find_magnitudes

June 15, 2020

1 Calculate magnitudes of the stars

Written by Evgenii N.

This code calculates the magnitudes of stars.

1.1 Prerequisite code

[1]:
```
# Import libraries that we will use later in this notebook
import os
from uncertainties.umath import log10
import shutil
import math
import ccdproc
import numpy as np
from astropy.visualization import ZScaleInterval, MinMaxInterval, ImageNormalize
from astropy import units as u
from astropy.stats import sigma_clipped_stats
from matplotlib.colors import LogNorm
from mpl_toolkits.axes_grid1 import make_axes_locatable, axes_size
from ccdproc import CCDData
import matplotlib.pyplot as plt
from photutils.aperture import CircularAperture, aperture_photometry
from photutils.centroids import centroid_2dg, centroid_com, centroid_1dg
from scipy.ndimage import shift
import pandas as pd
from tabulate import tabulate
from photutils import DAOStarFinder
from itertools import cycle
from cycler import cycler
from uncertainties import ufloat

# Make images non-blurry on high pixel density screens
%config InlineBackend.figure_format = 'retina'


def set_plot_style():
```

```
    """Set global style"""

    plt.rcParams['font.family'] = 'serif'

    TINY_SIZE = 15
    SMALL_SIZE = 18
    NORMAL_SIZE = 22
    LARGE_SIZE = 25

    # Title size
    plt.rcParams['axes.titlesize'] = LARGE_SIZE

    # Axes label size
    plt.rcParams['axes.labelsize'] = NORMAL_SIZE

    # Tick label size
    plt.rcParams['xtick.labelsize'] = TINY_SIZE
    plt.rcParams['ytick.labelsize'] = TINY_SIZE

    # Legend text size
    plt.rcParams['legend.fontsize'] = SMALL_SIZE

    plt.rcParams['font.size'] = NORMAL_SIZE
    plt.rcParams['legend.fontsize'] = NORMAL_SIZE


    # Grid color
    plt.rcParams['grid.color'] = '#cccccc'

    # Define plot size
    plt.rcParams['figure.figsize'] = [12, 8]

    # Marker size
    plt.rcParams['lines.markersize'] = 13


set_plot_style()
```

[2]:
```
def show_image(image, title=None, apertures=None, plot_dir=None, file_name=None,
               colorbar_fraction_width=0.05, pad_fraction=0.5, show=True):
    """
    Display an image.

    Parameters
    ----------

    image: astropy.nddata.ccddata.CCDData
```

```
    A fits image to show.

title: str
    Plot title.

apertures: list of CircularAperture
    List of apertures to plot over the image, optional.

plot_dir: str
    Directory where the plot file is saved.

file_name: str
    Name of the file where the plot is saved.

colorbar_fraction_width: float
    The width of the colorbar relative to the image width.

pad_fraction: float
    The size of the margin between the plot and the colorbar, relative
    to the colorbar width.

show: bool
    If False, do not or save the image


Returns
-------
fig, ax
    Matplotlib's figure and axis for the plot.
"""

fig, ax = plt.subplots()

# Scale the image similar to 'zscale' mode in DS9.
# This makes easier to spot things in the image.
interval=ZScaleInterval()
vmin, vmax = interval.get_limits(image)
norm = ImageNormalize(vmin=vmin, vmax=vmax)

plt.imshow(image, cmap='gray', norm=norm)  # Set color map and pixel scaling

if apertures is not None:
    apertures.plot(color='#33ff33', lw=1.5, alpha=0.8)

# Show the colorbar
divider = make_axes_locatable(ax)
width = axes_size.AxesY(ax, aspect=colorbar_fraction_width)
```

```
    pad = axes_size.Fraction(pad_fraction, width)
    cax = divider.append_axes("right", size=width, pad=pad)
    plt.colorbar(cax=cax)

    # Set axis labels
    ax.set_xlabel('x [pixel]')
    ax.set_ylabel('y [pixel]')

    fig.tight_layout()

    if show:
        # Save to file, show and close
        save_plot(fig, plot_dir=plot_dir, file_name=file_name)
        plt.show(fig)
        plt.close(fig)

        # Print plot label
        if title is not None:
            print(title)
            print()


    return fig, ax



def save_plot(fig, plot_dir, file_name):
    """
    Save a plot to a file.

    Parameters
    ----------

    fig: matplotlib.figure.Figure
        Plot's figure

    plot_dir: str
        Directory where the plot file is placed.

    file_name: str
        Plot file name

    """

    if file_name is None:
        return
```

```
    if not os.path.exists(plot_dir):
        os.makedirs(plot_dir)

    image_path = os.path.join(plot_dir, file_name)

    plt.savefig(image_path, fig=fig, dpi=150, transparent=False,
                bbox_inches='tight', pad_inches=0)



def print_image_stats(image, title):
    """
    Print first pixel value, average and standard deviation for an image.

    Parameters
    ---------

    image: astropy.nddata.ccddata.CCDData
        A fits image to show.

    title: str
        Image name.
    """

    data = np.asarray(image)# Get numpy array for image data
    label_len = 10  # Length of the text label
    first_pixel = data[0, 0]  # First pixel
    average = np.mean(data)  # Average
    standard_deviation = np.std(data)   # Standard deviation


    # Print values
    # -------

    print(
        f'\n{title}',
        f"\n{'-' * len(title)}",
        f"\n{'Pixel:':<10}{first_pixel:>10.2f} ADU",
        f"\n{'Avg:':<10}{average:>10.2f} ADU",
        f"\n{'Std:':<10}{standard_deviation:>10.2f} ADU\n"
    )


def save_image(image, file_path):
    """
    Save image to disk. Overwrites the file if it already exist.
```

```
    Parameters
    ---------
    image: astropy.nddata.ccddata.CCDData
        Image to be saved

    file_path: str
        Path where the image is saved
    """

    # Delete the file if it already exists

    try:
        os.remove(file_path)
    except OSError:
        pass

    # Create directory
    # ------

    dirname = os.path.dirname(file_path)

    if not os.path.exists(dirname):
        os.makedirs(dirname)

    image.write(file_path)
```

1.2 Selecting reference stars

I have choosen the reference stars shown in Table 1. These are the stars that I will use to calculate marnitudes of all other stars in non-photometric night image (March 9) using the photometric callibrations (Eq. 1-4) and measurements from photometric night image (March 29). I have chosen the stars using the following method:

- In DS9, opened images for all filters from photometric night, located in code/050_scaling_and_combining/march_29_2018_stacked directory and well as code/040_shift/data/shifted/march_29_2018/NGC_3201_B_30.000secs_00000472.fit image.
- Chose Zoom > Invert Y in DS9.
- In DS9, opened images for all filters from non-photometric night, located in code/050_scaling_and_combining/march_09_2018_stacked directory.
- Chose Zoom > Invert X in DS9. Now all FITS images are in same orientation. The images also almost match orientation on Aladin Lite web site: our fits files are rotated about 3 degrees clockwise with respect to Aladin.
- First I use the B-filter photometric image and look at stars that are visible on it (meaning they have significant blue component).

- Then, I locate those stars on other FITS images and make sure they are not oversaturated, meaning that the radius is not larger than 10 pixels.
- I make sure there are no other bright stars closer than 20 pixels. Our science images are oversaturated and it is hard to distibuish stars in the center of the cluter. Because of this, I used stars far away from the center.

Table 1: X-y pixel coordinates of stars from photomoteric (March 29 2018) and non-photometric (March 9 2018) images. RA/DEC coordinates and apparent B, V magnitudes are taken from SIMBAD database using Aladin Lite web site. Uncertainties for magnitudes were missing for all stars in SIMBAD except for V magnitude of star 2 (uncertainty 0.002 mag).

