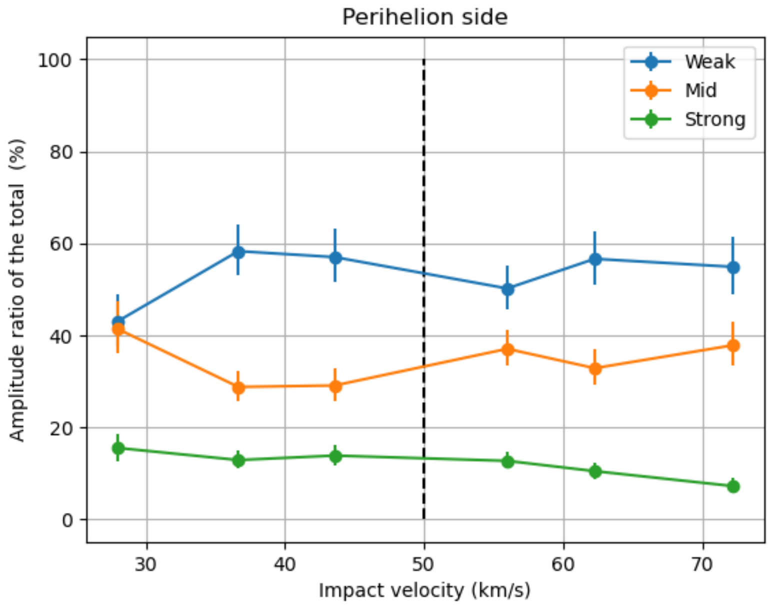
For Figure 5.6**a** the right hand side of the dashed line is when the spacecraft is moving towards the perihelion where the 70km/s mark is at the minimum $v_{rad} \approx -23$km/s as shown in Figure 5.5. The left hand side is then after the perihelion where the 30km/s mark is at the maximum $v_{rad} \approx$ 23km/s. For Figure 5.6**b** the left hand side of the dashed line is from the maximum v_{rad} towards the aphelion. The right hand side is then from the aphelion to minimum v_{rad}.

Table 5.5: Shows the "perihelion side" counts for *weak*, *mid*, *strong* and the total for different impact velocities.

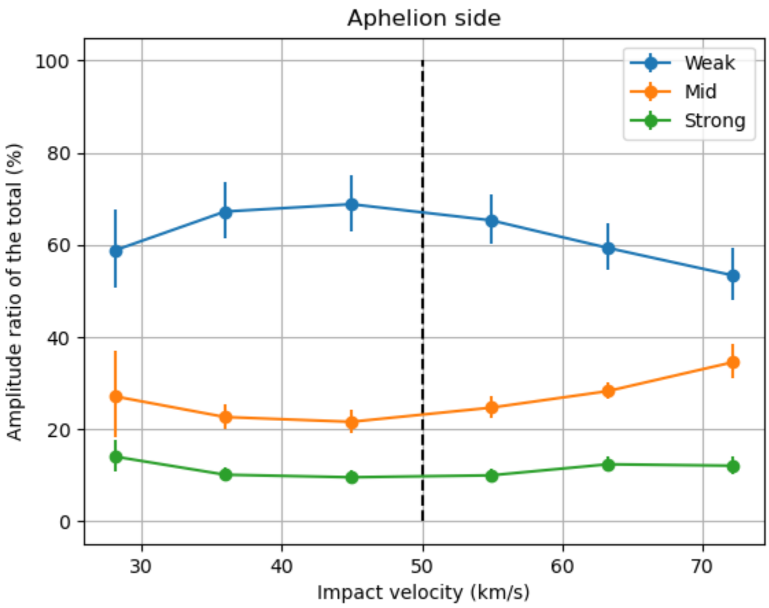
Impact velocity (km/s)	Weak	Mid	Strong	Total
20 – 30	141	136	51	328
30 – 40	448	241	108	837
40 – 50	378	193	92	663
50 – 60	505	373	128	1006
60 – 70	367	213	68	648
70 – 80	258	178	34	470

Table 5.6: Shows the "aphelion side" counts for *weak*, *mid*, *strong* and the total for different impact velocities.

Impact velocity (km/s)	Weak	Mid	Strong	Total
20 – 30	117	54	28	199
30 – 40	677	228	102	1007
40 – 50	669	210	93	972
50 – 60	1059	400	162	1621
60 – 70	886	422	185	1493
70 – 80	318	206	72	596



(a) The ratio of the categories for the perihelion as a function of the impact velocity.



(b) The ratio of the categories for the perihelion as a function of impact velocity

Figure 5.6: Shows the ratio of the categories for the aphelion and aphelion trajectories as a function of the impact velocity.

It does not appear to be a clear trend at the perihelion side in Figure 5.6**a**, we see that the *weak* and *mid* groups are somewhat mirrored while the *strong* is on a decrease towards the perihelion and beyond. One has to take into account that the spacecraft spends significantly less days on this side of the trajectory which means that the counts are low compared to the aphelion side. The *mid* category does have a decrease from 55 – 45km/s which would be on the inbound orbit towards the perihelion. This could be a relative velocity change that decreases the *mid* group but it is not a strong indication of this.

For the aphelion side in Figure 5.6**b** we see an increase of *mid* classified dust impacts and a decrease in *weak* classified dust impacts from 30 to 70km/s. The *strong* classified dust impacts stays roughly constant along this trajectory. The right hand side of the black dashed line is on an inbound course which means the relative velocity increases. This could be an explanation for the increase in the *mid* category and decrease in the *weak* category.

Overall this comparison did not give much insight on how the groups varies with the impact velocity as thought. We do see an increase on the inbound course of the aphelion which can be due to the increase of relative velocity. Other than that maybe there is no variation to consider here. Since the charge production depends also on the mass, it could be that the mass has a more significant influence on the charge production compared to the impact velocity. Of course the impact velocity is not constant which makes this a simplified comparison, nevertheless it is hard to draw a conclusion on this topic.

6

Conclusion

The aim of this master's thesis was to infer information on the mass and impact velocity of the impacting dust particles from RPW measurements. A key aspect of this work was to search for systematic variations of recorded dust signals along the orbit of Solar Orbiter. The data that was used in this thesis is CNN processed data from June 2020 to June 2023. The CNN process is explained in Kvammen et al. (2022). The data cover the period where Solar Orbiter's heliocentric distance ranged from 0.28 - 1.01 AU.

We investigated the voltage amplitudes of the dust impacts that was measured by the RPW. The amplitudes were divided into three groups *weak*, *mid* and *strong*. This was done because *weak* classified dust impacts may include noise and *strong* classified dust impacts may be saturated. We examined the categories both on inbound and outbound trajectories of the orbit and found that the *mid* category increased and *weak* decreased on the inbound trajectory compared to the outbound. A possible explanation for this is the increase of relative velocity between the dust particles and the spacecraft on a inbound trajectory. This would be the case for dust particles that move radially outwards from the Sun. An increase in relative velocity leads to an increase in the impact velocity of the dust particles causing more classified *mid* on a inbound path compared to a outbound path. We also saw that the total of impacts are greater on the inbound trajectory compared to the outbound trajectory which is expected for dust that moves radially outwards from the Sun. Further when we consider the perihelion and aphelion part of the trajectories, it was difficult to find significant trends on the perihelion trajectory. On the other hand when looking at the aphelion trajectory we see an increase of *mid* dust impacts after the aphelion passage.

We derived a charge production threshold using a constant mass of 1.0×10^{-17} kg and assumed a known dust velocity that was derived by Zaslavsky et al. (2021). A comparison of the ratio of the events that exceeded the charge threshold and those that did not was done along the orbit. The comparison showed that there are more dust particles with smaller masses than the fixed mass closer to the Sun. The results suggest that the mass distribution is changing when coming closer to the Sun.

Previous research on the voltage amplitudes conducted by Zaslavsky et al. (2021) shows that the distribution of the voltage measurements follows a linear fit at double logarithmic scales. Upon deriving the voltage distribution for the extended data set that was used in this thesis, we see a similar result for the distribution. We calculated a slope value under the assumption of constant impact velocity which provided insight about the mass distribution given the impact velocity. The slope value we found was at $a = -1.56 \pm 0.18$ and is steeper closer to the Sun. Since we are comparing the counts of our *mid* and *strong* categories, the steepness tells us that there are few classified *strong* amplitudes near the Sun compared to the *mid* classified amplitudes. This observation could also be an indication of smaller dust particles in the vicinity of the Sun as opposed to those found at larger distances.

The investigations of the observations that we made, indicate that the mass distribution changes with the distance from the Sun and that less massive particles are more abundant near the Sun. The results also agree with the assumption that the detected dust particles are moving away from the Sun. We note that there are uncertainties in the results, limiting the conclusiveness of the analysis. The methods used in this work are based on assumptions which we described above. The assumptions can to some extent influence the results. Nonetheless some of the research agrees with what has been previously done. For future work one can look into the saturation effect and see what influences it. This can lead to more information on the *strong* amplitudes and possibly make a more accurate mass distribution of the impacts.

Bibliography

A. Auer and K. Sitte. Detection technique for micrometeoroids using impact ionization. *Earth and Planetary Science Letters*, 4(2):178–183, 1968.

A. Collette, D. Malaspina, and Z. Sternovsky. Characteristic temperatures of hypervelocity dust impact plasmas. *Journal of Geophysical Research: Space Physics*, 121(9):8182–8187, 2016.

A. J. Henriksen. Interstellar dust in the inner heliosphere and impact detection capabilities with esa's solar orbiter spacecraft. Master's thesis, University of Tromsø, 2022.

E. G. Henriksen. Interplanetary dust fluxes observed with parker solar probe. Master's thesis, University of Tromsø, 2020.

Y. V. Khotyaintsev, D. B. Graham, A. Vaivads, K. Steinvall, N. J. Edberg, A. I. Eriksson, E. P. Johansson, L. Sorriso-Valvo, M. Maksimovic, S. Bale, et al. Density fluctuations associated with turbulence and waves-first observations by solar orbiter. *Astronomy & Astrophysics*, 656:A19, 2021.

