3D Mapping the stars in Gaia DR2

Eirik Bratli

July 2019

Part I
Report

1 Introduction

This is a product of my summer project with the CMB/CO group at the Institute of Theoretical Astrophysics (ITA), University of Oslo, where the main goal is to create 3D maps of the Gaia Data Release 2 (Gaia DR2) and make the CMB/CO group at ITA familiar with Gaia DR2. Gaia is an European satellite govern by ESA, its scientific goal is to map the Milky Way galaxy in 3D. In order to achieve this the satellite has obtained information of about 1 % of the stars in the galaxy (ESA). These data can be used to map the influence of the stars in the foreground signals in the mapping of the Cosmic Microwave Background radiation, and what parts of the foreground radiation comes from stars. The Gaia DR2 contain other kind information that can also be used to map dust. The parameters that are useful for our goals in the Gaia DR2 are listed in table 1.

We want to make a 3D model with several map layers. Each of the maps we create contain the stars in a distance range from distance a to distance b, i.e. 1 kpc to 2 kpc. The maps are created in Healpix coordinates and shown in Mollweide projection, where one map are the stars within a distance range. When using multiple distance ranges we get several map layers that are used to make a 3D map of the stars to see how the stars are distributed in space. In order to analyse the Gaia DR2 we developed two programs to formatting the data files from Gaia, one to create parameter files of the listed parameters in table 1 and one to program to make astrophysical parameters, like distance and Healpix coordinates, from the parameter files from the first program. The main program are used to analyse the parameter files and create the maps, either for all of the stars in one map or create several maps for stars in different distance ranges. There is a fourth program that contain help functions to the main programs.

This report are divided in two parts. First part contain the methodology and our results, section 2 and 3. The second part contain the documentation of the programs and user guide, section 4. All plots that are describing the results are found in my github, link below in section 4 Documentation. Read the documentation if you want to create maps yourself.

1.1 Gaia DR2

The Data Release 2 from Gaia contain almost 1.7 billion objects in total, where there is positional data for all of the objects. While the distance are measured for about 1.3 billion objects. The instruments in the satellite have three filter bands, green G, blue G_{BP} and red G_{RP} to measure the photon flux, this is then converted to magnitude. For the mean photon magnitude of the G_{BP} and G_{RP} almost 1.4 billion objects have data, while in the green filter all objects are listed. There are many more parameters in DR2, but they are not interesting for creating 3D map of stellar positions. The objects span a wide range of distances, the furthest objects are at about 10^{12} pc away. The objects that are at distances far away compared to the size of the Milky way galaxy may be considered as galaxies and Quazars. In general we will here just map the objects up to 30 kpc, but the programs handle both larger and smaller distances well, and may be specified when running the program. The Gaia DR2 also

Table 1: List of the parameters used in this analysis of Gaia DR2. First column is the parameter with units, second column is the column name in the data files, third column is the number of objects of that parameter.

Parameter	[unit]	column name	Number of objects
Source ID		source_id	1 692 919 135
Right ascension	[deg]	ra	1 692 919 135
Declination	[deg]	dec	1 692 919 135
Parallax	[mas]	parallax	1 331 909 727
Error in parallax	[mas]	parallax_error	1 331 909 727
Mean Photon magnitude G	[mag]	phot_G_mean_mag	1 692 919 135
Mean Photon magnitude G_{BP}	[mag]	phot_BP_mean_mag	1 381 964 755
Mean Photon magnitude G_{RP}	[mag]	phot_RP_mean_mag	1 383 551 713

contains data for the Solar system and other types of objects that are not relevant for this analysis. These types of data are stored in different files than the stellar mapping data files.

In order to analyse the data the tools provided by Python are used. The data files from Gaia comes in 61 237 .csv files, each of about 10 MB size. To work with so many files containing many parameters are a mess, so the files are converted into parameter files of type Hierarchical Data Format 5 (HDF5). With one .h5 file per parameter.

2 Method

In order to solve the tasks of creating 3D map involves different steps. First we needed to get the relevant data from the Gaia DR2 files What we did when reading and writing the parameter files was to use 5 different servers and then run the code manually in parallel. The .csv files ends with a even number, so every file ending with a given number was run on one of the servers available at ITA, i.e. all filenames ending with 0 was run on server 1, all filenames ending with 2 was run on server 2, etc. Each of these processes looped over the second to last digit in the .csv filename to save memory. When this was finished the parameter files for each parameter was merged into one file. Then a distance file can be created from the parallax file, and stellar pixel coordinate files can be created from the position files, in Healpix coordinates. We are not using the uncertainty in the right ascension and declination. This is because the assumed effect is small. This assumtion is based on the much smaller uncertainty than the angular position, comparing milli arc-seconds to degrees. Also if including the positional angular error, the complexity of getting the pixel coordinates will increase by a lot.

In creating the stellar maps, we first create one map with all stars. Then we add the magnitudes to the stars. This is done by weighting the stars by their magnitude, as described below in section 2.2. Next is to see which N_{side} gives a good map, N_{side} goes as a power law of two, so $N_{side} = 2^p$ for $p > 0, p \in \mathcal{N}$. A good map is judged by the balance between the resolution and the computation time and memory usage. We want high resolution in order to see enough details, but high resolution means the computation time is also high. High resolution also use a lot of memory since the numbers of pixels in the map is given as $N_{pix} = 12 * N_{side}^2$, so high N_{side} means large N_{pix}.

The next step is to distribute the stars into distances for each map layer, these map layers can be either manually set or use a number of maps between a minimum distance and a maximum distance. The map creating function depend on one or more N_{side} to let the number of stars decide the N_{side} of the map layer. Each star has an uncertainty in the parallax angle, which can be converted to a distance. This uncertainty in distance are then included in the maps by assuming a Gaussian probability function and integrate over the part of the distance distribution of the star up to the edge of the map. For most of the stars all of the distance distribution are contained within the edges of the map layer. But for the stars which are not fully contained in the map layer, the part of the star not contained in the map are added to the neighbouring maps.

2.1 Parallax

Since the parallax is an angel, it needs to be converted to parsec for us to get a distance. The parallax to distance conversion is given by definition,

$$d = \frac{1}{p}, \tag{1}$$

where d is the distance in parsec and p is the parallax angle in arc-seconds. The uncertainty of the distance is not directly related, but is given as

$$\delta d = \delta\left(\frac{1}{p}\right) \approx \left|\frac{\delta p}{p^2}\right|, \tag{2}$$

where δd is the error in the distance and δp is the error in parallax angle. This parallax to distance conversion are quite accurate when the uncertainty is less than 10 % (van Leeuwen, 2008). Our data contain stars with larger errors than 10%, so for these stars we assume this conversion is still valid. Parallax is not the best distance measure for distant objects, so for distant stars the uncertainty are large. The stars with unknown distances are set to 0 for easy handling. we choose zero since there is no stars at zero parsec except the sun.

2.2 Magnitude weighting the stars

Each pixel in the map has a count value of how many stars there are in each pixel, we obtain these star counts by using NumPy histogram function. The histogram function has a input argument, 'weights' which weights the pixel values according to the input weights. This means, when adding the magnitude to the stars we need to consider that the magnitude are tenth logarithmic to get a reasonable factor for each star. The stars are weighted by a factor

$$w = 10^{-M}. \tag{3}$$

Here w is the weight and M is the magnitude. The weights are then used on the pixel coordinates in creating the maps. For the stars with missing magnitude values, we use the zero point values from the Gaia DR2 documentation. The zero point values are defined as the measured flux of a object to be 1 photoelectron per second, and are used to generate the magnitudes (M. Riello, P. Montegriffo, et.al., 2018). The zero point values are consistently a lower value than the stellar magnitudes we DR2, where the DR2 magnitudes are defined as the sum om of the zero point magnitude and the instrumental measurement of magnitude.