#	x, y position, March 29 2018	x, y position, March 9 2018	RAJ2000 [deg]	DEJ2000 [deg]	B	V
1	738, 468	48, 27	154.3111238862	-46.4466457933	14.730	13.800
2	494, 439	293, 56	154.3751324274	-46.4435287544	15.146	14.021
3	664, 127	123, 370	154.3346246701	-46.3859927913	15.020	13.870
4	460, 80	328, 417	154.3882668536	-46.3793802270	15.560	14.770
5	135, 250	653, 245	154.4705311270	-46.4124240426	15.520	14.540

Locations of the reference stars in the images are shown on Fig 1.

Figure 1: Five reference stars from Table 1. Top: photometric image from March 29 2018, B-band, y-axis is inverted. Middle: non-photometric image, March 9 2018, I-band, x-axis is inverted. Bottom: view of the cluster from Aladin Lite selected stars are from SIMBAD database.

1.3 Selecting empty regions for uncertainty calculation

I have described uncertainty calculations here

https://github.com/evgenyneu/asp3231_project/blob/master/doc/approximating_uncertainties/approximating_uncertainties.pdf

First, I need to select 20 regions without stars.

[3]:
```
def show_empty_regions(image, df_empty_regions, aperture_radius, title):
    """
    Show empty regions (i.e. regions without stars)

    Parameters
    ---------

    image: astropy.nddata.ccddata.CCDData
```

```
        An image.

    df_empty_regions: pandas.core.frame.DataFrame
        Positions of empty regions

    aperture_radius: float
        Aperture radius for empty regions.
    """

    # Create apertures for reference stars
    # ----------

    positions = list(
        zip(df_empty_regions["x"], df_empty_regions["y"])
    )

    apertures = CircularAperture(positions, r=aperture_radius)

    # Show reference image with apertures
    fig, ax = show_image(image=image, apertures=apertures, show=False)

    star_numbers = df_empty_regions.index.values

    for number, position in zip(star_numbers, positions):
        ax.text(x=position[0], y=position[1] - 13, s=str(number), color="white",
                horizontalalignment='center', fontsize=13)


    plt.show()
    print(title)
    print()
    plt.close(fig)


def calculate_flux_uncertainty_from_empty_regions(image, df_empty_regions,␣
↪aperture_radius):
    """
    Calculate flux uncertainty from empty regions.

    Parameters
    ----------

    image: astropy.nddata.ccddata.CCDData
        An image.

    df_empty_regions: pandas.core.frame.DataFrame
        Positions of empty regions
```

```
    aperture_radius: float
        Aperture radius for empty regions.

    Returns
    ------
    float
        Flux uncertainty in ADU.
    """

    positions = list(
        zip(df_empty_regions["x"], df_empty_regions["y"])
    )

    apertures = CircularAperture(positions, r=aperture_radius)
    fluxes = aperture_photometry(image, apertures)
    fluxes = fluxes['aperture_sum'].value
    uncertainty = fluxes.std()
    print(f"Empty regions mean: {fluxes.mean():.0f} ADU, std: {uncertainty:.0f}␣
↪ADU")
    return uncertainty


def empty_regions_uncertainties(image_dir, empty_regions_path, filter_names,
                                apertures, figure_number, file_name_func):
    filter_uncertainties = {}

    for filter_name in filter_names:
        figure_number += 1
        file_name = file_name_func(filter_name)
        image_path = os.path.join(image_dir, file_name)
        title = f"Figure {figure_number}: Empty regions in {file_name} file."

        aperture_radius = apertures[filter_name]

        # Read empty positions of empty regions
        df_empty_regions = pd.read_csv(empty_regions_path,␣
↪index_col="region_number")

        # Read image
        image = CCDData.read(image_path)

        show_empty_regions(image=image, df_empty_regions=df_empty_regions,
                           aperture_radius=aperture_radius, title=title)

        uncertainty = calculate_flux_uncertainty_from_empty_regions(
```

```
            image=image, df_empty_regions=df_empty_regions,␣
↪aperture_radius=aperture_radius)

        filter_uncertainties[filter_name] = uncertainty

    return filter_uncertainties
```

1.4 Calculate flux uncertainties for the empty regions for photometric image

[4]:
```
empty_regions_path = os.path.join("data", "empty_regions_march_29.csv")

# Apertures used to measure fluxes in photometric images
photometric_apertures = {
    "B": 6,
    "V": 8,
    "R": 10,
    "I": 10
}

figure_number = 1
filter_names = ["B", "V", "R", "I"]

photometric_flux_uncertainties = empty_regions_uncertainties(
    image_dir="../050_scaling_and_combining/march_29_2018_stacked",
    empty_regions_path=empty_regions_path,
    filter_names=filter_names, apertures=photometric_apertures,
    figure_number=figure_number,
    file_name_func=lambda filter_name: f"NGC_3201_{filter_name}_median_30.0s.
↪fits")

figure_number = 5
```

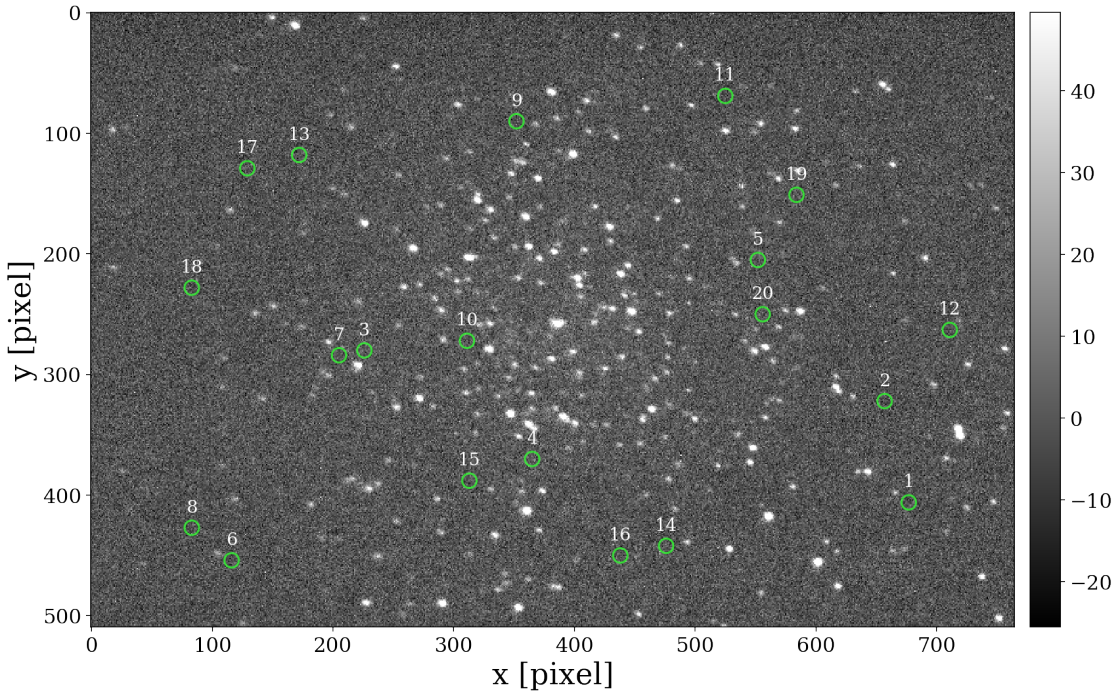


Figure 2: Empty regions in NGC_3201_B_median_30.0s.fits file.