S. Kočiščák, A. Kvammen, I. Mann, S. H. Sørbye, A. Theodorsen, and A. Zaslavsky. Modeling solar orbiter dust detection rates in the inner heliosphere as a poisson process. *Astronomy & Astrophysics*, 670:A140, 2023a.

S. Kočiščák, I. Mann, N. Meyer-Vernet, A. Theodorsen, J. Vaverka, and A. Zaslavsky. Impact ionization double peaks analyzed in high temporal resolution on solar orbiter. *EGUsphere*, 2023:1–39, 2023b.

A. Kvammen, K. Wickstrøm, S. Kociscak, J. Vaverka, L. Nouzak, A. Zaslavsky, K. Rackovic Babic, A. Gjelsvik, D. Pisa, J. Souček, et al. Machine learning detection of dust impact signals observed by the solar orbiter. 2022.

M. Maksimovic, S. Bale, T. Chust, Y. Khotyaintsev, V. Krasnoselskikh, M. Kretzschmar, D. Plettemeier, H. Rucker, J. Souček, M. Steller, et al. The solar orbiter radio and plasma waves (rpw) instrument. *Astronomy & Astrophysics*, 642:A12, 2020.

I. Mann, H. Kimura, D. A. Biesecker, B. T. Tsurutani, E. Grün, R. B. McKibben, J.-C. Liou, R. M. MacQueen, T. Mukai, M. Guhathakurta, et al. Dust near the sun. *Space science reviews*, 110:269–305, 2004.

I. Mann, L. Nouzák, J. Vaverka, T. Antonsen, Å. Fredriksen, K. Issautier, D. Malaspina, N. Meyer-Vernet, J. Pavl, Z. Sternovsky, et al. Dust observations with antenna measurements and its prospects for observations with parker solar probe and solar orbiter. In *Annales Geophysicae*, volume 37, pages 1121–1140. Copernicus GmbH, 2019.

N. McBride and J. McDonnell. Meteoroid impacts on spacecraft:: sporadics, streams, and the 1999 leonids. *Planetary and Space Science*, 47(8-9):1005–1013, 1999.

N. Meyer-Vernet. Detecting dust with electric sensors in planetary rings, comets and interplanetary space. In *Spacecraft Charging Technology*, volume 476, page 635, 2001.

D. Mueller, R. G. Marsden, O. St. Cyr, H. R. Gilbert, and S. O. Team. Solar orbiter: exploring the sun–heliosphere connection. *Solar Physics*, 285:25–70, 2013.

D. Müller, O. S. Cyr, I. Zouganelis, H. R. Gilbert, R. Marsden, T. Nieves-Chinchilla, E. Antonucci, F. Auchère, D. Berghmans, T. Horbury, et al. The solar orbiter mission-science overview. *Astronomy & Astrophysics*, 642:A1, 2020.

J. Soucek, D. Píša, I. Kolmasova, L. Uhlir, R. Lan, O. Santolík, V. Krupar, O. Kruparova, J. Baše, M. Maksimovic, et al. Solar orbiter radio and plasma waves–time domain sampler: In-flight performance and first results. *Astronomy & Astrophysics*, 656:A26, 2021.

R. Srama, T. J. Ahrens, N. Altobelli, S. Auer, J. Bradley, M. Burton, V. Dikarev, T. Economou, H. Fechtig, M. Görlich, et al. The cassini cosmic dust analyzer. *Space Science Reviews*, 114:465–518, 2004.

V. J. Sterken, N. Altobelli, S. Kempf, G. Schwehm, R. Srama, and E. Grün. The flow of interstellar dust into the solar system. *Astronomy & Astrophysics*, 538: A102, 2012.

M. Wilck and I. Mann. Radiation pressure forces on “typical” interplanetary dust grains. *Planetary and Space Science*, 44(5):493–499, 1996.

A. Zaslavsky. personal communication.

A. Zaslavsky. Floating potential perturbations due to micrometeoroid impacts: Theory and application to s/waves data. *Journal of Geophysical Research: Space Physics*, 120(2):855–867, 2015.

A. Zaslavsky, I. Mann, J. Soucek, A. Czechowski, D. Píša, J. Vaverka, N. Meyer-Vernet, M. Maksimovic, E. Lorfèvre, K. Issautier, et al. First dust measurements with the solar orbiter radio and plasma wave instrument. *Astronomy & Astrophysics*, 656:A30, 2021.

A
Data files

The data files of the RPW measurements can be downloaded from this link: https://rpw.lesia.obspm.fr/roc/data/pub/solo/rpw/data/

The ephemeris for Solar Orbiter can be generated and downloaded from this link: https://ssd.jpl.nasa.gov/horizons/app.html#/

The CNN files were provided to me privately by Samuel Kočiščák.

B

Code

This appendix shows the scripts that was used in this thesis in Python language. Listing B.1 is used to read the cdf and cnn files and loads the processed data into a list that we use to plot the graphs used in this thesis. Listing B.2 is used to plot the trajectories, voltage distribution and calculate the slope value and plot it.

```
#Importing packages
import numpy as np
import cdflib
import pandas as pd
import glob
from datetime import datetime

import pickle

# inserts the datafile into a list
cdf_list = glob.glob('C:\\06\\*.cdf')
#highrate_list = glob.glob('C:\\Users\\alenf\\PycharmProjects\\
    Project Paper\\data\\test-data_highrate\\*.txt')
data_11_2022 = glob.glob('C:\\Users\\afe029\\OneDrive - UiT
    Office 365\\Masters\\data\\Label-MaxAmplitude\\*.txt')

# constant
AU = 1.496e+8

def data_product_single_CDF(file):
    """
    Reads the CDF files and extracts the CNN data files
    corresponding to the CDF File.
```

```
    Also reads for both sample rates 262137.5 and 524275.0
    :param file:
    :return: amplitude, date, mean value and median
    """
    cdf_file = cdflib.CDF(file)
    voltage = cdf_file.varget('waveform_data_voltage')

    monopole_1 = voltage[:, 2, :] - voltage[:, 1, :]
    monopole_2 = voltage[:, 2, :]
    monopole_3 = voltage[:, 2, :] - voltage[:, 1, :] - voltage[:, 0, :]

    waveform = np.array([monopole_1, monopole_2, monopole_3])
    waveform = np.transpose(waveform, (1, 0, 2))
    quality = cdf_file.varget('QUALITY_FACT')
    flagger = quality == 65535
    sample_rate = cdf_file.varget('SAMPLING_RATE')
    yyyymmdd = file[-16:-8]
    ## two sample rates,262137.5 and 524275.0

    if sample_rate[0] == 262137.5:
                cnn_file = glob.glob(
                    f'C:\\Users\\afe029\\OneDrive - UiT Office 365\\Masters\\data\\cnn label\\2022\\2022\\*{yyyymmdd}*.txt')
                cnn_filer = glob.glob(
                    f'C:\\Users\\afe029\\OneDrive - UiT Office 365\\Masters\\data\\Label-MaxAmplitude\\*{yyyymmdd}*.txt')
                index_list = np.zeros(0, dtype=np.uint32)

                if any(yyyymmdd in file for file in cnn_file):
                    print(yyyymmdd)
                    cnn = pd.read_csv(cnn_file[0])
                    dust = np.array(cnn['Label'], dtype=bool)

                    normal_index = np.arange(len(waveform))

                    arange_index = np.append(normal_index[flagger == 1], normal_index[flagger == 0])
                    label_finder = np.array(cnn['Index'])[dust] - 1
                    indice = arange_index[label_finder]
                    waveformz = waveform[indice,:,:]

                    amplitude = np.zeros(len(waveformz[:,0,0]))
                    for i in range(len(waveformz)):
                        bias_wf1 = waveformz[i,0,:]
                        bias_wf2 = waveformz[i,1,:]
                        bias_wf3 = waveformz[i,2,:]

                        mean_wf1 = np.mean(bias_wf1)
                        mean_wf2 = np.mean(bias_wf2)
                        mean_wf3 = np.mean(bias_wf3)