2.3 Distance uncertainty

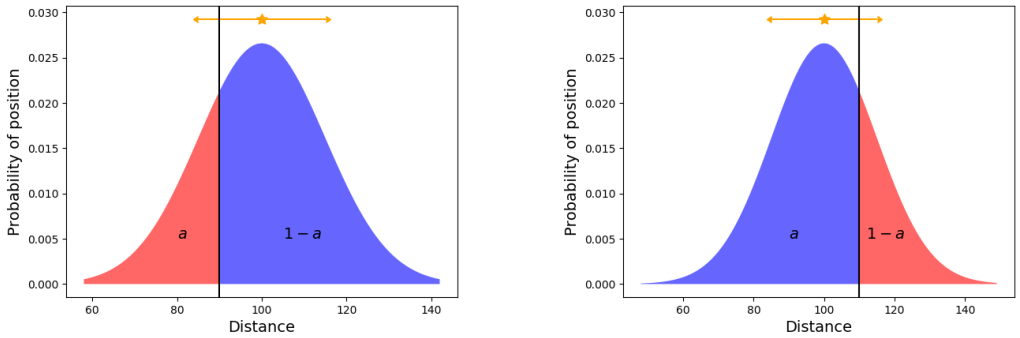
The uncertainty in the distance are assumed to be Gaussian distributed. To handle the uncertainty we distribute the distance with uncertainty given as the standard error. This means the stellar distances are given by a Gaussian distribution.

$$P(d) = \frac{1}{\sqrt{2\pi}\sigma_d} \exp\left\{-\frac{(d-\mu)^2}{2\sigma_d^2}\right\}, \tag{4}$$

where d is the distance variable to be integrated over μ is the mean distance given from the parallax and σ_d is the standard error. we include the uncertainty for stars with large uncertainty, $\sigma_d > 1\%$, and positioned close to a bin edge, 1% from the bin edge, to increase the precision of the maps. Most objects in the DR2 got low uncertainty and some objects have a very large uncertainty $\sigma_d > 100\%$. The later are usually objects far away and may be included in the bin at largest distance.

In order to get the contribution of an object on the different sides of the bin edges, we integrate up to the bin edge we look around. The integral is preformed as a sum, since we already have the distance distributed. The summation goes from small distance to larger distance. The contribution weight on the bin edges on bin i is then

$$W_l = w(1-a) \quad W_u = wa, \tag{5}$$



(a) Lower bin edge

(b) Upper bin edge

Figure 1: Illustration of the uncertainty of the stellar position. The orange star is the mean location of the star, with standard error as the horizontal bars, given from (1) and (2). The black vertical line is the bin edge. The blue area represent her the contribution to the current bin, while the red area are the contribution to the previous/ next bin.

where a is the sum up to the bin edge, w is the weight given by equation (3) and W_l, W_u is the weight contribution around the lower and upper bin edges. The array with weights are then updated with the new weights for the stars in the current bin. For the contribution to the next bin, red part in figure 1b, the $1 - a$ part of the stellar magnitude are added to the upcoming bin. Figure 1 show illustrations of the distance distribution of a star close to the upper bin edge and a star close to the lower bin edge. This is just an illustration of how the machinery works. Also the contribution to the previous bin for the stars close to the lower bin edge are neglected, since we are moving from close distance to larger distances. This is not ideal but a lot easier, and may be improved in the future.

3 Results

All the maps we have generated are not included, only a chosen selection of the maps that describes the main patterns in the distribution of the stars, since the figures take a lot of space in the report.

3.1 Testing Nside

By testing different values of N_{side} we found that $N_{side} \geq 512$ and $N_{side} \leq 4096$ are good. Figure 2 to 4 show the different maps of all stars with these N_{side}. From these figures we see that $N_{side} = 512$ and $N_{side} = 2048$ gives the best results. With $N_{side} = 1024$ the map is darker, because it is a bit random how many stars goes into each bin. We will still consider 1024 as a good N_{side} since it is still fast and give better resolution than $N_{side} = 512$. The problem with higher N_{side} is that the amount of time used to generate a map increase a lot, while low N_{side} give lower resolution.

3.2 All stars with magnitude included

By using $N_{side} = 512$ of the above result and add intensity to that map we get figure 5a, 5b, 5c and 5d. Here we see that there are some minor differences. Since only the green filter covers all the stars, we would consider using either only the green filter or the mean of the filters.

3.3 Distance distribution

By making a histogram of the distances to all of the stars in DR2 we get figure 6a, in figure 6b the distribution for stars up to 10 kpc are shown, here we clearly see a peak at about 1 kpc, then the star distribution decrease with a long tail. The star distribution reach maximum at about 1 kpc and then fall off with increasing distance. Figure 6 show that most star are located at 0.1 kpc to 100 kpc, with a

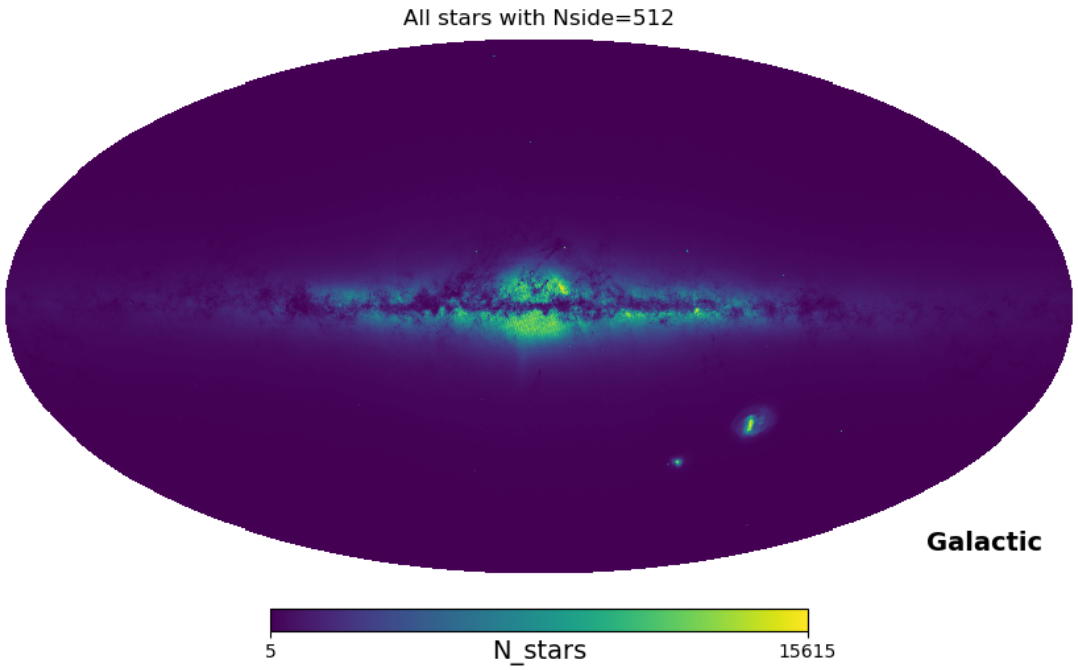


Figure 2: Full sky map for $N_{side} = 512$.

peak at about 1 kpc. The high amount of objects up to 10^7 pc comes from the neighbouring galaxies. we also see that there are not so many objects closer than 100 pc.