Empty regions mean: -19 ADU, std: 120 ADU

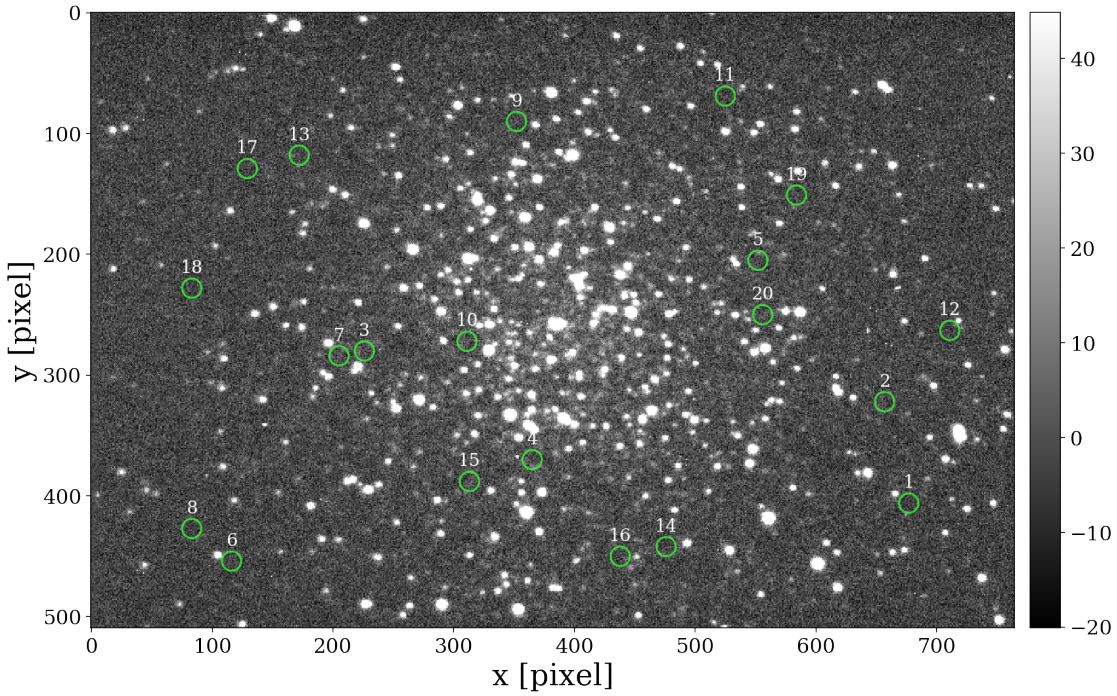


Figure 3: Empty regions in NGC_3201_V_median_30.0s.fits file.

Empty regions mean: -141 ADU, std: 285 ADU

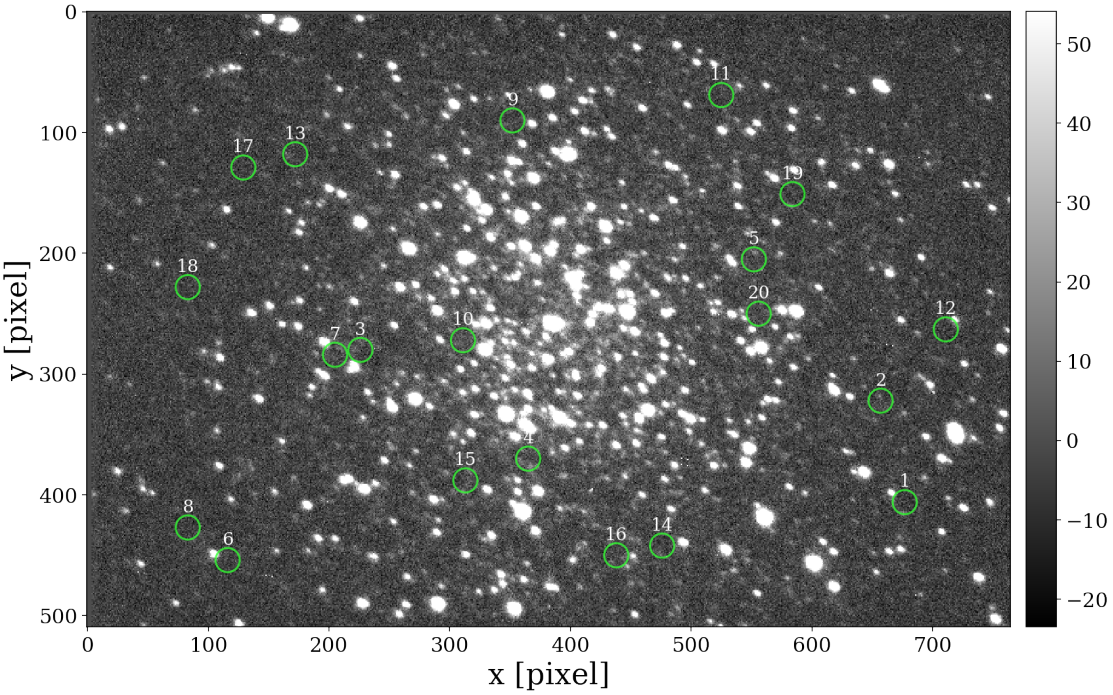


Figure 4: Empty regions in NGC_3201_R_median_30.0s.fits file.

Empty regions mean: -207 ADU, std: 672 ADU

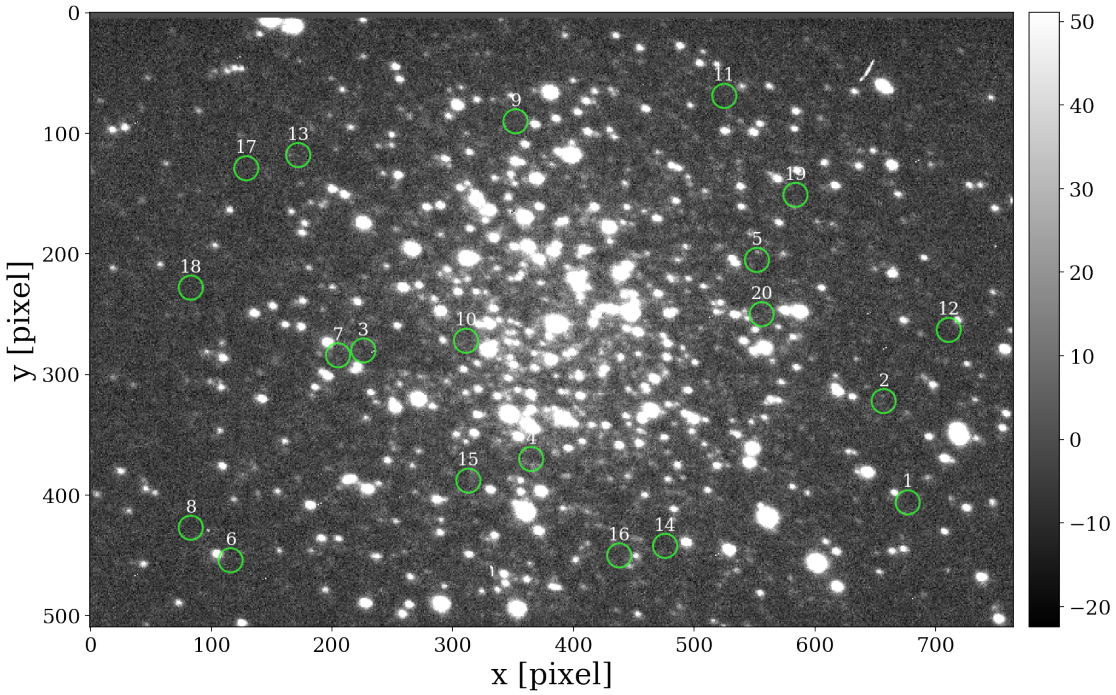


Figure 5: Empty regions in NGC_3201_I_median_30.0s.fits file.

Empty regions mean: -57 ADU, std: 773 ADU

1.5 Are empty region uncertainties any good?

- I look at Fig 2.5 and check that the green circles do not capture any stars.
- The standard deviations are form 100 to 800 ADU, not sure how reasonable this is.