```

```
                    wf1 = bias_wf1 - mean_wf1
                    wf2 = bias_wf2 - mean_wf2
                    wf3 = bias_wf3 - mean_wf3

                    max1 = max(wf1)
                    max2 = max(wf2)
                    max3 = max(wf3)


                    amplitude[i] = (min(max1,max2,max3))

                average = np.mean(amplitude)
                median = np.median(amplitude)

                return amplitude, yyyymmdd, average, median, waveformz

            if any(yyyymmdd in filer for filer in cnn_filer):
                print(yyyymmdd)
                for filer in cnn_filer:
                    index = int(filer[-7:-4])
                    index_list = np.append(index_list, index)

                normal_index = np.arange(len(waveform))

                arange_index = np.append(normal_index[flagger == 1], normal_index[flagger == 0])
                indice = arange_index[index_list]

                waveformzz = waveform[indice,:,:]

                amplitude = np.zeros(len(waveformzz[:, 0, 0]))
                for i in range(len(waveformzz)):
                    bias_wf1 = waveformzz[i, 0, :]
                    bias_wf2 = waveformzz[i, 1, :]
                    bias_wf3 = waveformzz[i, 2, :]

                    mean_wf1 = np.mean(bias_wf1)
                    mean_wf2 = np.mean(bias_wf2)
                    mean_wf3 = np.mean(bias_wf3)

                    wf1 = bias_wf1 - mean_wf1
                    wf2 = bias_wf2 - mean_wf2
                    wf3 = bias_wf3 - mean_wf3

                    max1 = max(wf1)
                    max2 = max(wf2)
                    max3 = max(wf3)

                    amplitude[i] = (min(max1, max2, max3))
```

```
                    average = np.mean(amplitude)
                    median = np.median(amplitude)

                    return amplitude, yyyymmdd, average, median, waveformzz

    elif sample_rate[0] == 524275.0:
                cnn_files = glob.glob(f'C:\\cnn_high-sample\\cnn_high_sampling\\*{yyyymmdd}*.txt')
                indexes = np.zeros(0, dtype=np.uint32)
                print(cnn_files)
                for file in cnn_files:
                    index = int(file[-8:-4])

                    indexes = np.append(indexes, index)

                waveformz = waveform[indexes, :, :]

                amplitude2 = np.zeros(len(waveformz[:, 0, 0]))
                for i in range(len(waveformz)):
                    bias_wf1 = waveformz[i, 0, :]
                    bias_wf2 = waveformz[i, 1, :]
                    bias_wf3 = waveformz[i, 2, :]

                    mean_wf1 = np.mean(bias_wf1)
                    mean_wf2 = np.mean(bias_wf2)
                    mean_wf3 = np.mean(bias_wf3)

                    wf1 = bias_wf1 - mean_wf1
                    wf2 = bias_wf2 - mean_wf2
                    wf3 = bias_wf3 - mean_wf3

                    max1 = max(wf1)
                    max2 = max(wf2)
                    max3 = max(wf3)

                    amplitude2[i] = (min(max1, max2, max3))


                average2 = np.mean(amplitude2)
                median2 = np.median(amplitude2)


                return amplitude2, yyyymmdd, average2, median2, waveformz


def read_ephemeris(date):
    """
    This function reads a txt file. The txt file is a generated ephemeris from the link in the appendix.
    Takes the a date parameter and calculates the heliocentric distance, tangetial velocity and radial velocity of the spacecraft at that date.
```

```
    :param date: date of the cdf file
    :return: helicentric distance, tangetial velocity, radial
velocity
    """

    hae_r = np.zeros(0)
    hae_v = np.zeros(0)

    file = pd.read_csv('C:\\Users\\afe029\\OneDrive - UiT
Office 365\\Masters\\data\\horizons_results.txt')
    calender = file['Calendar Date (TDB)']
    cord_x = file['X']
    cord_y = file['Y']
    cord_z = file['Z']

    cord_vx = file['VX']
    cord_vy = file['VY']
    cord_vz = file['VZ']


    julian = file['JDTDB']
    date_in_YY_MM_DD = []
    yyyymmdd_list = []
    for i in range(len(julian)):
        date_in_YY_MM_DD.append(str(calender[i][6:17]))

        dt_object1 = datetime.strptime(date_in_YY_MM_DD[i], "%Y
-%b-%d")
        yyyymmdd = datetime.strftime(dt_object1, "%Y%m%d")
        yyyymmdd_list.append(str(yyyymmdd))

    if date in yyyymmdd_list:
        index = yyyymmdd_list.index(date)
        x = cord_x[index]
        y = cord_y[index]
        z = cord_z[index]

        vx = cord_vx[index]
        vy = cord_vy[index]
        vz = cord_vz[index]


        hae_r = np.append(hae_r, [x, y, z])
        hae_v = np.append(hae_v, [vx, vy, vz])

        hae_r = np.reshape(hae_r, ((len(hae_r) // 3, 3)))
        hae_v = np.reshape(hae_v, ((len(hae_v) // 3, 3)))
        hae_phi = np.degrees(np.arctan2(hae_r[:, 1], hae_r[:,
0]))

        hae_radius = np.linalg.norm(hae_r) / AU # radius of
Solar orbiter in reference to the Sun in AU

```

```
        radial_v = np.zeros(len(hae_r[:, 0]))
        tangential_v = np.zeros(len(hae_r[:, 0]))

        for i in range(len(hae_r[:, 0])):
            unit_radial = hae_r[i, :] / np.linalg.norm(hae_r[i, :])
            radial_v[i] = np.inner(unit_radial, hae_v[i, :])
            tangential_v[i] = np.linalg.norm(hae_v[i, :] - radial_v[i] * unit_radial)


        return hae_radius, radial_v, tangential_v

# Makes numpy arrays
totals_impacts= np.zeros(0)
dates = np.zeros(0)
weaks = np.zeros(0)
strongs = np.zeros(0)
mids = np.zeros(0)

weaks_2 = np.zeros(0)
strongs_2 = np.zeros(0)
mids_2 = np.zeros(0)

solO_radius = np.zeros(0)
tang_velocity = np.zeros(0)
radial_velocity = np.zeros(0)
phi = np.zeros(0)

means = np.zeros(0)
medians = np.zeros(0)

weak_Q = np.zeros(0)
strong_Q = np.zeros(0)
threshold_Q = np.zeros(0)

result_array = np.empty((2, 0))

mass = 1e-17
v_dust = 50

def Q_finder(v_rad, V_peak):
    """
    Calculates the charge threshold and find the charge that corresponds to the voltage amplitude.
    :param v_rad: radial velocity of the spacecraft
    :param V_peak: peak amplitude of the voltage measurement
    :return: charge value, charge threshold
    """
    C_sc = 3.55e-10
    Gamma = 0.36
    xi = 0.7
    vel = v_dust + v_rad
    Q_value = (C_sc * V_peak) / Gamma
```

```
    amplitude_peak = (xi * mass * (vel**3.5) * Gamma) / C_sc


    Q_threshold = 0.7 * mass * (vel**3.5)

    return Q_value, Q_threshold, amplitude_peak

# Numpy arrays
Strong_V = np.zeros(0)
weak_V = np.zeros(0)

charge_error_low_limit = np.zeros(0)
charge_error_upper_limit = np.zeros(0)
charge_error_boundary = np.zeros(0)

# Reads through all the cdf files and returns the needed data
for file in cdf_list:
    try:
        print(file)
        amplitude, date, mean, median, voltage = data_product_single_CDF(file)
        radius, radial_v, tan_v, hae_phi  = read_ephemeris(date)

        Q, Q_threshold, V_peak = Q_finder(radial_v, amplitude)

        charge_great = sum(Q < Q_threshold)
        charge_low = sum(Q > Q_threshold)

        charge_error_upper = sum(Q > 1.5* Q_threshold)
        charge_error_lower= sum(Q < 0.75 * Q_threshold)
        charge_error_left = (charge_low + charge_great) - charge_error_lower - charge_error_upper
        print(charge_error_upper)
        print(charge_error_lower)

        charge_error_low_limit = np.append(charge_error_low_limit, charge_error_lower)
        charge_error_upper_limit = np.append(charge_error_upper_limit, charge_error_upper)
        charge_error_boundary = np.append(charge_error_boundary, charge_error_left)

        amplitude_greater = sum(V_peak < amplitude)
        amplitude_less = sum (V_peak > amplitude)

        velocity_values = np.full_like(amplitude, radial_v)

        stacker = np.vstack((amplitude, velocity_values))

        result_array = np.hstack((result_array,stacker))


```

```
        dates = np.append(dates, date)
        solO_radius = np.append(solO_radius, radius)
        tang_velocity = np.append(tang_velocity, tan_v)
        radial_velocity = np. append(radial_velocity, radial_v)
        phi = np.append(phi,  hae_phi)

        threshold_Q = np.append(threshold_Q, Q_threshold)

        totals_impacts = np.append(totals_impacts, len(amplitude))
        weak = sum((amplitude * 1e3) < 50)
        strong = sum((amplitude * 1e3) > 250)
        mid = len(amplitude) - strong - weak

        weak_2 = sum((amplitude * 1e3) < 20)
        strong_2 = sum((amplitude * 1e3) > 200)
        mid_2 = len(amplitude) - strong_2 - weak_2


        weaks = np.append(weaks, weak)
        strongs = np.append(strongs, strong)
        mids = np.append(mids, mid)

        weaks_2 = np.append(weaks_2, weak_2)
        strongs_2 = np.append(strongs_2, strong_2)
        mids_2 = np.append(mids_2, mid_2)

        weak_V = np.append(weak_V, amplitude_less)
        Strong_V = np.append(Strong_V, amplitude_greater)

        weak_Q = np.append(weak_Q, charge_low)
        strong_Q = np.append(strong_Q, charge_great)

        means = np.append(means, mean)
        medians = np.append(medians, median)
    except IndexError:
        pass



#Puts all the data into a single list
Data_array = [weaks,mids, strongs, totals_impacts, dates, solO_radius, tang_velocity, radial_velocity, phi,voltage, weak_Q, strong_Q, threshold_Q, result_array, weak_V, Strong_V, weaks_2,mids_2, strongs_2, charge_error_low_limit, charge_error_upper_limit, charge_error_boundary]
#dumps the data into a file
data_file = open('data_product_long_long_long', 'wb')
pickle.dump(Data_array,data_file)
data_file.close()
```

Listing B.1: Code used to read the Solar Orbiter data files and CNN files

```
import numpy as np
```

```
import pickle
import matplotlib.pyplot as plt
import datetime as dt
import matplotlib.dates as mdates
import scipy.signal as sps

data_file = open('data_product_long_long_long', 'rb')

#Data_array = [weaks,mids, strongs, totals_impacts, dates,
    solO_radius, tang_velocity, radial_velocity, phi, voltage,
    weak_Q, strong_Q, threshold_Q, amplitude_velocity[2,:],
    weak_V, strong_V,
# weaks_2, mids_2, strongs_2, charge_error_lower,
    charge_error_upper, charge_error_whatsleft

data_array = pickle.load(data_file)

# Assign the data products from data_array into single arrays
weak_V = data_array[14]
strong_V = data_array[15]
twoD_array = data_array[13]

amplitude_array = twoD_array[0,:] *1e3
velocity_array = twoD_array[1,:]


weak_Q = data_array[10]
strong_Q = data_array[11]

charge_error_lower = data_array[19]
charge_error_upper = data_array[20]
charge_error_within = data_array[21]


weaks=data_array[16] #number of impacts < 20mV
mids = data_array[17] # number of impacts 20 < mV < 200
strongs = data_array[18] # number of impacts > 200

total_impacts = data_array[3] #total impacts for each day


date = data_array[4]
SolO_radius = data_array[5]
tan_velocity = data_array[6]
radial_velocity = data_array[7]
phi = data_array[8]

#Plots the voltage amplitude distribution with logarithmic
    scales

"""hist, edges = np.histogram(amplitude_array, bins=np.logspace
    (np.log10(min(amplitude_array)), np.log10(max(
    amplitude_array)), 75), density=True)