When comparing distance with percentile between the distance and the uncertainty in figure 7, we see that at low distances, $d < 10$ kpc the uncertainty are small in general, while for larger distances the uncertainty increase. For the largest distances the uncertainty are very large.

3.4 Main results, distance mapping:

The maps of the main results are made with $N_{side} = 1024$ to get good resolution, but still not too time consuming. Using a list of N_{side}, the change in N_{side} gives too much difference in the maps to be descriptive enough. When doing the settings of the main results, we choose to divide the maps into three categories as described in the next three paragraphs. For our close neighbourhood, 0 pc to 1000 pc we wanted smaller bins to see better our neighbourhood than is needed for greater distances. Then we want to see the whole galaxy which is in distances, 0 kpc to 30 kpc. And one part for distant objects of distances larger than 30 kpc. In these results we are including both the intensity of the star and the uncertainty of the distance to the stars. The maps do not include stars with unknown distance.

Closing in on our neighbourhood, from 0 pc to 1000 pc, we get the maps shown in figure 8. Here we see that the stars closest to us are more evenly distributed on the sky, while moving away the galactic disc become more apparent.

We get the following figures for distances 0 pc to 30 kpc, 9, 10 and 12, this covers most parts of the Milky way galaxy. Comparing with the close region maps, there are a lot more stars in each map, so they get more dense. Here the galaxy disc are bright at distances over 1 kpc and up to about 15 kpc. For the close figure 9a, the star are contained within the galactic disc. While for the stars far away the galactic disc become less apparent, but we still see it clearly. One thing to notice is that the Magellianic Clouds are visible at distances they are not part of.

Looking at distances outside the Milky way, from 30 kpc to 1000 kpc we see in figures 12a to 12c we see that there are fewer and fewer objects appearing ther further out we go. In the last map, figure 12d we see all the objects with distances larger than 1000 kpc. In these figures there are a lot of

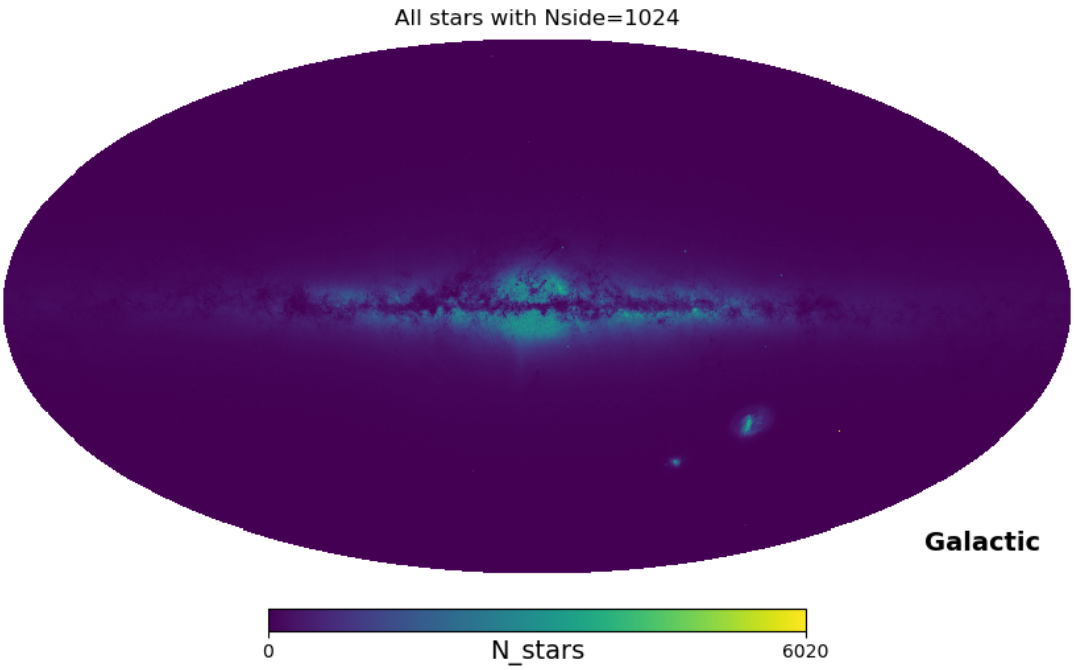


Figure 3: Full sky map for $N_{side} = 1024$.

uncertainties in the distance to the objects, i.e. that the Magellanic Clouds are visible at distance up to some 1000 kpc.

For some reason the Milky way are still apparent at these large distances. This may come form uncertainties in the distant parts of the galaxy. The further away the stars are the larger their uncertainties are, just as the appearance of the Magellanic Clouds in the closer regions and furthest away regions. This demonstrates that the error handling we have used is not good for large errors and that parallax precision is limited to small distances, up to about 10 kpc. At small distances there are vanishing few stars with percentile over 1%. This is supported by figure 7 where we see the errors increase with increasing distance.

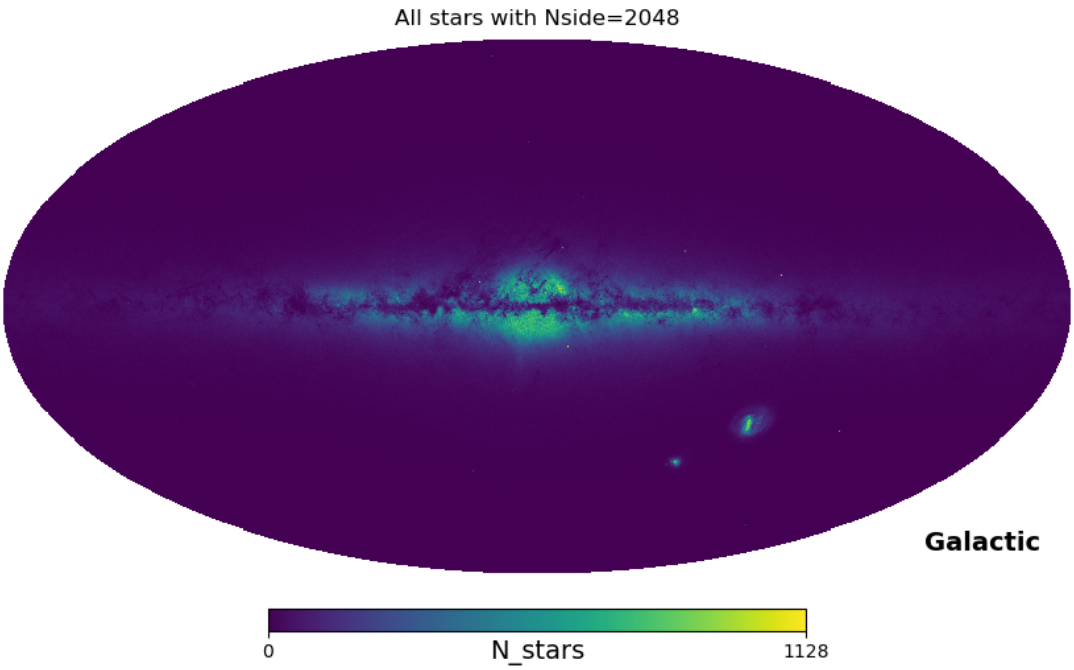
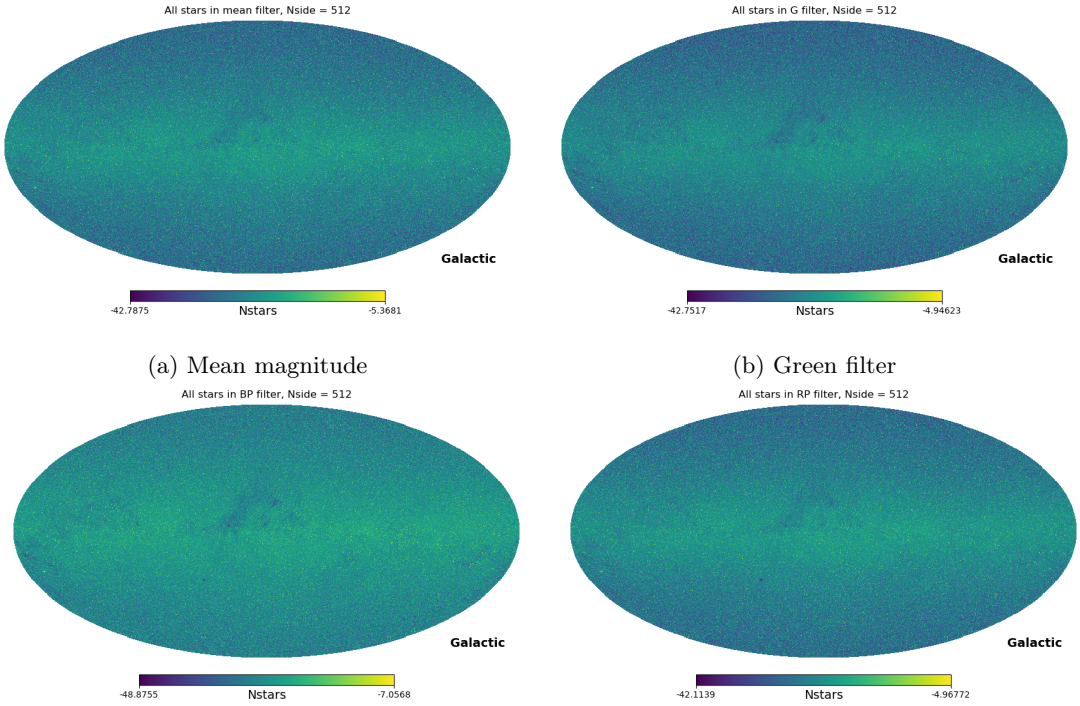