1.6 Photometric callibation

The following equations are photometric calibrations for March 9 2018 image, supplied to us by teachers:

$B = 19.237 - 2.5\log(f/t) - 0.330 * A \quad (1)$

$V = 19.696 - 2.5\log(f/t) - 0.210 * A \quad (2)$

$R = 19.679 - 2.5\log(f/t) - 0.119 * A \quad (3)$

$I = 19.079 - 2.5\log(f/t) - 0.134 * A \quad (4)$,

where

- A is the airmass number, taken from AIRMASS value from header of the FITS file,
- f is the star's flux we measure in the image, in ADU,

- t is exposure time of the frame, in seconds.

[5]:
```
def calculate_reference_stars_magnitudes(reference_stars, reference_image_data,
                                         photometric_dir, figure_number,
                                        photometric_apertures, flux_uncertainties):
    """
    Calculates magnitued of the reference stars.

    Parameters
    ----------

    reference_stars: pandas.core.frame.DataFrame
        Positions of the reference stars

    reference_image_data: dict
        filter_name: str
            Name of the filter for the image
        aperture_radius: float
            Radius of the apertude to measure stars
        zero_point_mag: float
            The zero point magnitude term in the photometric equation.
        airmass_factor: float
            The air mass multiplier term in the photometric equation.

    figure_number: int
        The figure number that will be used in its caption.

    photometric_apertures: dict
        Dictionary containing apertures for measuring fluxes for each filter.
        Key: str
            Name of filter: "B", "V", etc.
        Value: float
            Aperture radius in pixels.

    flux_uncertainties: dict
        Key: str
            Filter name ("B", "V" etc.)
        Value: float
            Uncertainty of the flux in ADU.

    Returns
    --------
    list of float
        Magnitudes of the reference stars

    """
```

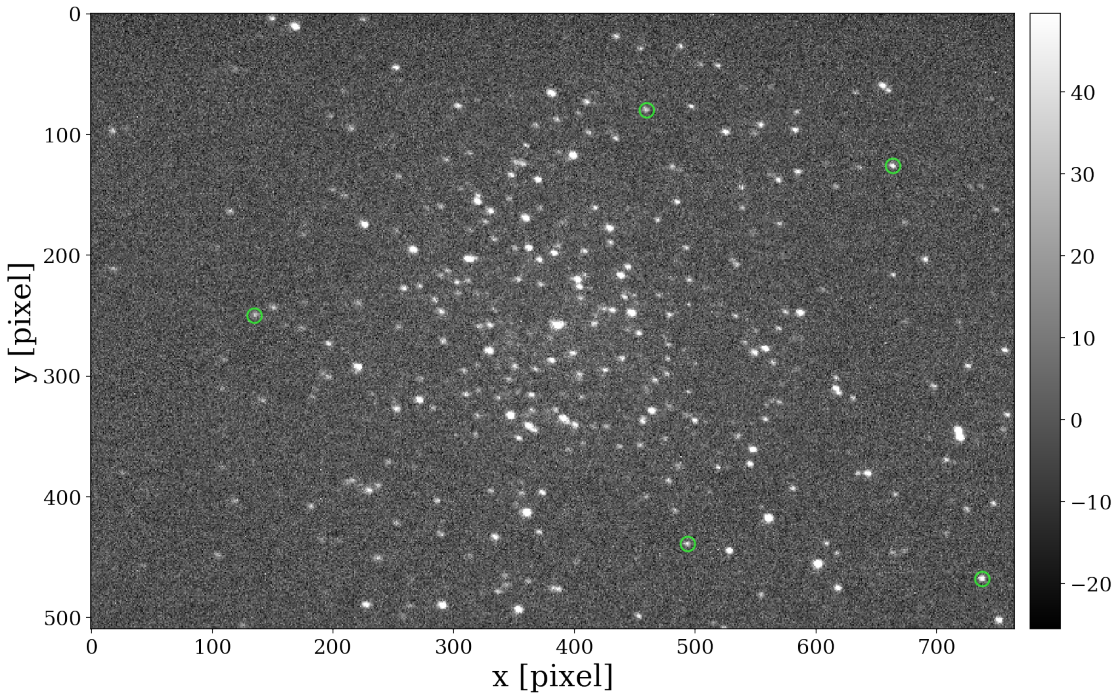
```
    # Set path to photometric image
    filter_name = reference_image_data["filter_name"]
    file_name = f"NGC_3201_{filter_name}_median_30.0s.fits"
    reference_image_path = os.path.join(photometric_dir, file_name)

    # Read photometric image
    reference_image = CCDData.read(reference_image_path)

    # Create apertures for reference stars
    # ----------

    reference_positions = list(
        zip(reference_stars["photometric_x"], reference_stars["photometric_y"])
    )

    aperture_radius = photometric_apertures[filter_name]
    reference_apertures = CircularAperture(reference_positions,␣
↪r=aperture_radius)

    # Show reference image with apertures
    title = f"Figrue {figure_number}: Reference stars in photometric image,␣
↪{filter_name} filter."
   show_image(image=reference_image, apertures=reference_apertures, title=title)
    plt.show()

    # Calculate fluxes of reference stars
    fluxes = aperture_photometry(reference_image, reference_apertures)

    # Calculate magnitudes of the stars using the photometric callibrations (Eq.␣
↪1-4)
    # -----------

    zero_point_mag = reference_image_data["zero_point_mag"]
    airmass_factor = reference_image_data["airmass_factor"]
    exposure_time = reference_image.header['EXPTIME']
    airmass_value = reference_image.header['AIRMASS']
    airmass_std = reference_image.header['AIRMSTD']
    airmass = ufloat(airmass_value, airmass_std)
    flux_uncertainty = flux_uncertainties[filter_name]

    fluxes_with_uncertainties = [
        ufloat(flux, flux_uncertainty) for flux in fluxes['aperture_sum'].value
    ]

    magnitudes = [
        zero_point_mag - \
                    2.5 * log10(flux / exposure_time) - \
```

```
                    airmass_factor * airmass
        for flux in fluxes_with_uncertainties
    ]

    return magnitudes
```

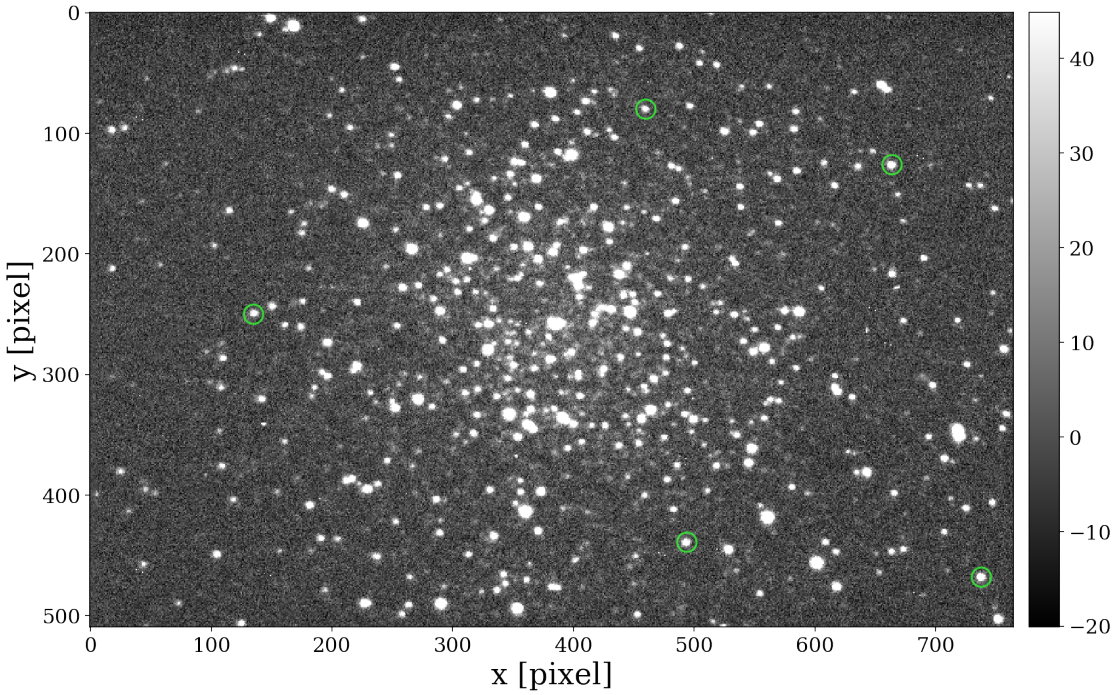
1.7 Calculate magnitudes of reference stars in photometric image