```

```
# Calculate bin centers for the scatter plot
bin_centers = (edges[:-1] + edges[1:]) / 2

# Calculate errors using the given formula
bin_width = edges[1] - edges[0]
total_events = len(amplitude_array)
errors = np.sqrt(hist) / (bin_width * total_events)


# Create a scatter plot with error bars and log axes
plt.errorbar(bin_centers, hist, yerr=errors, fmt='o', color='black', alpha=0.75, capsize=3)
plt.vlines([20,200], ymin=1e-6, ymax=1, linestyles='--', colors='k')
plt.text(s= 'Weak', x= 2, y=10**-4)
plt.text(s = 'Mid', x = 50, y =10**-4)
plt.text(s = 'Strong', x = 300, y = 10**-1)


# Set logarithmic scales for both axes
plt.xscale('log')
plt.yscale('log')
plt.ylim(1e-6,1)
plt.grid()

# Set labels and title
plt.xlabel('Monopole peak voltage (mV)')
plt.ylabel('Normalized density [1/mV)')

# Show the plot
plt.show()"""


#Plots the heliocentric distance and the radial velocity
"""dates_x_axis = [dt.datetime.strptime(d,'%Y%m%d').date() for d in date]

fig, axs = plt.subplots(2)

axs[0].set_title('Solar Orbiter')
axs[0].plot(dates_x_axis,SolO_radius)
axs[0].set_ylabel('Distance from sun (AU)')

axs[1].plot(dates_x_axis,radial_velocity)
axs[1].set_ylabel(' Radial Velocity (km/s)')
axs[0].grid()
axs[1].grid()

axs[0].xaxis.set_major_formatter(mdates.DateFormatter('%d/%m/%Y'))
axs[0].xaxis.set_major_locator(mdates.DayLocator(interval=225))
axs[1].xaxis.set_major_formatter(mdates.DateFormatter('%d/%m/%Y'))
axs[1].xaxis.set_major_locator(mdates.DayLocator(interval=225))
```

```
plt.show()"""


def data_table(a,b):
    """
    Caluclates the weak, mid, strong and etc. within the intervals a and b
    :param a: lower limit of interval
    :param b: upper limit of interval
    """
    total_Q = strong_Q + weak_Q
    distance_index = np.where((SolO_radius >= a)& (SolO_radius <= b))

    weak = sum(weaks[distance_index])
    mid = sum(mids[distance_index])
    strong = sum(strongs[distance_index])
    total = sum(total_impacts[distance_index])
    Q_total = sum(total_Q[distance_index])

    Weaks_Q = sum(weak_Q[distance_index])
    strongs_Q = sum(strong_Q[distance_index])

    high_Q = sum(charge_error_upper[distance_index])
    low_Q = sum(charge_error_lower[distance_index])
    within_Q = sum(charge_error_within[distance_index])

    Q_error = sum(charge_error_within[distance_index])


    weak_ratio = weak / total
    mid_ratio = mid / total
    strong_ratio = strong / total

    Q_weak_ratio = Weaks_Q / Q_total
    Q_strong_ratio = strongs_Q / Q_total

    Q_errors = Q_error / Q_total



    return weak_ratio, mid_ratio, strong_ratio, weak, mid, strong, Weaks_Q, strongs_Q, Q_weak_ratio, Q_strong_ratio, Q_total, total, Q_errors, high_Q, low_Q, within_Q

#distances
distance_01 = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

#empty arrays
weak_list = np.zeros(0)
mid_list = np.zeros(0)
strong_list = np.zeros(0)
weak_list_number = np.zeros(0)
```

```python
mid_list_number = np.zeros(0)
strong_list_number = np.zeros(0)
Q_weak_list = np.zeros(0)
Q_strong_list = np.zeros(0)
Q_weak_list_r = np.zeros(0)
Q_strong_list_r = np.zeros(0)
totals = np.zeros(0)
Q_totals = np.zeros(0)

error_Q = np.zeros(0)

high_Q_list = np.zeros(0)
low_Q_list = np.zeros(0)
within_Q_list = np.zeros(0)


#assign values into empty arrays with the data_table function
for i in range(len(distance_01) -1):
    try:
        weak_ratio, mid_ratio, strong_ratio, weak_number,
    mids_number, strong_number, Q_weak, Q_strong, Q_weak_ratio,
     Q_strong_ratio, Q_total, total, Q_error, high_Q, low_Q,
    within_Q = data_table(distance_01[i], distance_01[i+1])

        weak_list = np.append(weak_list, weak_ratio)
        mid_list = np.append(mid_list, mid_ratio)
        strong_list = np.append(strong_list, strong_ratio)
        weak_list_number = np.append(weak_list_number,
    weak_number)
        mid_list_number = np.append(mid_list_number,
    mids_number)
        strong_list_number =  np.append(strong_list_number,
    strong_number)
        Q_weak_list = np.append(Q_weak_list, Q_weak)
        Q_strong_list = np.append(Q_strong_list, Q_strong)
        Q_weak_list_r = np.append(Q_weak_list_r, Q_weak_ratio)
        Q_strong_list_r = np.append(Q_strong_list_r,
    Q_strong_ratio)
        Q_totals = np.append(Q_totals, Q_total)
        totals = np.append(totals, total)
        error_Q = np.append(error_Q, Q_error)

        high_Q_list = np.append(high_Q_list, high_Q)
        low_Q_list = np.append(low_Q_list, low_Q)
        within_Q_list = np.append(within_Q_list, within_Q)

    except ZeroDivisionError:
        pass
        #weak_list = np.append(weak_list, 0)
        #mid_list = np.append(mid_list, 0)
        #strong_list = np.append(strong_list, 0)



```

```

def charge_error_bars():
    """
    calculates the error bar for the charge production that is calculated with a fixed mass
    :return:
    """
    points_strong = Q_strong_list_r
    points_lower = np.zeros(0)
    points_higher = np.zeros(0)


    for i in range(len(Q_weak_list_r)):
        points_lower = np.append(points_lower, low_Q_list[i] / Q_totals[i])
        points_higher = np.append(points_higher, (low_Q_list[i] + within_Q_list[i]) / Q_totals[i])




    error_p = points_higher - points_strong
    error_m =  points_strong - points_lower


    errorbar = np.array([error_m, error_p])

    return errorbar

charge_error = charge_error_bars()


## Plots the charge production ratio that corresponds to masses less or greater than 1e-17kg
"""plt.errorbar(np.array(distance_01[2:]) + 0.05, Q_weak_list_r * 100 , fmt='o-',label = "Mass < 1e-17 Kg", capsize=3,yerr=(charge_error[1,:] * 100, charge_error[0,:] * 100))
plt.errorbar(np.array(distance_01[2:]) + 0.05, Q_strong_list_r * 100 , fmt='o-',label = "Mass > 1e-17 Kg", capsize=3, yerr=(charge_error[0,:] * 100, charge_error[1,:] * 100))
plt.xlabel('Distance from the Sun (AU)')
plt.ylabel('Ratio (%)')
plt.grid()
plt.legend()
plt.show()

plt.plot(distance_01[2:], weak_list_number, label= 'weak amplitude >50 mV')
plt.plot(distance_01[2:], mid_list_number, label = 'mid amplitude 50 > mV > 250')
plt.plot(distance_01[2:], strong_list_number, label = 'strong amplitude 250>mV')
plt.xlabel('Distance (AU)')
plt.ylabel('Number of events')
```

```
plt.grid()
#plt.legend(loc= "upper left")
plt.show()"""



#finds the minimum and maxima for the heliocentric distance
maxima = sps.argrelextrema(SolO_radius, np.greater)
minimum = sps.argrelextrema(SolO_radius, np.less)

maximum_rad_v = sps.argrelextrema(radial_velocity, np.greater)
minimum_rad_v = sps.argrelextrema(radial_velocity, np.less)

def calculate_error(counts, total):
    """
    Calculates the error bar with a 5% classification error as
    well as the square root of counts
    :param counts: counts of a specific category
    :param total: total for all the categories
    :return: errorbar in absolute value
    """
    # Calculate the error with a 5% confidence interval
    error_margin = 0.05 * total
    lower_bound = (counts - np.sqrt(counts)) / (total +
    error_margin)
    upper_bound = (counts + np.sqrt(counts)) / (total -
    error_margin)

    regular = counts / total

    error_upper = upper_bound - regular
    error_lower = lower_bound - regular

    # Calculate the relative error as a percentage of the total
    relative_error = np.array([error_lower, error_upper]) * 100

    return np.abs(relative_error)


perihelion_velocity = [radial_velocity[0:33], radial_velocity
    [179:237], radial_velocity[385:452], radial_velocity
    [584:617], radial_velocity[775:807], radial_velocity
    [912:942] ]
aphelion_velocity = [radial_velocity[34:178], radial_velocity
    [238:384], radial_velocity[453:583], radial_velocity
    [618:774], radial_velocity[808:911], radial_velocity
    [943:-1]]

def new_velocity():

    """
    Finds the perihelion and aphelion trajectories and plots
    them as a function of the impact velocity
    :return:
```

```
    """

    average_distance_03_inward = np.mean(np.concatenate((radial_velocity[584:590],radial_velocity[775:783], radial_velocity[912:918]))) + 50
    average_distance_03_outward = np.mean(np.concatenate((radial_velocity[572:584],radial_velocity[761:775], radial_velocity[899:912]))) + 50


    average_distance_04_inward = np.mean(np.concatenate((radial_velocity[179:188],  radial_velocity[385:396], radial_velocity[590:598], radial_velocity[783:788],
        radial_velocity[918:923]))) + 50
    average_distance_04_outward = np.mean(np.concatenate((radial_velocity[153:179],  radial_velocity[374:385], radial_velocity[542:576], radial_velocity[728:761],
        radial_velocity[884:899]))) + 50

    average_distance_05_inward = np.mean(np.concatenate((radial_velocity[188:206], radial_velocity[396:418], radial_velocity[598:600], radial_velocity[788:791], radial_velocity[918:923], radial_velocity[923:927]))) + 50
    average_distance_05_outward = np.mean(np.concatenate((radial_velocity[93:153], radial_velocity[317:374], radial_velocity[505:542], radial_velocity[692:728], radial_velocity[860:884]))) + 50

    average_distance_06_inward = np.mean(np.concatenate((radial_velocity[0:13], radial_velocity[206:222], radial_velocity[418:435], radial_velocity[600:604], radial_velocity[791:794], radial_velocity[927:923]))) + 50
    average_distance_06_outward = np.mean(np.concatenate((radial_velocity[45:93], radial_velocity[260:317], radial_velocity[482:505], radial_velocity[655:692], radial_velocity[839:860], radial_velocity[972:978]))) + 50

    average_distance_07_inward = np.mean(np.concatenate((radial_velocity[13:34], radial_velocity[222:238], radial_velocity[435:453], radial_velocity[604:610], radial_velocity[794:799], radial_velocity[930:935]))) + 50
    average_distance_07_outward = np.mean(np.concatenate((radial_velocity[34:45], radial_velocity[238:260], radial_velocity[453:482], radial_velocity[624:655], radial_velocity[817:839], radial_velocity[949:971]))) + 50


    average_distance_08_inward = np.mean(np.concatenate((radial_velocity[610:618], radial_velocity[799:808], radial_velocity[936:943]))) +50
    average_distance_08_outward = np.mean(np.concatenate((radial_velocity[618:624], radial_velocity[808:817], radial_velocity[943:949]))) + 50