Figure 4: Full sky map for $N_{side} = 2048$.

Part II



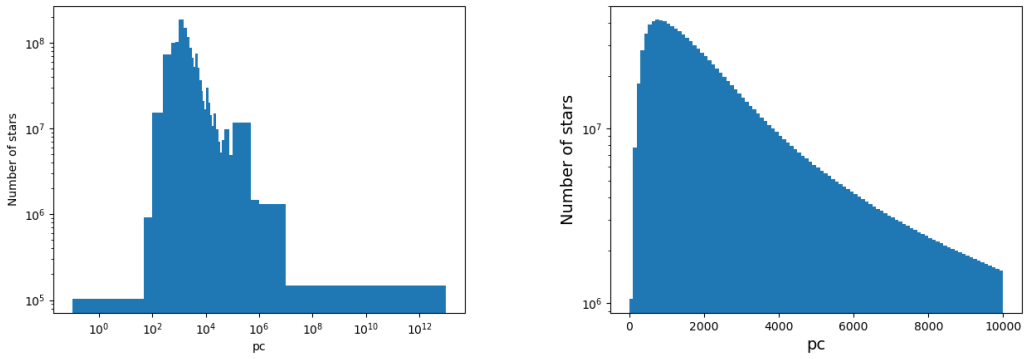
(a) Mean magnitude

(b) Green filter

(c) Blue filter

(d) Red filter

Figure 5: Full sky map using the three filters, green, blue, red, and the mean magnitude between the filters. The stars are weighted according to equaiton (3). All maps are made with $N_{side} = 512$.



(a) All stars.

(b) Stars up to 10 kpc.

Figure 6: Histograms of the stellar distribution with respect to distance. To the left we see the distribution of all stars. Here the bins are manually set in order to cover the whole distance range and to have enough bins at small distances. The x-axis are also logarithmic in order to see the full range of distances. To the right we see the stellar distribution up to 10 kpc, using linearly spaced bins. Here a linear x-axis show the stellar distribution in a good way.

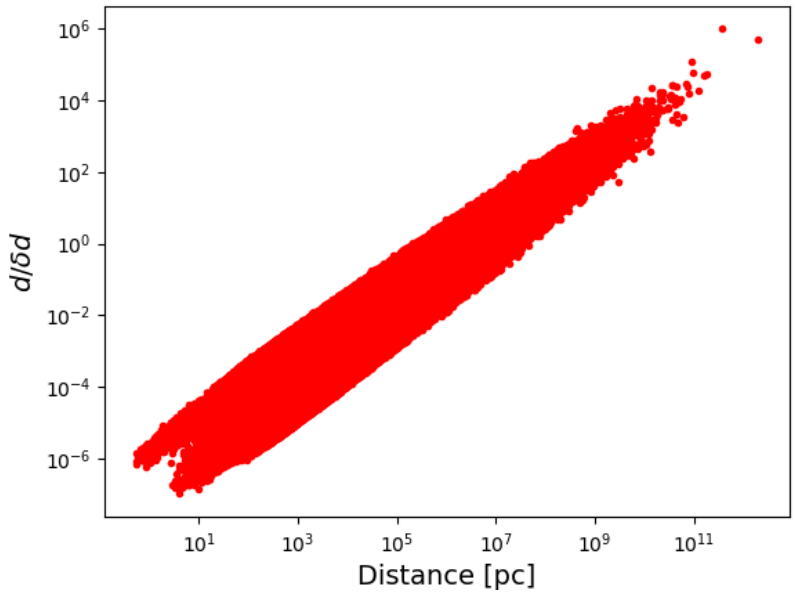
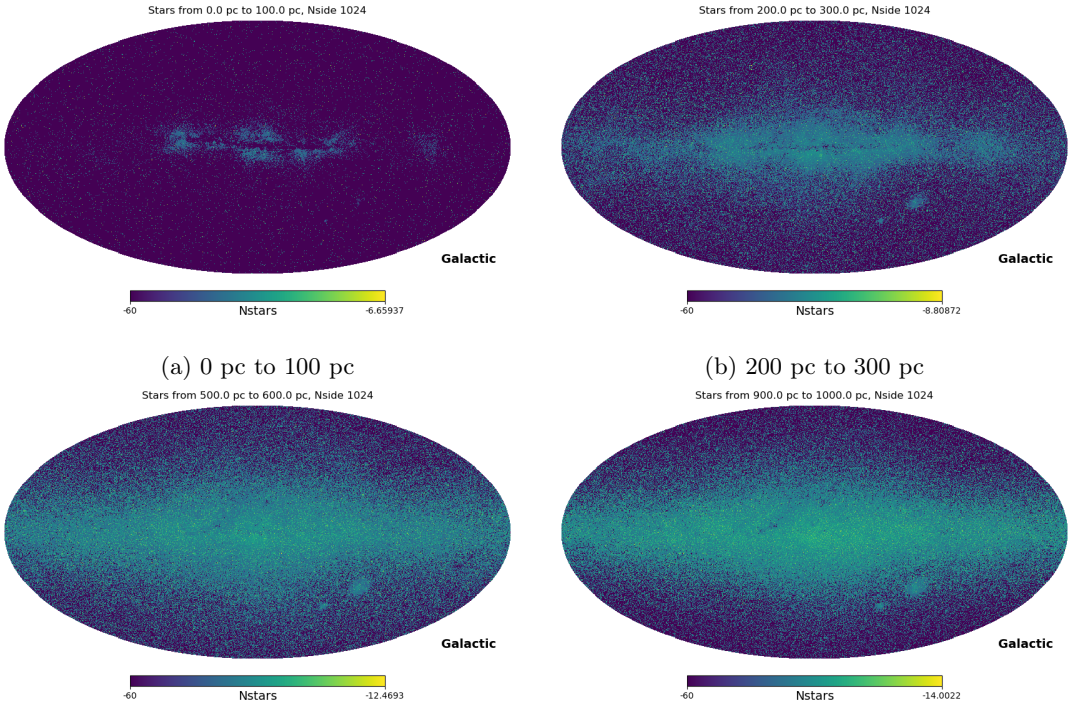


Figure 7: Comparison between the distance to the stars and the ratio between the error and the distance, $p = \delta d/d$. The plot are logarithmic scaled on both axis to see the effect on all distances. Have used every tenth star in this plot.