[6]:
```
# Load positions of reference stars
# ------

photometric_dir = "../050_scaling_and_combining/march_29_2018_stacked"
data_dir = "data"
reference_stars_path = os.path.join(data_dir, "reference_stars.csv")
reference_stars = pd.read_csv(reference_stars_path, index_col="star_number")

# Set properties of photometric images
reference_images = [
    dict(filter_name="B", zero_point_mag=19.237, airmass_factor=0.330),
    dict(filter_name="V", zero_point_mag=19.696, airmass_factor=0.210),
    dict(filter_name="R", zero_point_mag=19.679, airmass_factor=0.119),
    dict(filter_name="I", zero_point_mag=19.079, airmass_factor=0.134)
]

# Go over all photometric images
for i, reference_image_data in enumerate(reference_images):
    filter_name = reference_image_data["filter_name"]

    figure_number += 1

    # Calculate magnitudes of the reference stars
    magnitudes = calculate_reference_stars_magnitudes(
        reference_stars=reference_stars,␣
 ↪reference_image_data=reference_image_data,
        photometric_dir=photometric_dir,
        figure_number=figure_number,
        photometric_apertures=photometric_apertures,
        flux_uncertainties=photometric_flux_uncertainties)

    # Store magnitudes for the stars

    magnitude_values = [ round(magnitude.nominal_value, 2) for magnitude in␣
 ↪magnitudes ]
```

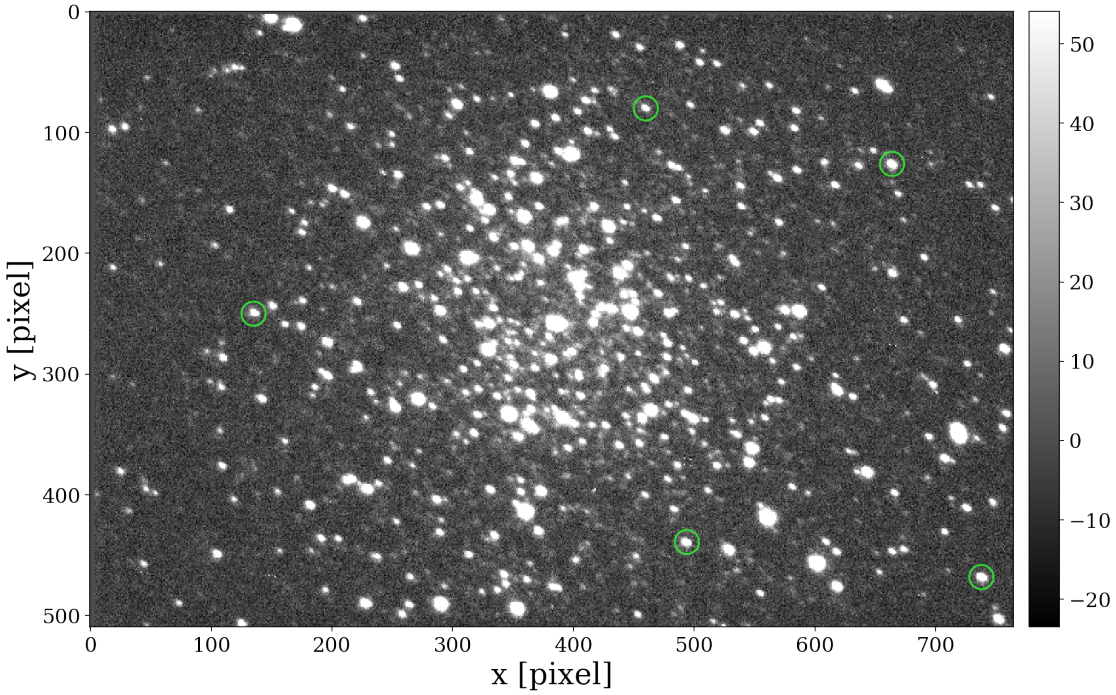
```
    magnitude_uncertainties = [ round(magnitude.std_dev, 2) for magnitude in␣
↪magnitudes ]

    reference_stars[f"{filter_name.lower()}_mag"] = pd.Series(magnitude_values,
                                                              index=reference_stars.index)

    reference_stars[f"{filter_name.lower()}_mag_err"] = pd.
↪Series(magnitude_uncertainties,
                                                                                 ␣
↪index=reference_stars.index)


# Save magnitudes to a CSV file
reference_stars_path = os.path.join(data_dir, "reference_stars_magnitudes.csv")
reference_stars.to_csv(reference_stars_path)
```



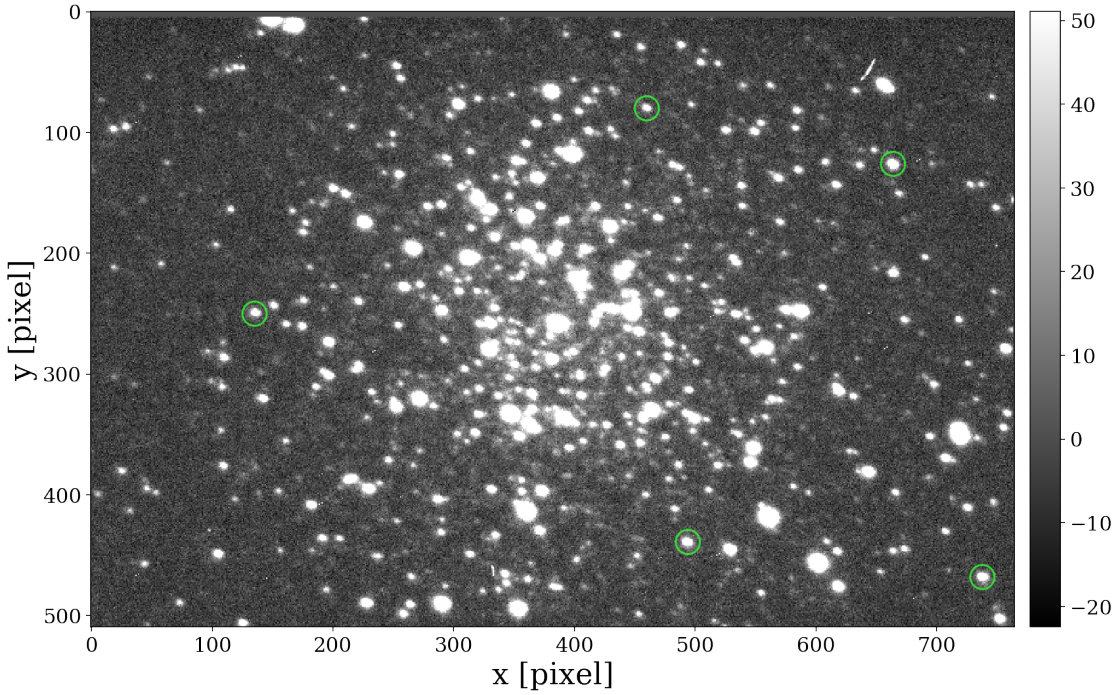
Figrue 6: Reference stars in photometric image, B filter.



Figrue 7: Reference stars in photometric image, V filter.



Figrue 8: Reference stars in photometric image, R filter.



Figrue 9: Reference stars in photometric image, I filter.

I look at Fig. 6-9 and check that green circles (apertures) of reference stars are located exactly like on Fig. 1. Also, I verify that the green circles have proper sizes such that they cover entire star with about 2 pixels of background around it.