```

```

    inward_08_weak = sum(weaks[584:590]) + sum(weaks[775:783]) + sum(weaks[912:918])
    outward_08_weak = sum(weaks[572:584]) + sum(weaks[761:775]) + sum(weaks[899:912])

    inward_07_weak = sum(weaks[179:188]) + sum(weaks[385:396]) + sum(weaks[590:598]) + sum(weaks[783:788]) + sum(
        weaks[918:923])
    outward_07_weak = sum(weaks[153:179]) + sum(weaks[374:385]) + sum(weaks[542:572]) + sum(weaks[728:761]) + sum(
        weaks[884:899])

    inward_06_weak = sum(weaks[188:206]) + sum(weaks[396:418]) + sum(weaks[598:600]) + sum(weaks[788:791]) + sum(
        weaks[918:923]) + sum(weaks[923:927])
    outward_06_weak = sum(weaks[93:153]) + sum(weaks[317:374]) + sum(weaks[505:542]) + sum(weaks[692:728]) + sum(
        weaks[860:884])

    inward_05_weak = sum(weaks[0:13]) + sum(weaks[206:222]) + sum(weaks[418:435]) + sum(weaks[600:604]) + sum(
        weaks[791:794]) + sum(weaks[927:930])
    outward_05_weak = sum(weaks[45:93]) + sum(weaks[260:317]) + sum(weaks[482:505]) + sum(weaks[655:692]) + sum(
        weaks[839:860]) + sum(weaks[972:978])

    inward_04_weak = sum(weaks[13:34]) + sum(weaks[222:238]) + sum(weaks[435:453]) + sum(weaks[604:610]) + sum(
        weaks[794:799]) + sum(weaks[930:935])
    outward_04_weak = sum(weaks[34:45]) + sum(weaks[238:260]) + sum(weaks[453:482]) + sum(weaks[624:655]) + sum(
        weaks[817:839]) + sum(weaks[949:971])

    inward_03_weak = sum(weaks[610:618]) + sum(weaks[799:808]) + sum(weaks[936:943])
    outward_03_weak = sum(weaks[618:624]) + sum(weaks[808:817]) + sum(weaks[943:949])

    inward_08_mid = sum(mids[584:590]) + sum(mids[775:783]) + sum(mids[912:918])
    outward_08_mid = sum(mids[572:584]) + sum(mids[761:775]) + sum(mids[899:912])

    inward_07_mid = sum(mids[179:188]) + sum(mids[385:396]) + sum(mids[590:598]) + sum(mids[783:788]) + sum(
        mids[918:922])
    outward_07_mid = sum(mids[153:179]) + sum(mids[374:385]) + sum(mids[542:572]) + sum(mids[728:761]) + sum(
        mids[884:899])

    inward_06_mid = sum(mids[188:206]) + sum(mids[385:418]) + sum(mids[598:600]) + sum(mids[788:791]) + sum(
        mids[918:923]) + sum(mids[923:927])
```

```
    outward_06_mid = sum(mids[93:153]) + sum(mids[317:385]) + sum(mids[505:542]) + sum(mids[692:728]) + sum(
        mids[860:884])

    inward_05_mid = sum(mids[0:13]) + sum(mids[206:222]) + sum(mids[418:435]) + sum(mids[600:604]) + sum(
        mids[791:794]) + sum(mids[927:930])
    outward_05_mid = sum(mids[45:93]) + sum(mids[260:317]) + sum(mids[482:505]) + sum(mids[655:692]) + sum(
        mids[839:860]) + sum(mids[972:978])

    inward_04_mid = sum(mids[13:34]) + sum(mids[222:238]) + sum(mids[435:453]) + sum(mids[604:610]) + sum(
        mids[794:799]) + sum(mids[930:935])
    outward_04_mid = sum(mids[34:45]) + sum(mids[238:260]) + sum(mids[453:482]) + sum(mids[624:655]) + sum(
        mids[817:839]) + sum(mids[949:971])

    inward_03_mid = sum(mids[610:618]) + sum(mids[799:808]) + sum(mids[936:943])
    outward_03_mid = sum(mids[618:624]) + sum(mids[808:817]) + sum(mids[943:949])

    inward_08_strong = sum(strongs[584:590]) + sum(strongs[775:783]) + sum(strongs[912:918])
    outward_08_strong = sum(strongs[572:584]) + sum(strongs[761:775]) + sum(strongs[899:912])

    inward_07_strong = sum(strongs[179:188]) + sum(strongs[385:396]) + sum(strongs[590:598]) + sum(
        strongs[783:788]) + sum(strongs[918:923])
    outward_07_strong = sum(strongs[153:179]) + sum(strongs[374:385]) + sum(strongs[542:572]) + sum(
        strongs[728:761]) + sum(strongs[884:899])

    inward_06_strong = sum(strongs[188:206]) + sum(strongs[385:418]) + sum(strongs[598:600]) + sum(
        strongs[788:791]) + sum(strongs[918:923]) + sum(strongs[923:927])
    outward_06_strong = sum(strongs[93:153]) + sum(strongs[317:385]) + sum(strongs[505:542]) + sum(
        strongs[692:728]) + sum(strongs[860:884])

    inward_05_strong = sum(strongs[0:13]) + sum(strongs[206:222]) + sum(strongs[418:435]) + sum(strongs[600:604]) + sum(
        strongs[791:794]) + sum(strongs[927:930])
    outward_05_strong = sum(strongs[45:93]) + sum(strongs[260:317]) + sum(strongs[482:505]) + sum(
        strongs[655:692]) + sum(strongs[839:860]) + sum(strongs[972:978])

    inward_04_strong = sum(strongs[13:34]) + sum(strongs[222:238]) + sum(strongs[435:453]) + sum(
```

```
        strongs[604:610]) + sum(strongs[794:799]) + sum(strongs[930:935])
    outward_04_strong = sum(strongs[34:45]) + sum(strongs[238:260]) + sum(strongs[453:482]) + sum(
        strongs[624:655]) + sum(strongs[817:839]) + sum(strongs[949:971])

    inward_03_strong = sum(strongs[610:618]) + sum(strongs[799:808]) + sum(strongs[936:943])
    outward_03_strong = sum(strongs[618:624]) + sum(strongs[808:817]) + sum(strongs[943:949])

    inward_list_weak = np.array([inward_03_weak, inward_04_weak, inward_05_weak, inward_06_weak, inward_07_weak, inward_08_weak])
    inward_list_mid = np.array([inward_03_mid, inward_04_mid, inward_05_mid, inward_06_mid, inward_07_mid, inward_08_mid])
    inward_list_strong = np.array([inward_03_strong, inward_04_strong, inward_05_strong, inward_06_strong, inward_07_strong, inward_08_strong])

    outward_list_weak = np.array([outward_03_weak, outward_04_weak, outward_05_weak, outward_06_weak, outward_07_weak, outward_08_weak])
    outward_list_mid = np.array([outward_03_mid, outward_04_mid, outward_05_mid, outward_06_mid, outward_07_mid, outward_08_mid])
    outward_list_strong = np.array([outward_03_strong, outward_04_strong, outward_05_strong, outward_06_strong,
                                outward_07_strong, outward_08_strong])

    total_inward = np.array([inward_03_weak + inward_03_mid + inward_03_strong,inward_04_weak + inward_04_mid + inward_04_strong,
                                inward_05_weak + inward_05_mid + inward_05_strong,inward_06_weak + inward_06_mid + inward_06_strong,
                                inward_07_weak + inward_07_mid + inward_07_strong, inward_08_weak + inward_08_mid + inward_08_strong])

    total_outward = np.array(
        [outward_03_weak + outward_03_mid + outward_03_strong, outward_04_weak + outward_04_mid + outward_04_strong,
         outward_05_weak + outward_05_mid + outward_05_strong, outward_06_weak + outward_06_mid + outward_06_strong,
         outward_07_weak + outward_07_mid + outward_07_strong, outward_08_weak + outward_08_mid + outward_08_strong])

    weak_error_in_lower = np.zeros(0)
    weak_error_in_upper = np.zeros(0)