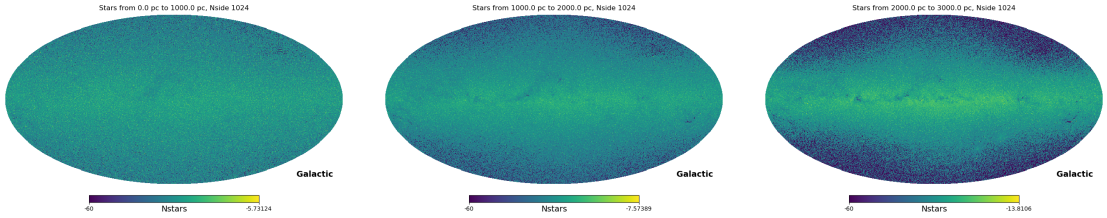
(a) 0 pc to 100 pc

(b) 200 pc to 300 pc

(c) 500 pc to 600 pc

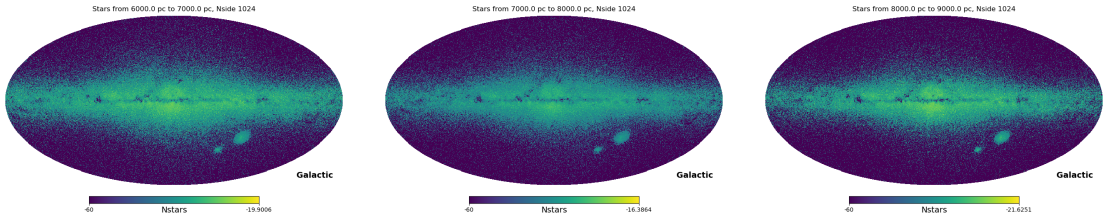
(d) 900 pc to 1000 pc

Figure 8: Four maps to show how the stellar distribution are from 0 pc and up to 1000 pc. In the two upper maps the stars are quite evenly distributed in each map, except along the equator in the upper right. Together with the lower maps it is clear that within 1000 pc, we start ro get regions outside the galactic plane.



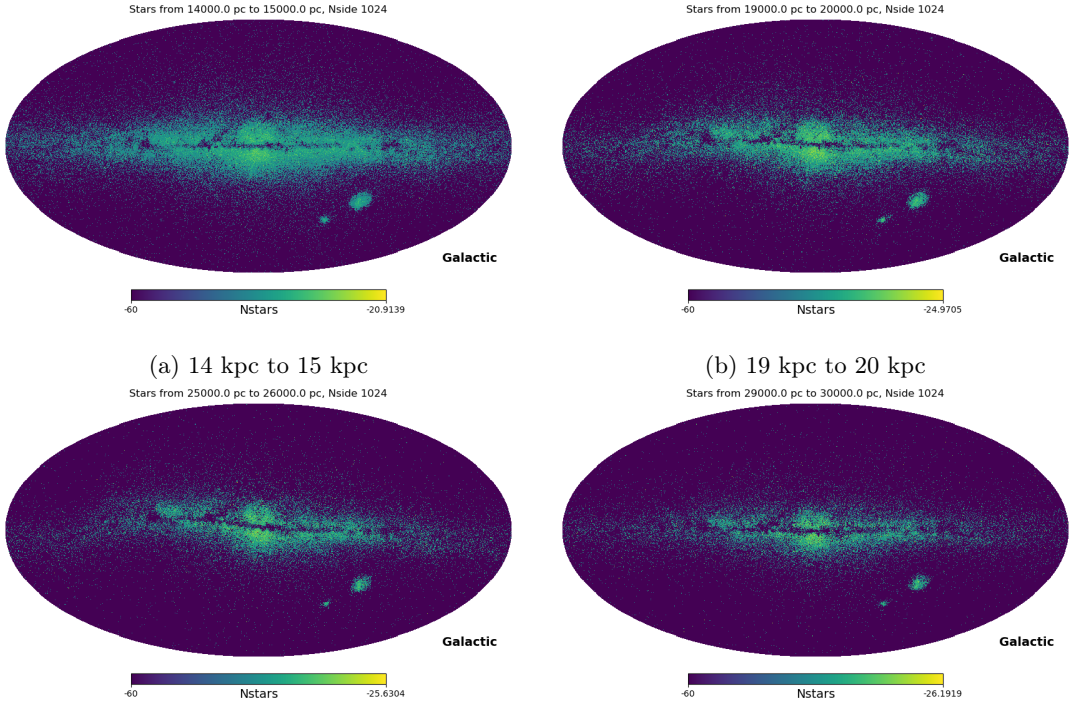
(a) 0 kpc to 1 kpc (b) 1 kpc to 2kpc (c) 2 kpc to 3 kpc

Figure 9: The three closest map layers of the Milky Way disc. There are 1 kpc between each map layer. we see clearly that we get more a disc as we get further out.



(a) 6 kpc to 7 kpc (b) 7 kpc to 8 kpc (c) 8 kpc to 9 kpc

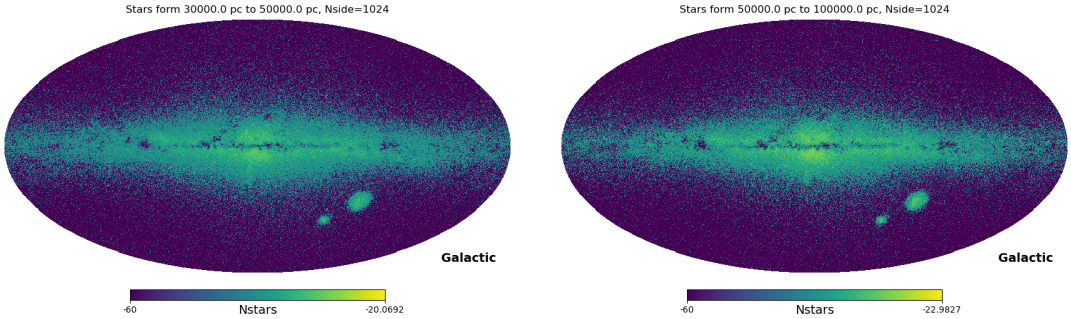
Figure 10: The Three map layers around the center of the Milky Way galaxy. There are 1 kpc between each map layer. we see the Magellianic Clouds to the right under the disc.