1.8 Print magnitudes of reference stars in photometric image

I print calculated magnitudes of the reference stars to Fig. 2. I also show the magnitudes from SIMBAD for comparison.

[7]:
```
# Get only magnitude columns, since I don't want to print the entire table
df_printout = reference_stars[["b_mag_simbad", "b_mag", "b_mag_err",␣
 ↪"v_mag_simbad", "v_mag", "v_mag_err"]]

# Calculate difference between my magnitudes and SIMBAD's
# ------
delta_b = df_printout["b_mag_simbad"] - df_printout["b_mag"]
df_printout = df_printout.assign(delta_b=delta_b)
delta_v = df_printout["v_mag_simbad"] - df_printout["v_mag"]
df_printout = df_printout.assign(delta_v=delta_v)
```

```
# Set table column names
columns = ["#"] + list(df_printout.columns)

# Show the table
print("\nTable 2: Magnitudes of reference stars I calculated from photometric␣
 ↪images, as well as magnitudes from SIMBAD database.\n")
print(tabulate(df_printout, headers=columns, floatfmt=".2f", tablefmt="github"))
```

Table 2: Magnitudes of reference stars I calculated from photometric images, as well as magnitudes from SIMBAD database.

#	b_mag_simbad	b_mag	b_mag_err	v_mag_simbad	v_mag	v_mag_err	delta_b	delta_v
1	14.73	14.72	0.09	13.80	13.95	0.06	0.01	-0.15
2	15.15	15.14	0.14	14.02	14.17	0.08	0.01	-0.15
3	15.02	14.96	0.12	13.87	13.99	0.07	0.06	-0.12
4	15.56	15.49	0.19	14.77	14.84	0.14	0.07	-0.07
5	15.52	15.40	0.17	14.54	14.65	0.12	0.12	-0.11

1.9 Are my magnitudes any good?

The difference between our magnitudes and those from SIMBAD are:

[8]:
```
print(f"B mag: {df_printout['delta_b'].mean():.2f} ± {df_printout['delta_b'].
 ↪std():.2f}")
print(f"V mag: {df_printout['delta_v'].mean():.2f} ± {df_printout['delta_v'].
 ↪std():.2f}")
```

```
B mag: 0.05 ± 0.05
V mag: -0.12 ± 0.03
```

I can see that B and V magnitues are almost all within 2 uncertainties. Only exception is the first V stars, which is within 3 uncertainties.

1.10 Are magnitude uncertainties any good?

The magnitude uncertainties are about 0.1 - 0.2 mag, which are reasonable.

[9]:
```
def plot_mag_vs_uncertainties(filter_name, magnitudes, uncertainties):
    """
    Plot magnitudes and uncertainties

    Parameters
    ---------
    filter_name: str
        Filter name.

    magnitudes, uncertainties: list of float
        Magnitudes and their uncertainties.
    """

    fig, ax = plt.subplots()
    ax.grid(zorder=1)
    ax.scatter(magnitudes, uncertainties, zorder=2,
               color="#0084ff80",
               edgecolor="#0084ff", s=70)

    ax.set_xlabel(f"{filter_name} magnitude")
    ax.set_ylabel("Uncertainty [mag]")

    plt.show(fig)


for filter_name in ["B", "V"]:
    plot_mag_vs_uncertainties(filter_name=filter_name,
                              magnitudes=reference_stars[f"{filter_name.
 ↪lower()}_mag"],
                              uncertainties=reference_stars[f"{filter_name.
 ↪lower()}_mag_err"])

    figure_number += 1
    print(f"Figure {figure_number}: Magnitudes and uncertainties of reference␣
 ↪stars.")
```

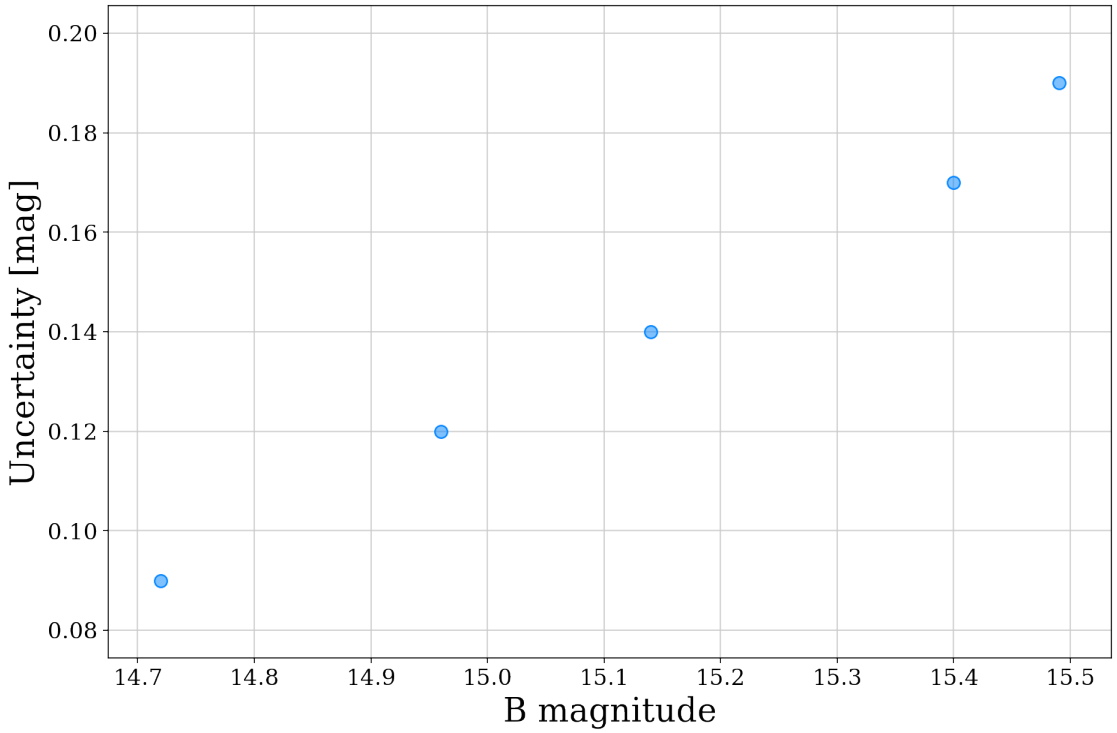


Figure 10: Magnitudes and uncertainties of reference stars.

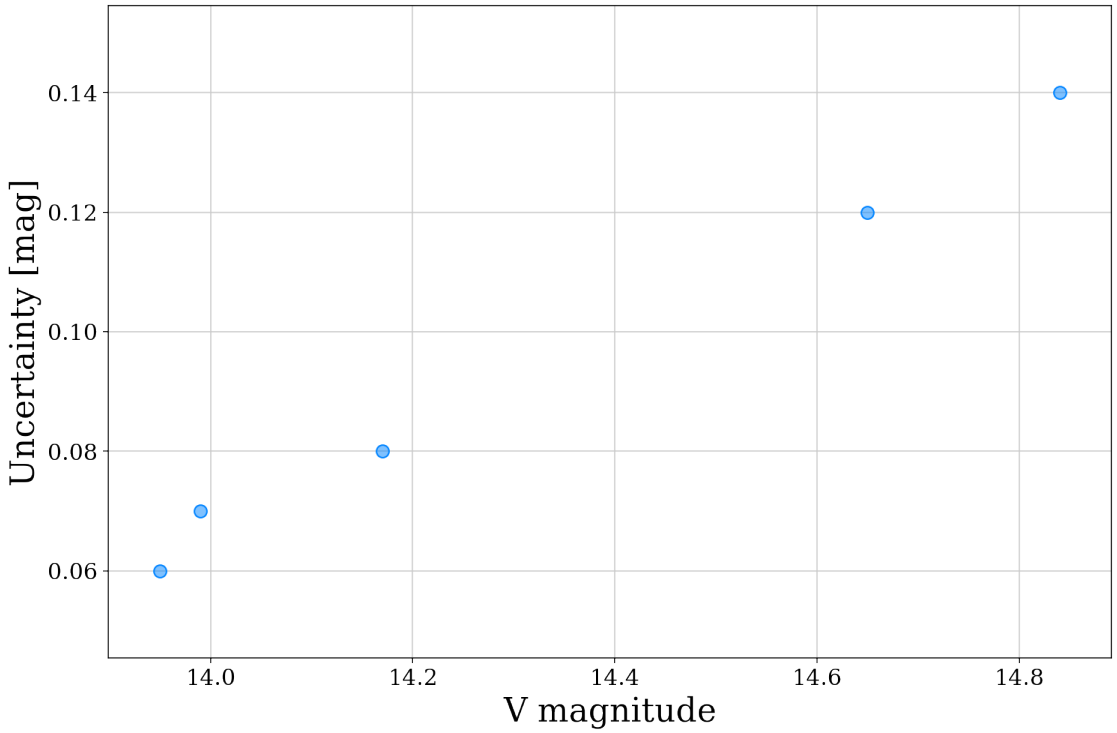


Figure 11: Magnitudes and uncertainties of reference stars.