```

```
    mid_error_in_lower = np.zeros(0)
    mid_error_in_upper = np.zeros(0)

    strong_error_in_lower = np.zeros(0)
    strong_error_in_upper = np.zeros(0)

    weak_error_out_lower = np.zeros(0)
    weak_error_out_upper = np.zeros(0)

    mid_error_out_lower = np.zeros(0)
    mid_error_out_upper = np.zeros(0)

    strong_error_out_lower = np.zeros(0)
    strong_error_out_upper = np.zeros(0)


    for i in range(len(total_outward)):
        error_weak_inward = calculate_error(inward_list_weak[i
], total_inward[i])
        error_mid_inward = calculate_error(inward_list_mid[i],
total_inward[i])
        error_strong_inward = calculate_error(
inward_list_strong[i], total_inward[i])

        error_weak_outward = calculate_error(outward_list_weak[
i], total_outward[i])
        error_mid_outward = calculate_error(inward_list_mid[i],
 total_outward[i])
        error_strong_outward = calculate_error(
outward_list_strong[i], total_outward[i])

        weak_error_in_lower = np.append(weak_error_in_lower,
error_weak_inward[0])
        weak_error_in_upper = np.append(weak_error_in_upper,
error_weak_inward[1])

        mid_error_in_lower = np.append(mid_error_in_lower,
error_mid_inward[0])
        mid_error_in_upper = np.append(mid_error_in_upper,
error_mid_inward[1])

        strong_error_in_lower = np.append(strong_error_in_lower
, error_strong_inward[0])
        strong_error_in_upper = np.append(strong_error_in_upper
, error_strong_inward[1])

        weak_error_out_lower = np.append(weak_error_out_lower,
error_weak_outward[0])
        weak_error_out_upper = np.append(weak_error_out_upper,
error_weak_outward[1])

        mid_error_out_lower = np.append(mid_error_out_lower,
error_mid_outward[0])
        mid_error_out_upper = np.append(mid_error_out_upper,
```

```
error_mid_outward[1])

    strong_error_out_lower = np.append(
strong_error_out_lower, error_strong_outward[0])
    strong_error_out_upper = np.append(
strong_error_out_upper, error_strong_outward[1])



average_velocity_inward = [average_distance_03_inward,
average_distance_04_inward, average_distance_05_inward,
average_distance_06_inward, average_distance_07_inward,
average_distance_08_inward]
average_velocity_outward = [average_distance_03_outward,
average_distance_04_outward, average_distance_05_outward,
average_distance_06_outward ,average_distance_07_outward ,
average_distance_08_outward]

plt.errorbar(average_velocity_inward, (inward_list_weak /
total_inward) * 100, label = 'Weak', fmt='o-',yerr=(
weak_error_in_lower, weak_error_in_upper))
plt.errorbar(average_velocity_inward, (inward_list_mid /
total_inward) * 100, label = 'Mid', fmt='o-', yerr=(
mid_error_in_lower, mid_error_in_upper))
plt.errorbar(average_velocity_inward, (inward_list_strong /
 total_inward) * 100, label = 'Strong', fmt='o-', yerr=(
strong_error_in_lower, strong_error_in_upper))
plt.vlines(50,ymin=0, ymax=100, linestyles='--', colors = '
k')
plt.title('Perihelion side')
plt.xlabel('Impact velocity (km/s)')
plt.ylabel('Amplitude ratio of the total  (%)')
plt.grid()
plt.legend()

plt.show()

plt.errorbar(average_velocity_outward, (outward_list_weak /
 total_outward) * 100, label = 'Weak', fmt='o-', yerr=(
weak_error_out_lower, weak_error_out_upper))
plt.errorbar(average_velocity_outward, (outward_list_mid /
total_outward)* 100, label = 'Mid', fmt='o-', yerr=(
mid_error_out_lower, mid_error_out_upper))
plt.errorbar(average_velocity_outward, (outward_list_strong
/ total_outward) * 100, label = 'Strong', fmt='o-', yerr=(
strong_error_out_lower, strong_error_out_upper))
plt.vlines(50, ymin=0, ymax=100, linestyles='--', colors='k
')
plt.title('Aphelion side')
plt.xlabel('Impact velocity (km/s)')
plt.ylabel('Amplitude ratio of the total (%)')
plt.grid()
plt.legend()