(a) 14 kpc to 15 kpc (b) 19 kpc to 20 kpc

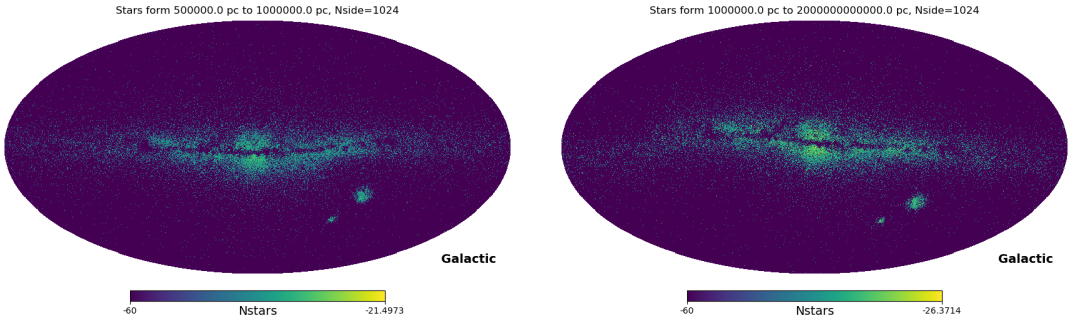
(c) 25 kpc to 26 kpc (d) 29 kpc to 30 kpc

Figure 11: For map layers of the Milky Way disc at greater distances to show how the maps evolve further out. we see at large distances that the Milky way disc shrink into the middle of the map. This is because in the direction away from the center, the distances are getting close to being outside the reaches of the Milky Way. In 11b and 11c we see the disc is not flat but tilted.



(a) 30 kpc to 50 kpc

(b) 50 kpc to 100 kpc



(c) 500 kpc to 1000 kpc

(d) 1 Mpc kpc to 2×10^6 Mpc

Figure 12: Four maps at distances outside the Milky Way galaxy, the two upper for distances up to 100 kpc, and the two lower for distances from 500 kpc and up to the most distant objects. In 12b map the Magellanic Cloud are located and we see them most bright there.

4 Documentation

We developed four python programs to solve the tasks of this project, the programs are described below. All of the programs have a highly use of the NumPy package and Healpy package. The later is the python version of Healpix, and contain the tools to create a Mollweide projection of the stars. For a few functions, Numba is included to speed up the calculation time. The package, h5py, are used to read and write the HDF files. Some of the programs are very time consuming like converting the .csv files to .h5 files, this should only needed to be done once and the .h5 files are available in the directory below. Also reading multiple parameter files take some time when later working with the data. In creating the maps, adding intensity and distance error to the stars increase the total run time, this is because of extra for loops and `if`-statements. When testing different resolutions, higher N_{side} use a long time to generate the maps. This is because of the huge amount of bins needed.

The .csv data files are stored in the directory,

`/mn/stornext/u3/eirikgje/data/gaia/cdn.gea.esac.esa.int/Gaia/gdr2/gaia_source/csv/`

And the programs including the .h5 files are stored in,

`/mn/stornext/d16/cmbco/pasiphae/eiribrat/Gaia/`

Running the programs: The way we call the reading .csv file functions in Datahandling.py, it is convenient to make these function calls inside a function. There are lines before the function calls for the reading of .csv file functions that change the working directory to the directory with the .csv files, and these function calls are done inside a for loop. So you need to call the calling function to read the .csv files. To show an example for reading the .csv filenames ending with the last digit, 2, and use the columns named 'ra', 'dec' and 'parallax', i.e. right ascension, declination and parallax:

```
# from Datahandling.py:
call_get_data(2, ['ra', 'dec', 'parallax'])

fileMerger('ra', 'ra', 'RightAscension', 'float')
fileMerger('dec', 'dec', 'Declination', 'float')
PixelCoord(1024, 'RightAscension.h5', 'Declination.h5')
```

This example show the calling of `call_get_data()` and the usage of the file handling functions to further work with the .h5 files, `fileMerger()` and `PixelCoordinates()`. For the other functions in the Datahandling.py program, there is no need for changing the directory. They are called in the normal way in python, by one line at the end of the program.

Running the functions in DistanceMapping.py are done in the normal fashion of python. The next example show two different ways of using the function `DistributeDistance()` in the DistanceMapping.py program, first for making ten maps from 0.1 pc to 1000 pc using the mean value of the filters to weight the star count in each pixel. Added also examples on how to call `testNside()` and `AllStars()`.

```
# from DistanceMapping.py:
testNside(5, 10)
AllStars(1024, 'mean')
# using Nmaps:
DistributeDistance(1024, Nmaps=10, Rmax=1000., Rmin=0.1, filter='mean')

# Using manually defined bins:
bins = [1,100,500,1000,2000,3000,4000,5000,7500,10000,20000,30000]
DistributeDistance(1024, Bin_seq=bins, cutoff=5, filter='G', h=True)
```

The second call are using manually defined bin edges, as specified in the example, with a cut off after 5 maps. This uses the green filter and also create histograms of the distance distribution of the stars within each bin. Note that the function calls used in this project are available for reuse, they are commented out at the bottom of the program script.
The programs can be found in my GitHub: `github.com/EirikBratli/Gaia3Dmapping`

4.1 Datahandling.py

The first program to handles the data files, like reading and writing files .csv files to .h5 files or merge HDF files. The .csv files have 94 columns, each with its own column name. A list with description of all columns with column name are found at `https://gea.esac.esa.int/archive/documentation/GDR2/Gaia_archive/chap_datamodel/sec_dm_main_tables/ssec_dm_gaia_source.html`.

1. **get_data(path, savepath, colnames):**
   Read data from .csv files and write it to .h5 files for the specified columns in the "colnames" argument. Create a number of parameter files with the specified columns. Can be wise to send in chuncks of the total amount of .csv files. The program stops if it cant find any .csv files

   Input:

path	*string*	The path to the .csv files folder.
savepath	*string*	The path to where the output files are stored.
colnames	*list*	The column names to use typed as string.

   Return:

   None

2. **get_csv_data(read_path, save_path):**
   Read in .csv data files of the Gaia DR2 and convert the relevant columns into .h5 files. This is time consuming, so call the function in five or more batches of the total number of files. The .csv files ends with even numbers before the extension. Then loop over the second to last digit. This approach gives 25 files per parameter. The program stops if it cant find any .csv files.

   Input:

read_path	*string*	The path to the .csv files
save_path	*string*	The path to where the output files are stored.

   Return:

   None

3. **fileMerger(in_files, name, out_file_name, type):**
   Merge the different data files for a parameter into one file. Then delete the old files.

   Input:

in_file	*string*	filename without extension of the parameter files to be merged.
name	*string*	name of the column.
out_file_name	*string*	name of parameter file to write.
type	*string*	what data type are to be written 'float', 'int' or 'str'.

   Return:

   None

4. **getCols(file, colname):**
   Get the column number for .csv files. Need to know what column names you want to use!

   Input

file	*string*	filename of the parameter files to read.
colname	*list*	name of the column.

   Return:

colnum	*list*	List of culumn number.

5. **call_get_data(last_digit, colnames):**
   This function call the **get_data()** function basically. The call are put into a function since we

need to change the working directory when running, the .csv files are not located in the same directory as the program file. Since there are a lot of .csv files, the call are done in a for loop over the second to last digit for lower time and memory usage. A list with description of all columns with column name are found at `https://gea.esac.esa.int/archive/documentation/GDR2/Gaia_archive/chap_datamodel/sec_dm_main_tables/ssec_dm_gaia_source.html`.

Input:

last_digit	*integer*	The last digit of the .csv files before extension. The last digit are given by even numbers so only five different combination. Use the last digit to read the .csv files in a simple and systematic way
colnames	*list*	List of the column names to use when reading the .csv files. See the Gaia DR2 documentation for what the alternatives are, link above.