1.11 How our uncertainties lookin'?

From Fig 10, 11 I can see that magnitudes and uncertainties have very strong linear relationship. Fainter stars have lager uncertainties, which is expected, since background uncertainties are more important for faint stars. I don't see anything weird here, so I'm happy. :)

1.12 Measure flux uncertainties of background in non-photometric images

Now I want to measure uncertainties of fluxes in non-photometric images. They will be used as uncertainties for the fluxes of the stars to calculate the uncertainties of their magnitudes. First, I plot the empty background regions.

[10]:
```
empty_regions_path = os.path.join("data", "empty_regions_march_09.csv")
non_photometric_aperture = 7

# Apertures used to measure fluxes in photometric images
non_photometric_apertures = {
    "B": non_photometric_aperture,
    "V": non_photometric_aperture,
    "R": non_photometric_aperture,
    "I": non_photometric_aperture
}

non_photometric_flux_uncertainties = empty_regions_uncertainties(
    image_dir="../050_scaling_and_combining/march_09_2018_stacked",
    empty_regions_path=empty_regions_path,
    filter_names=filter_names, apertures=non_photometric_apertures,
    figure_number=figure_number,
    file_name_func=lambda filter_name: f"NGC_3201_{filter_name}_median_60.0s.
 ↪fits")

figure_number = 16
```

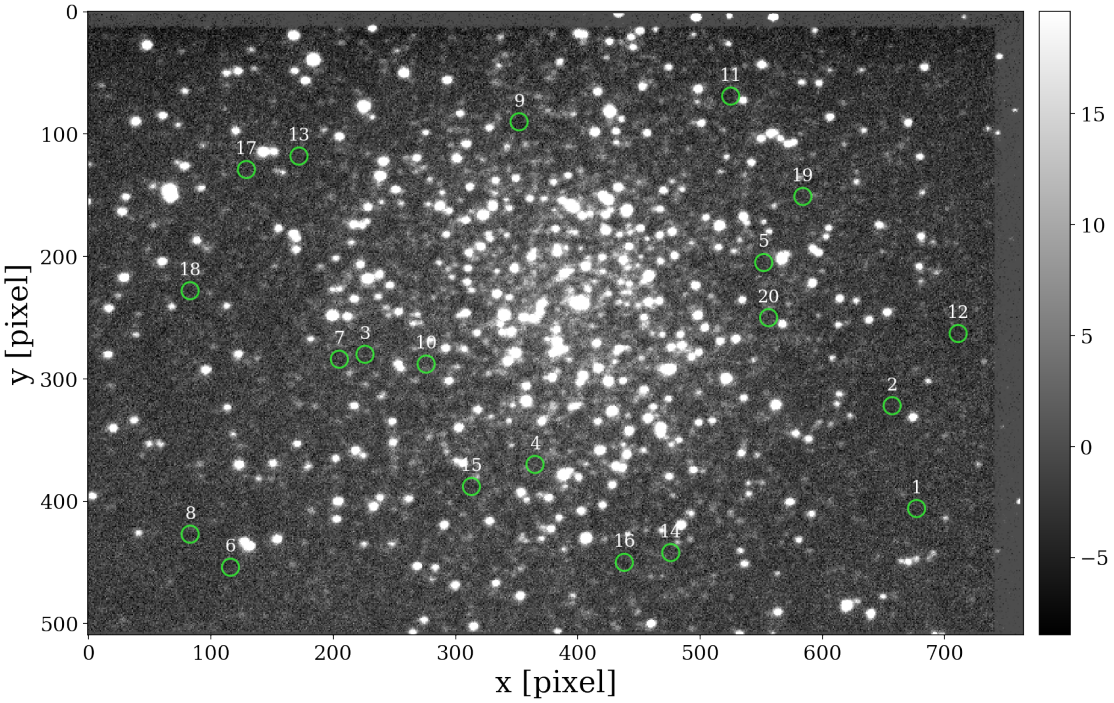


Figure 12: Empty regions in NGC_3201_B_median_60.0s.fits file.

Empty regions mean: -174 ADU, std: 111 ADU

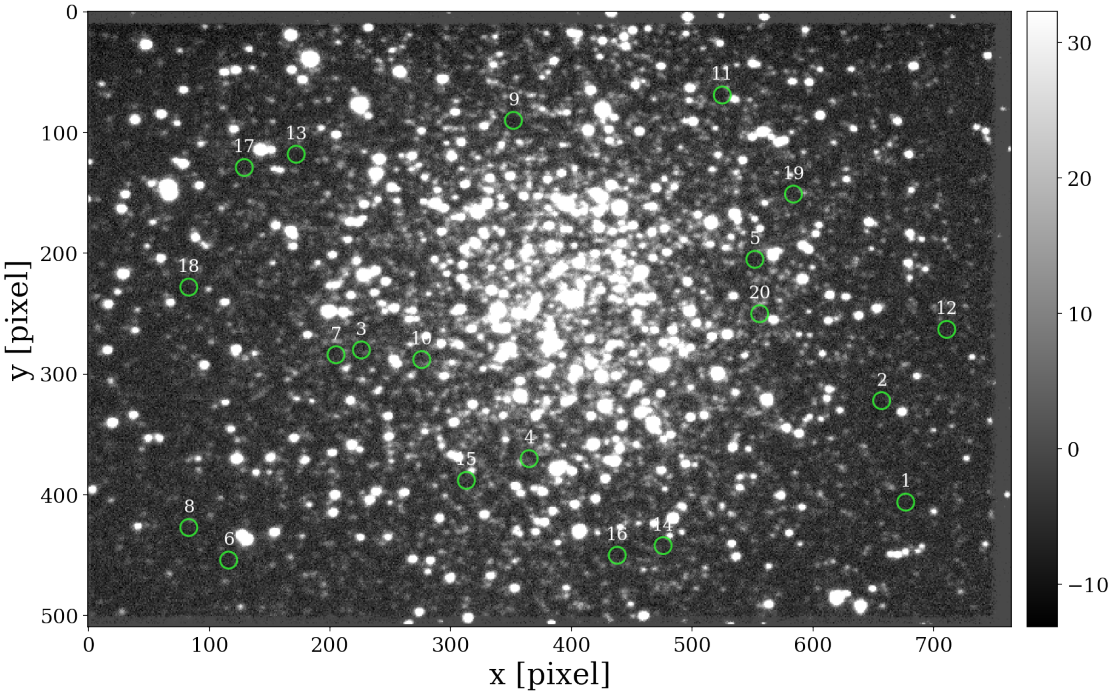


Figure 13: Empty regions in NGC_3201_V_median_60.0s.fits file.