```

```
    plt.show()

new_velocity()

#Average distances  for the inbound and outbound trajectories
average_distance_02_outbound = np.mean(((SolO_radius[924 :
    928])))
average_distance_02_inbound = np.mean(((SolO_radius[924 : 928])
    ))

average_distance_03_outbound = np.mean(np.concatenate((
    SolO_radius[599:609], SolO_radius[794:803], SolO_radius
    [929:938])))
average_distance_03_inbound = np.mean(np.concatenate((
    SolO_radius[590:599], SolO_radius[778:787], SolO_radius
    [914:923])))

average_distance_04_outbound = np.mean(np.concatenate((
    SolO_radius[206:213], SolO_radius[610:617], SolO_radius
    [804:810], SolO_radius[939:946])))
average_distance_04_inbound = np.mean(np.concatenate((
    SolO_radius[197:205], SolO_radius[582:589], SolO_radius
    [771:777], SolO_radius[909:913])))

average_distance_05_outbound = np.mean(np.concatenate((
    SolO_radius[0:16], SolO_radius[214:230], SolO_radius
    [418:421], SolO_radius[618:623], SolO_radius[811:814],
    SolO_radius[947:948])))
average_distance_05_inbound = np.mean(np.concatenate((
    SolO_radius[185:196], SolO_radius[408:417], SolO_radius
    [574:581], SolO_radius[763:770], SolO_radius[901:908])))

average_distance_06_outbound = np.mean(np.concatenate((
    SolO_radius[17:40], SolO_radius[231:245], SolO_radius
    [422:440], SolO_radius[624:628], SolO_radius[815:823],
    SolO_radius[949:953])))
average_distance_06_inbound = np.mean(np.concatenate((
    SolO_radius[171:184], SolO_radius[389:407], SolO_radius
    [567:573], SolO_radius[754:762], SolO_radius[893:900])))

average_distance_07_outbound = np.mean(np.concatenate((
    SolO_radius[41:44], SolO_radius[246:260], SolO_radius
    [441:454], SolO_radius[629:639], SolO_radius[824:831],
    SolO_radius[954:963])))
average_distance_07_inbound = np.mean(np.concatenate((
    SolO_radius[155:170], SolO_radius[373:388], SolO_radius
    [557:566], SolO_radius[743:753], SolO_radius[887:892])))

average_distance_08_outbound = np.mean(np.concatenate((
    SolO_radius[45:54], SolO_radius[261:282], SolO_radius
    [455:469], SolO_radius[640:653], SolO_radius[832:845],
    SolO_radius[964:977])))
average_distance_08_inbound = np.mean(np.concatenate((
    SolO_radius[136:154], SolO_radius[351:372], SolO_radius
```

```
    [544:556], SolO_radius[730:742], SolO_radius[878:886]]))

average_distance_09_outbound = np.mean(np.concatenate((SolO_radius[55:92], SolO_radius[283:317], SolO_radius[470:504], SolO_radius[654:691], SolO_radius[846:860]]))
average_distance_09_inbound = np.mean(np.concatenate((SolO_radius[93:135], SolO_radius[318:350], SolO_radius[505:543], SolO_radius[692:729], SolO_radius[861:877]]))



outbound_weak_02 =  sum(weaks[924 : 929])
inbound_weak_02 = sum(weaks[788:794])

outbound_mids_02 =  sum(mids[924 : 929])
inbound_mids_02 = sum(mids[788:794])

outbound_strongs_02 = sum(strongs[924 : 929])
inbound_strongs_02 = sum(strongs[788:794])


outbound_weak_03 = sum(weaks[599:610]) + sum(weaks[794:804]) + sum(weaks[929:939])
inbound_weak_03 = sum(weaks[590:600]) + sum(weaks[778:788]) + sum(weaks[914:924])

outbound_mids_03 = sum(mids[599:610]) + sum(mids[794:804]) + sum(mids[929:939])
inbound_mids_03 = sum(mids[590:600]) + sum(mids[778:788]) + sum(mids[914:924])

inbound_strongs_03 = sum(strongs[590:600]) + sum(strongs[778:788]) + sum(strongs[914:924])
outbound_strongs_03 = sum(strongs[599:610]) + sum(strongs[794:804]) + sum(strongs[929:939])

outbound_weak_04 = sum(weaks[206:214]) + sum(weaks[610:618]) + sum(weaks[804:811]) + sum(weaks[939:947])
inbound_weak_04 = sum(weaks[197:206]) + sum(weaks[582:590]) + sum(weaks[771:778]) + sum(weaks[909:914])

outbound_mids_04 = sum(mids[206:214]) + sum(mids[610:618]) + sum(mids[804:811]) + sum(mids[939:947])
inbound_mids_04 = sum(mids[197:206]) + sum(mids[582:590]) + sum(mids[771:778]) + sum(mids[909:914])

outbound_strongs_04 = sum(strongs[206:214]) + sum(strongs[610:618]) + sum(strongs[804:811]) + sum(strongs[939:947])
inbound_strongs_04 = sum(strongs[197:206]) + sum(strongs[582:590]) + sum(strongs[771:778]) + sum(strongs[909:914])

outbound_weak_05 = sum(weaks[0:17]) + sum(weaks[214:231]) + sum(weaks[418:422]) + sum(weaks[618:624]) + sum(weaks[811:815]) + sum(weaks[947:949])
```

```
outbound_mids_05 = sum(mids[0:17]) + sum(mids[214:231]) + sum(mids[418:422]) + sum(mids[618:624]) + sum(mids[811:815]) + sum(mids[947:949])
outbound_strongs_05 = sum(strongs[0:17]) + sum(strongs[214:231]) + sum(strongs[418:422]) + sum(strongs[618:624]) + sum(strongs[811:815]) + sum(strongs[947:949])


inbound_weak_05 = sum(weaks[185:197]) + sum(weaks[408:418]) + sum(weaks[574:582]) + sum(weaks[763:771]) + sum(weaks[901:909])
inbound_mids_05 = sum(mids[185:197]) + sum(mids[408:418]) + sum(mids[574:582]) + sum(mids[763:771]) + sum(mids[901:909])
inbound_strongs_05 = sum(strongs[185:197]) + sum(strongs[408:418]) + sum(strongs[574:582]) + sum(strongs[763:771]) + sum(strongs[901:909])


outbound_weak_06 = sum(weaks[17:41]) + sum(weaks[231:246]) + sum(weaks[422:441]) + sum(weaks[624:629]) + sum(weaks[815:824]) + sum(weaks[949:954])
outbound_mids_06 = sum(mids[17:41]) + sum(mids[231:246]) + sum(mids[422:441]) + sum(mids[624:629]) + sum(mids[815:824]) + sum(mids[949:954])
outbound_strongs_06 = sum(strongs[17:41]) + sum(strongs[231:246]) + sum(strongs[422:441]) + sum(strongs[624:629]) + sum(strongs[815:824]) + sum(strongs[949:954])


inbound_weak_06 = sum(weaks[171:185]) + sum(weaks[389:408]) + sum(weaks[567:574]) + sum(weaks[754:763]) + sum(weaks[893:901])
inbound_mids_06 = sum(mids[171:185]) + sum(mids[389:408]) + sum(mids[567:574]) + sum(mids[754:763]) + sum(mids[893:901])
inbound_strongs_06 = sum(strongs[171:185]) + sum(strongs[389:408]) + sum(strongs[567:574]) + sum(strongs[754:763]) + sum(strongs[893:901])


outbound_weak_07 = sum(weaks[41:45]) + sum(weaks[246:261]) + sum(weaks[441:455]) + sum(weaks[629:640]) + sum(weaks[824:832]) + sum(weaks[954:964])
outbound_mids_07 = sum(mids[41:45]) + sum(mids[246:261]) + sum(mids[441:455]) + sum(mids[629:640]) + sum(mids[824:832]) + sum(mids[954:964])
outbound_strongs_07 = sum(strongs[41:45]) + sum(strongs[246:261]) + sum(strongs[441:455]) + sum(strongs[629:640]) + sum(strongs[824:832]) + sum(strongs[954:964])


inbound_weak_07 = sum(weaks[155:171]) + sum(weaks[373:389]) + sum(weaks[557:567]) + sum(weaks[743:754]) + sum(weaks[887:893])
inbound_mids_07 = sum(mids[155:171]) + sum(mids[373:389]) + sum
```

```
(mids[557:567]) + sum(mids[743:754]) + sum(mids[887:893])
inbound_strongs_07 = sum(strongs[155:171]) + sum(strongs[373:389]) + sum(strongs[557:567]) + sum(strongs[743:754]) + sum(strongs[887:893])


outbound_weak_08 = sum(weaks[45:55]) + sum(weaks[261:283]) + sum(weaks[455:470]) + sum(weaks[640:654]) + sum(weaks[832:846]) + sum(weaks[964:979])
outbound_mids_08 = sum(mids[45:55]) + sum(mids[261:283]) + sum(mids[455:470]) + sum(mids[640:654]) + sum(mids[832:846]) + sum(mids[964:979])
outbound_strongs_08 = sum(strongs[45:55]) + sum(strongs[261:283]) + sum(strongs[455:470]) + sum(strongs[640:654]) + sum(strongs[832:846]) + sum(strongs[964:979])


inbound_weak_08 = sum(weaks[136:155]) + sum(weaks[351:373]) + sum(weaks[544:557]) + sum(weaks[730:743]) + sum(weaks[878:887])
inbound_mids_08 = sum(mids[136:155]) + sum(mids[351:373]) + sum(mids[544:557]) + sum(mids[730:743]) + sum(mids[878:887])
inbound_strongs_08 = sum(strongs[136:155]) + sum(strongs[351:373]) + sum(strongs[544:557]) + sum(strongs[730:743]) + sum(strongs[878:887])

outbound_weak_09 = sum(weaks[55:93]) + sum(weaks[283:318]) + sum(weaks[470:505]) + sum(weaks[654:692]) + sum(weaks[846:861])
outbound_mids_09 = sum(mids[55:93]) + sum(mids[283:318]) + sum(mids[470:505]) + sum(mids[654:692]) + sum(mids[846:861])
outbound_strongs_09 = sum(strongs[55:93]) + sum(strongs[283:318]) + sum(strongs[470:505]) + sum(strongs[654:692]) + sum(strongs[846:861])


inbound_weak_09 = sum(weaks[93:136]) + sum(weaks[318:351]) + sum(weaks[505:544]) + sum(weaks[692:730]) + sum(weaks[861:878])
inbound_mids_09 = sum(mids[93:136]) + sum(mids[318:351]) + sum(mids[505:544]) + sum(mids[692:730]) + sum(mids[861:878])
inbound_strongs_09 = sum(strongs[93:136]) + sum(strongs[318:351]) + sum(strongs[505:544]) + sum(strongs[692:730]) + sum(strongs[861:878])


outbound_total_02 = outbound_weak_02 + outbound_mids_02 + outbound_strongs_02
outbound_total_03 = outbound_weak_03 + outbound_mids_03 + outbound_strongs_03
outbound_total_04 = outbound_weak_04 + outbound_mids_04 + outbound_strongs_04
outbound_total_05 = outbound_weak_05 + outbound_mids_05 + outbound_strongs_05
```

```
outbound_total_06 = outbound_weak_06 + outbound_mids_06 + outbound_strongs_06
outbound_total_07 = outbound_weak_07 + outbound_mids_07 + outbound_strongs_07
outbound_total_08 = outbound_weak_08 + outbound_mids_08 + outbound_strongs_08
outbound_total_09 = outbound_weak_09 + outbound_mids_09 + outbound_strongs_09

inbound_total_02 = inbound_weak_02 + inbound_mids_02 + inbound_strongs_02
inbound_total_03 = inbound_weak_03 + inbound_mids_03 + inbound_strongs_03
inbound_total_04 = inbound_weak_04 + inbound_mids_04 + inbound_strongs_04
inbound_total_05 = inbound_weak_05 + inbound_mids_05 + inbound_strongs_05
inbound_total_06 = inbound_weak_06 + inbound_mids_06 + inbound_strongs_06
inbound_total_07 = inbound_weak_07 + inbound_mids_07 + inbound_strongs_07
inbound_total_08 = inbound_weak_08 + inbound_mids_08 + inbound_strongs_08
inbound_total_09 = inbound_weak_09 + inbound_mids_09 + inbound_strongs_09

outbound_total = [outbound_total_02, outbound_total_03, outbound_total_04,outbound_total_05,outbound_total_06, outbound_total_07,outbound_total_08,outbound_total_09]
inbound_total = [inbound_total_02, inbound_total_03, inbound_total_04, inbound_total_05, inbound_total_06, inbound_total_07,inbound_total_08, inbound_total_09]


weak_outbound_list = np.array([outbound_weak_02 / outbound_total_02, outbound_weak_03 / outbound_total_03, outbound_weak_04 / outbound_total_04, outbound_weak_05 / outbound_total_05, outbound_weak_06 / outbound_total_06, outbound_weak_07 / outbound_total_07,
                      outbound_weak_08 / outbound_total_08, outbound_weak_09 / outbound_total_09])

mid_outbound_list = np.array([outbound_mids_02 / outbound_total_02, outbound_mids_03 / outbound_total_03, outbound_mids_04 / outbound_total_04, outbound_mids_05 / outbound_total_05, outbound_mids_06 / outbound_total_06, outbound_mids_07 / outbound_total_07,