Return:

None

4.2 File: AstroData.py

This program work with the HDF parameter files and convert the parallax angle into distance, and find the Healpix coordinates from the angular position.

1. **PixelCoord(Nside, ra_file, dec_file, save_path):**
   Create a .h5 of Healpix coordiantes from the position files containing right ascension and declination. Returns an array with the Healpix coordinates and write a file with the same Healpix coordinates for the given Nside. The created filename is 'SPC_Nside_#.h5', here # means the N_{side} value used.

   Input:

Nside	*int*	Nside of the Healpix coordinates.
ra_file	*string*	path to the right ascention file, must be a .h5 file.
dec_file	*string*	path to the declination file, must be a .h5 file.
save_path	*string*	the path to where the Healpix coordinate file are saved.

   Return:

pixpos	*array*	The Healpix coordinates for a given N_{side}.

2. **Positionfunc(dist_file):**
   Function to calculate the distance to the stars from the parallax file. Then write the distances to file, with filename 'Distance.h5'.

   Input:

dist_file	*string*	path to the parallax file to read.

   Return:

   None

4.3 File: DistanceMapping.py

The third program is for creating the maps, and contain functions for the different stages, in finding appropriate N_{side}, include the magnitude of the stars and distribute the stars after their distance from us.

1. **testNside(lower, upper):**
   Test different N_{side} resolutions of the maps and see what is best. By best we mean the map with good enough resolutions and not too many pixels, such that the memory and disk usage is

unnecessary large. The inputs are of the lower and upper power of two, $N_{side} = 2^p$, for $p > 0$ and $p < 12$, to get an upper and lower limit of the N_{side}.

Input:

lower	*scalar*	The lower power of 2, when finding the smallest N_{side}.
upper	*scalar*	The maximum power of 2, when finding the largest N_{side}.

Return:

None

2. **AllStars(Nside, filter=None):**
Function to plot all of the stars, with or without intensity.

Input:

Nside	*integer*	Which N_{side} to use for the map.
filter	*string*	Which filter to use: 'G', 'BP', 'RP' and 'mean', default is None and stops the program. Calls the **AddIntensiy** function in the **Tools.py** library.

Return:

None

3. **AllDistances(Bins, xscale='linear', yscale='linear'):**
Make a distribution of the distances to all of the stars and plot it as a histogram.

Input:

Bins	*integer*	Number of bins to distribute over. Using Matplotlib's histogram function.
xscale	*string*	The scaling of the x-axis, default is 'linear'. Can take the 'log', etc following matplotlib.gca().set_xscale() functionality.
yscale	*string*	The scaling of the y-axis, default is 'linear'. Can take the 'log', etc following matplotlib.gca().set_xscale() functionality.

Return:

None

4. **IntensityMapping(Nside, filter=None):**
Map the distribution of the different magnitudes of the stars, finds which stars having a magnitude within a range of magnitudes, and plot the stars in each magnitude range. Create also a weighted plot of the different magnitudes.

Input:

Nside	*integer*	The N_{side} to create maps of.
filter	*string*	Which filter to use in the map, either 'G', 'BP', 'RP' or 'mean'. If filter = None, then there is no magnitude to map and make unweighted maps.

Return:

None

5. **DistributeDistance(Nside, Nmaps=None, Rmax=5e4, Rmin=0.1, Bin_seq=None, cutoff=None, filter=None, h=False, OverError=False):**
Distribute the distances to the stars. This functions takes either a number of maps to create within a given distance range. Or a predefined sequence of distances giving bin edges, with this option a cutoff value can be given for creating less plots.

The function also take up to three different N_{side} values. When more than one N_{side} is given, the number of stars within a bin decides which N_{side} to use. Higher star density uses higher N_{side}.

Integrate the distance error for the stars close to a bin edge, so the magnitude are smeared out over the bins.

Input:

Nside	*integer*	scalar or list. The resolution(s) of the maps.
Nmaps	*integer, optional*	Scalar. The number of maps to create.
Rmax	*float*	The maximum distance to create maps up to, default is 50 kpc. Only in use when Nmaps are specified.
Rmin	*float*	The minimum distance to distribute from, default is 0.1 pc (there should not be any stars closer!). Only in use when Nmaps are specified.
Bin_seq	*list, optional*	List with bin edges of distances to make distributions from.
cutoff	*scalar, optional*	Cutoff value if want to create fewer maps than specified in the Bin_seq input.
filter	*string, optional*	If None, no weights are added. Else use the different filters 'G', 'BP', 'RP', 'mean' to weight the stars intensity.
h	*bool*	If true, creates histogram of the stellar distribution of distances.
OverError	*bool*	If true, compute and apply the error of the objects with uncertainty larger than 100%.

Return:

None

4.4 Tools.py

The fourth program are a tooling program with different functions used to calculate properties for the main programs. Some of the functions are using Numba to speed up the processes.

1. **Map(a, bins):**
   Calculate the histogram of a input array using NumPy histogram. Usefull when there is no weight applied to the histogram. Also use Numba to speed the calculation up.
   Input:

   Input:

a	*array*	The array to be mapped.
bins	*scalar or seq.*	Number of bins to calculate the histogram over, either at the number of bins or a predefined list of bins.

   Return:

hist	*array*	The histogram of the input array.
b	*array*	The computed bins

2. **Find_Nside(Nside, Nrad)**:
   Find the most appropriate Nside as function of number of stars in the current bin.

Input:

Nside	*seq or integer*	If scalar, use the given N_{side}. If a sequence use the most appropriate N_{side} according to the number of stars in the bin.
Nrad	*integer*	The number of stars in the current bin.

Return:

Nside	*integer*	The N_{side} to return.
Npix	*integer*	The number of pixels to use in the maps.

3. **PlotMaps(pix, Ns, Npix, w, rad, i, filter=None, h=False):**
Plotting function for DistributeDistance(). Create a mollweide map, and optional a histogram for distance.

Input:

pix	*array*	The pixel coordinates to make a map of.
Ns	*integer*	The current N_{side} value.
Npix	*integer*	The number of pixels in the map.
w	*array*	The weights for the stars in the map
rad	*array*	The distance array for the stars in the map
i	*integer*	Index of the current bin
filter	*string, optional*	If None, make a unweighted map, else weights the map according to the filter name
h	*bool*	If True, make a histogram of the stellar distance distribution of the star in the current bin.

Return:

None

4. **DistError(dist):**
Compute the distance error from the error in parallax. Returns an array with the distance error

Input:

dist	*array*	The distance array. Is needed in the calculation of the distance error, eq. (2).

Return:

dist_err	*array*	Array with the uncertainty in the distance in parsec.

5. **ApplyErrorNmaps(Nmaps, i, ind, Bins, pixcoord, rad_dist, dist_err, percentile, weight, uu_ind, over_ind, OverError=False):**
Function to do the error integration when running DistributeDistance() with Nmaps. Modifies the weights for stars with large uncertainty and append the stars with large uncertainty of the previous bin.

Input:

Nmaps	*integer*	The number of maps to create
i	*integer*	The index of the current bin.
ind	*sequence*	The indices of the stars in the current bin.
Bins	*sequence*	The list/array of bin edges.
pixcoord	*array*	The full array of the pixel coordinates
rad_dist	*array*	Full array with the radial distances
dist_err	*array*	The full array with the distance errors
percentile	*array*	The array of the distance percentile.
weight	*array*	The array with the weights of the stars
uu_ind	*sequence*	List or array with the indices of the stars with large error close to the upper bin edge from the previous bin.
over_ind	*sequence*	List or array with the indices of stars with error larger than 100%.
h	*bool*	If true, compute and apply the over error of the stars with uncertainty larger than 100%.

Return:

w	*array*	The updated array with weights for the current bin.
pixel	*array*	The updated array with pixels for the current bin
rad	*array*	Updated array with the radial distances for the current bin
iup	*sequence*	The indices for the stars with large error close to the upper bin edge.

6. **ApplyErrorBinSeq(Bin_seq, i, ind, pixcoord, rad_dist, dist_err, percentile, weight, uu_ind, over_ind, cutoff, OverError=False):**
Function to do the error integration when running DistributeDistance() with manually defined bin edges. Modifies the weights for stars with large uncertainty and append the stars with large uncertainty of the previous bin.

Input:

Bin_seq	*sequence*	The sequence with bin edges
i	*integer*	The index of the current bin.
ind	*sequence*	The indices of the stars in the current bin.
pixcoord	*array*	The full array of the pixel coordinates
rad_dist	*array*	Full array with the radial distances
dist_err	*array*	The full array with the distance errors
percentile	*array*	The array of the distance percentile.
weight	*array*	The array with the weights of the stars
uu_ind	*sequence*	List or array with the indices of the stars with large error close to the upper bin edge from the previous bin.
over_ind	*sequence*	List or array with the indices of stars with error larger than 100%.
cutoff	*integer*	The highest bin number to create map up to.
h	*bool*	If true, compute and apply the over error of the stars with uncertainty larger than 100%.

Return:

pix	*array*	The updated array with pixels for the current bin
w	*array*	The updated array with weights for the current bin.
rad	*array*	Updated array with the radial distances for the current bin
iup	*sequence*	The indices for the stars with large error close to the upper bin edge in the map.

7. **ErrorHandling(rad_dist, dist_err, percentile, Bins, weight, uu_ind, ilo, iup, i):** Compute the contribution of the stellar weights to the current bins for stars close to the bin edges.

Input:

rad_dist	*array*	Full array with the radial distances
dist_err	*array*	Full array with the distance errors
percentile	*array*	The full array with the distance percentiles
Bins	*sequence*	The sequence with bin edges
weight	*array*	The full array with magnitude weights
uu_ind	*sequence*	The indices for the stars with large error close to the upper bin edge of the pervious bin
ll_ind	*sequence*	The indices for the stars with large error close to the lower bin edge of the previous bin
ilo	*sequence*	The indices for the stars with large error close to the lower bin edge
iup	*sequence*	The indices for the stars with large error close to the upper bin edge
i	*integer*	The index of the current bin

Return:

weight	*array*	The updated weight array
wuu	*array*	Array with the weights of stars close to upper bin edge form the previous bin that can be appended to the current bin

8. **OverError(over_ind, Bins, rad_dist, dist_err, weight, pixcoord)** Compute the contribution of the distant stars with error over 100% and at a distance from the last bin up to 100 times that distance. For further stars we assume the contribution is negelctible. Can be added to the last bin.

Input:

over_ind	*sequence*	Sequence with the indices for stars with large error
Bins	*sequence*	List of the bin edges
rad_dist	*array*	Array with all of the radial distances to the stars
dist_err	*array*	Array with the distance errors to all of the stars
weight	*array*	Array with the magnitude weights to the stars
pixcoord	*array*	Array with the pixel coordinates to the stars

Return:

w0	*list*	List with the weights of the stars with over error
p	*list*	List with the pixel coordinates of the stars with over error
r	*list*	List with the distances of the stars with over error.

9. **IntegrateError(mu, sigma, edge):**
Integrate over the uncertainty distribution, assuming gaussian distribution, up to the bin edge input. Return the area under the integral.

Input:

mu	*scalar*	The distance of the star to be integrated over.
sigma	*scalar*	Standar error of the distance, used as a estimated standard deviation.
edge	*scalar*	The bin edge to integrate up.

Return:

a	*scalar*	The contribution to the weights, as a percentage to the weights.

10. **IntegrateOvererror(mu, sigma, b):**
Integrate over the uncertainty distribution, assuming gaussian distribution, up to the last bin edge input. Return the area under the integral.

Input:

mu	*scalar*	The distance of the star to be integrated over.
sigma	*scalar*	Standar error of the distance, used as a estimated standard deviation.
b	*scalar*	The last bin edge to integrate up to.

Return:

a	*scalar*	The contribution to the weights, as a percentage to the weights.

11. **AddIntensity(filter):** Make weights to the stars form the magnitude. Bright stars should have a higher pixel value than a faint star. This function sets the unknown magnitudes to the zero point values. Return the weights according to eq. (3) as a array.

Input:

filter	*string, optional*	If None, return None. Else use either 'G', 'BP', 'RP' or 'mean' to specify which filter to use, 'mean' computes the mean of the three filters.

Return:

w	*array*	The computes weights to the stars from their magnitude. Values depend on which filter used, and missing values, NaN, are set to the zero point value.

12. **parallax2dist(p):** Compute the distance to the stars from the parallax angel. Returns the distance in parsec.

Input:

p	*array or scalar*	Parallax angel in mas.

Return:

dist	*scalar or array*	The computed distance to the star/object(s) from parallax angle in the same shape as the parallax angles.

13. **PlotDist_DistError(dist=None):**
Function to plot the relation between the distane and the error. Read the distance file and calls DistError to get the data before plotting. The axis are logarithmic. Create also a plot of distance verses percentile. No input arguments.

Input:

dist	*array*	If None, read the distance file, else use the input array.

Return:

None

14. **NoDistance(dist, plot=False):**
Create a unfiltered map of the stars with unknown distance. Return the non zero indices.

Input:

dist	*array*	Array of the distances
plot	*bool*	if True make a map, if False, no plot.

Return:

ii	*sequence*	The indices where the distance is unknown

15. **Create_3Dmap(max_r, x=False, y=False, z=False):**
Make a 3D map in x, y, z coordinates. And optional in the three different planes. Note, this can takes a lot of time.

Input:

Max_r	*scalar*	The maximal distance to work with.
xy	*bool*	if True, make a plot in the xy plane.
xz	*bool*	if True, make a plot in the xz plane
yz	*bool*	if True, make a plot in the yz plane

Return:

None

5 References

1. This work has made use of data from the European Space Agency (ESA) mission *Gaia* (`https://www.cosmos.esa.int/gaia`), processed by the *Gaia*Data Processing and Analysis Consortium (DPAC, `https://www.cosmos.esa.int/web/gaia/dpac/consortium`). Funding for the DPAC has been provided by national institutions, in particular the institutions participating in the *Gaia* Multilateral Agreement.

2. M. Riello, F. De Angeli, D. W. Evans, G. Busso, N. C. Hambly, M. Davidson, P. W. Burgess, P. Montegriffo, P. J. Osborne, A. Kewley, J. M. Carrasco, C. Fabricius, C. Jordi, C. Cacciari, F. van Leeuwen and G. Holland (2018) Gaia Data Release 2: processing of the photometric data. ArXiv e-prints. External Links: 1804.09367, ADS entry Cited by: Credit and citation instructions, Figure 1.18, 1.1.3, 1.2.3, 10.8.2, 14.1.1, 14.1.1, 5.3.3, 5.3.4, 5.4.3, 5.4.3, 8.3.1, Gaia Data Release 2 Documentation release 1.2.

3. van Leeuwen, Floor. Hipparcor, the New Reduction of the Raw Data, Institute of Astronomy, Cambrigde University, Springer. 2008