                      outbound_mids_08 / outbound_total_08, outbound_mids_09 / outbound_total_09])

strong_outbound_list = np.array([outbound_strongs_02 / outbound_total_02, outbound_strongs_03 / outbound_total_03, outbound_strongs_04 / outbound_total_04, outbound_strongs_05 / outbound_total_05,
```

```
    outbound_strongs_06 / outbound_total_06,outbound_strongs_07 / outbound_total_07,
                      outbound_strongs_08 / outbound_total_08, outbound_strongs_09 / outbound_total_09])

weak_inbound_list = np.array([inbound_weak_02 / inbound_total_02, inbound_weak_03 / inbound_total_03, inbound_weak_04 / inbound_total_04, inbound_weak_05 / inbound_total_05, inbound_weak_06 / inbound_total_06,
                      inbound_weak_07 / inbound_total_07, inbound_weak_08 / inbound_total_08, inbound_weak_09 / inbound_total_09])


mid_inbound_list = np.array([inbound_mids_02 / inbound_total_02, inbound_mids_03 / inbound_total_03, inbound_mids_04 / inbound_total_04, inbound_mids_05 / inbound_total_05, inbound_mids_06 / inbound_total_06,
                      inbound_mids_07 / inbound_total_07, inbound_mids_08 / inbound_total_08, inbound_mids_09 / inbound_total_09])

strong_inbound_list = np.array([inbound_strongs_02 / inbound_total_02, inbound_strongs_03 / inbound_total_03, inbound_strongs_04 / inbound_total_04, inbound_strongs_05 / inbound_total_05, inbound_strongs_06 / inbound_total_06,
                      inbound_strongs_07 / inbound_total_07, inbound_strongs_08 / inbound_total_08, inbound_strongs_09 / inbound_total_09])

weak_outbound_list2 = [outbound_weak_02, outbound_weak_03, outbound_weak_04, outbound_weak_05, outbound_weak_06, outbound_weak_07, outbound_weak_08, outbound_weak_09]
mid_outbound_list2 = [outbound_mids_02, outbound_mids_03, outbound_mids_04, outbound_mids_05, outbound_mids_06, outbound_mids_07, outbound_mids_08, outbound_mids_09]

strong_outbound_list2 = [outbound_strongs_02, outbound_strongs_03, outbound_strongs_04, outbound_strongs_05, outbound_strongs_06, outbound_strongs_07, outbound_strongs_08, outbound_strongs_09]

weak_inbound_list2 = [inbound_weak_02, inbound_weak_03, inbound_weak_04, inbound_weak_05, inbound_weak_06, inbound_weak_07, inbound_weak_08, inbound_weak_09]

mid_inbound_list2 = [inbound_mids_02, inbound_mids_03, inbound_mids_04, inbound_mids_05, inbound_mids_06, inbound_mids_07, inbound_mids_08, inbound_mids_09]

strong_inbound_list2 = [inbound_strongs_02, inbound_strongs_03, inbound_strongs_04, inbound_strongs_05, inbound_strongs_06, inbound_strongs_07, inbound_strongs_08,
```

```
    inbound_strongs_09]


average_distance_inbound = np.array([
    average_distance_02_inbound, average_distance_03_inbound,
    average_distance_04_inbound,average_distance_05_inbound,
    average_distance_06_inbound,average_distance_07_inbound,
    average_distance_08_inbound,average_distance_09_inbound])
average_distance_outbound = np.array([
    average_distance_02_outbound, average_distance_03_outbound,
     average_distance_04_outbound, average_distance_05_outbound
    , average_distance_06_outbound,
    average_distance_07_outbound, average_distance_08_outbound,
     average_distance_09_outbound])


distance = [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

mean_dist = np.zeros(0)


for i in range(2,10):
    print(i)
    mask = ((SolO_radius > i / 10) * (SolO_radius < (i + 1) /
    10))

    mean_dist = np.append(mean_dist, np.mean(SolO_radius[mask])
    )




error_weak_out_lower = np.zeros(0)
error_weak_out_upper = np.zeros(0)

error_mid_out_lower = np.zeros(0)
error_mid_out_upper = np.zeros(0)

error_strong_out_lower = np.zeros(0)
error_strong_out_upper = np.zeros(0)

error_weak_in_lower= np.zeros(0)
error_weak_in_upper= np.zeros(0)

error_mid_in_lower = np.zeros(0)
error_mid_in_upper = np.zeros(0)

error_strong_in_lower = np.zeros(0)
error_strong_in_upper = np.zeros(0)

for i in range(len(mid_outbound_list[1:])):
    error_weak_outbound = calculate_error(weak_outbound_list2[i
    ], outbound_total[i])
    error_mid_outbound = calculate_error(mid_outbound_list2[i],
```

```
        outbound_total[i])
    error_strong_outbound = calculate_error(strong_outbound_list2[i], outbound_total[i])

    error_weak_inbound = calculate_error(weak_inbound_list2[i], inbound_total[i])
    error_mid_inbound = calculate_error(mid_inbound_list2[i], inbound_total[i])
    error_strong_inbound = calculate_error(strong_inbound_list2[i], inbound_total[i])

    error_weak_out_lower = np.append(error_weak_out_lower, error_weak_outbound[0])
    error_weak_out_upper = np.append(error_weak_out_upper, error_weak_outbound[1])

    error_mid_out_lower = np.append(error_mid_out_lower, error_mid_outbound[0])
    error_mid_out_upper = np.append(error_mid_out_upper, error_mid_outbound[1])

    error_strong_out_lower = np.append(error_strong_out_lower, error_strong_outbound[0])
    error_strong_out_upper = np.append(error_strong_out_upper, error_strong_outbound[1])


    error_weak_in_lower = np.append(error_weak_in_lower, error_weak_inbound[0])
    error_weak_in_upper = np.append(error_weak_in_upper, error_weak_inbound[1])

    error_mid_in_lower = np.append(error_mid_in_lower, error_mid_inbound[0])
    error_mid_in_upper = np.append(error_mid_in_upper, error_mid_inbound[1])

    error_strong_in_lower = np.append(error_strong_in_lower, error_strong_inbound[0])
    error_strong_in_upper = np.append(error_strong_in_upper, error_strong_inbound[1])


#Plots the weak, mid and strong for inbound and outbound trajectories
plt.errorbar(average_distance_outbound[1:],weak_outbound_list[1:] * 100 , label='Outbound weak', capsize = 3, fmt = '-', color ='blue')
plt.fill_between(average_distance_outbound[1:], weak_outbound_list[1:] * 100 - error_weak_out_lower, weak_outbound_list[1:] * 100 + error_weak_out_upper, color = 'midnightblue',  alpha = 0.4, lw = 0)

plt.errorbar(average_distance_outbound[1:],mid_outbound_list
```

```
[1:] * 100, label = 'Outbound mid', capsize = 3 , fmt = '-', color ='orange',)
plt.fill_between(average_distance_outbound[1:], mid_outbound_list[1:] * 100 - error_mid_out_lower, mid_outbound_list[1:] * 100 + error_mid_out_upper, color = 'gold',  alpha = 0.4, lw = 0)

plt.errorbar(average_distance_outbound[1:],strong_outbound_list[1:] * 100 ,  label = 'Outbound strong', capsize = 3, fmt = '-', color = 'green')
plt.fill_between(average_distance_outbound[1:], np.array(strong_outbound_list)[1:] * 100 - error_strong_out_lower, np.array(strong_outbound_list)[1:] * 100 + error_strong_out_upper, color = 'forestgreen',  alpha = 0.5, lw= 0)

plt.errorbar(average_distance_inbound[1:],np.array(weak_inbound_list)[1:] * 100 ,label='Inbound weak', capsize = 3, fmt = '--', color = 'cyan')
plt.fill_between(average_distance_inbound[1:], np.array(weak_inbound_list)[1:] * 100 - error_weak_in_lower, np.array(weak_inbound_list)[1:] * 100 + error_weak_in_upper, color = 'darkslategrey',  alpha = 0.5, lw = 0)


plt.errorbar(average_distance_inbound[1:],np.array(mid_inbound_list)[1:] * 100, label = 'Inbound mid', capsize = 3,fmt = '--', color = 'maroon')
plt.fill_between(average_distance_inbound[1:], np.array(mid_inbound_list)[1:] * 100 - error_mid_in_lower, np.array(mid_inbound_list)[1:] * 100 + error_mid_in_upper, color = 'crimson', alpha = 0.5, lw = 0)


plt.errorbar(average_distance_inbound[1:],np.array(strong_inbound_list)[1:] * 100, label = 'Inbound strong', capsize = 3,fmt = '--', color='lawngreen')
plt.fill_between(average_distance_inbound[1:], np.array(strong_inbound_list)[1:] * 100 - error_strong_in_lower,np.array(strong_inbound_list)[1:] * 100 + error_strong_in_lower, color = 'seagreen', alpha = 0.5, lw = 0)


plt.xlabel('Distance from the Sun (AU)')
plt.ylabel('Ratio (%)')

plt.grid()
plt.legend(loc='center right', bbox_to_anchor = (1, 0.5))

plt.show()

print("inbound, weak", weak_inbound_list2)
print("inbound, mid", mid_inbound_list2)
```

```
print("inbound, strong", strong_inbound_list2)
print("inbound, total", inbound_total)

print("outbound, weak", weak_outbound_list2)
print("outbound, mid", mid_outbound_list2)
print("outbound, strong", strong_outbound_list2)
print("outbound, total", outbound_total)



weekly_weak = np.zeros(0)
weekly_mid = np.zeros(0)
weekly_strong = np.zeros(0)
weekly_distance = np.zeros(0)


#Finds the weak, mid and strong in weekly intervals
for i in range(0, len(weaks), 8):
    # Sum the elements in the current chunk (excluding the 7th element) and append to the weekly_sums list
    end_index = i +7
    weekly_sum = sum(weaks[i:end_index])
    weekly_d  = np.mean(SolO_radius[i:end_index])
    weekly_weak = np.append(weekly_weak,weekly_sum)
    weekly_mid = np.append(weekly_mid, sum(mids[i:end_index]))
    weekly_strong = np.append(weekly_strong, sum(strongs[i:end_index]))

    weekly_distance = np.append(weekly_distance, weekly_d)



def slope():
    """
    Finds the slope of the mass distribution
    :return:
    """
    R = weekly_strong/weekly_mid
    R_2 = sum(strongs) / sum(mids)
    Q_low = 20
    Q_high = 200
    #print(R)
    #print(R_2)
    a = (-np.log(Q_high) + np.log(Q_low) + np.log((R / (R+1)))) / (np.log(Q_high) - np.log(Q_low))

    exponent  = a
    #print(np.log(R/(R-1)))

    return exponent

exponent = slope()


```

```python
list_exponent = [[e,w ] for e, w in zip(exponent,
    weekly_distance) if np.isfinite(e)]




#plots the exponent points and a slope
fig, ax = plt.subplots()

exponent_list = np.zeros(0)
weekly_distance_list = np.zeros(0)

for item in list_exponent:
    ax.scatter(item[1],item[0], c = 'blue')
    exponent_list = np.append(exponent_list, item[0])
    weekly_distance_list = np.append(weekly_distance_list, item
    [1])

mean_a = np.mean(exponent_list)

percentile_5 = np.nanpercentile(exponent_list, 5)
percentile_95 = np.nanpercentile(exponent_list, 95)

coeff =np.polyfit(weekly_distance_list, exponent_list,1)

line = np.poly1d(coeff)
ax.plot(weekly_distance_list,line(weekly_distance_list), color
    = 'red')
ax.set_xlabel('Distance from Sun (AU)')
ax.set_ylabel('Exponent a')
plt.show()
```

Listing B.2: Code uses the cnn processed data files and plots the trajectories as well as calculates the slope value and plots it.