Empty regions mean: -435 ADU, std: 243 ADU

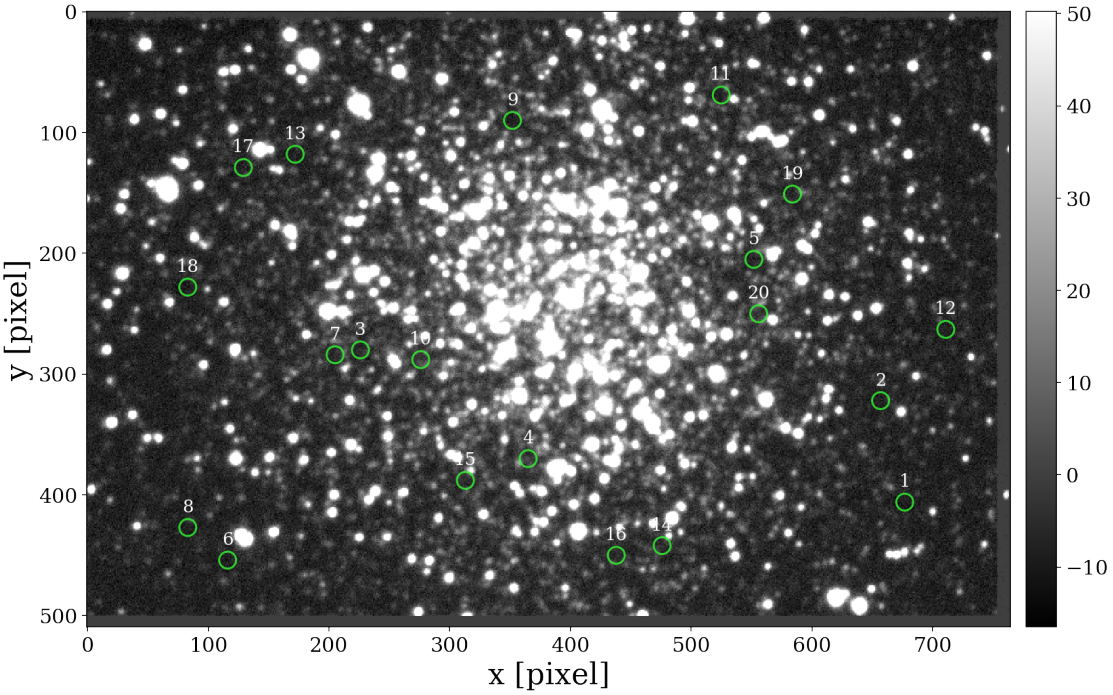


Figure 14: Empty regions in NGC_3201_R_median_60.0s.fits file.

Empty regions mean: -676 ADU, std: 427 ADU

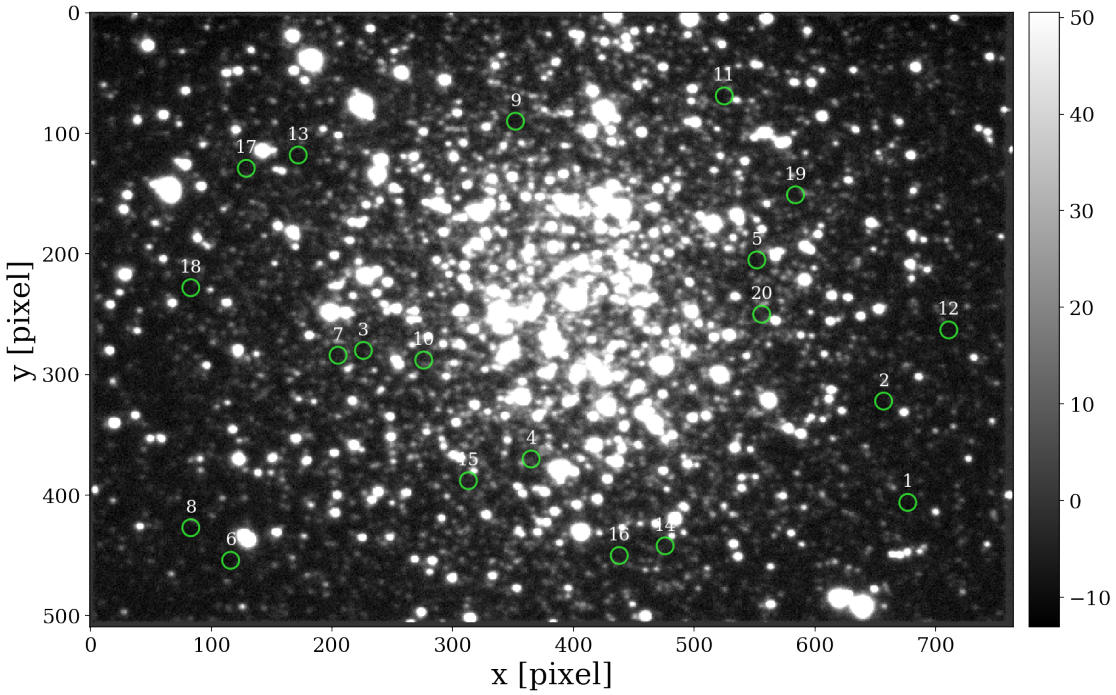


Figure 15: Empty regions in NGC_3201_I_median_60.0s.fits file.

Empty regions mean: -609 ADU, std: 433 ADU

1.13 How are empty regions lookin'?

I look at Fig. 11-15 and make sure the green circles do no capture anything super bright. I think is ok if there are moderately bright pixels inside, because those type of bright pixes or also captured by star apertures. So the background aperture better represent the averege background neighbourhood of a star.

1.14 Calculate magnitudes of stars in non-photometric night

Now that I calculated magnitued of reference stars in photometric night, I will use them to calculate magnitues of all stars in non-photometric night (March 9 2018).

Here is how it works.

1. Let m_2 be the magnitude of one reference star I calculated in photometric night, i.e. V=13.95 mag of the first star from Table 1.
2. I now use non-photometric night V image and measure the flux of the first reference star f_2.
3. Next, I measure flux f_1 of some other non-reference star I want to find magnitude for, in the same non-photometric image.

4. Then, I use the following equation to calculate the magnitude m_1 fo that star:

$$m_1 = m_2 - 2.5 \log(f_1/f_2). \quad (5)$$

5. Next, I repeat steps 1 through 4 for other four reference stars from Table 2. This will give me five measurements of m_1 for the same star.

6. Finally, I calculate the average and standard deviation of m_1, and that will be my final measured magnitude for the star.

[11]:
```
def get_distance(one, two):
    """
    Calculate distance in pixels between two points.

    Parameters
    ----------

    one, two: (x, y)
        Two points

    Returns
    -------
    float
        distance between `one` and `two`.
    """

    return np.sqrt((one[0] - two[0])**2 + (one[1] - two[1])**2)


def away_from_point(positions, distance, point):
    """
    Return positions that are `distance` pixels further from the `point`.

    Parameters
    ----------
    positions: list of (x, y)
        Positions.
    distance: float
        Min allowed distance from the center.
    point: (x, y)
        A coordinate.

    Returns
    -------
    list of (x, y)
        Positions that are `distance` pixels further away from the center
    """
```

```
    return [
        (x, y) for x, y in positions
        if get_distance(point, (x, y)) > distance
    ]


def away_from_center(positions, distance, image):
    """
    Return positions that are `distance` pixels firther from the center of the␣
 ↪image

    Parameters
    ----------
    positions: list of (x, y)
        Positions.
    distance: float
        Min allowed distance from the center.
    image: astropy.nddata.ccddata.CCDData
        Image.

    Returns
    -------
    list of (x, y)
        Positions that are `distance` pixels further away from the center
    """

    image_center = (image.data.shape[1] / 2, image.data.shape[0] / 2)

    return away_from_point(positions=positions, distance=distance,␣
 ↪point=image_center)


def away_from_each_other(positions, distance):
    """
    Exclude positions that are closer than `distance` to each other

    Parameters
    ----------
    positions: list of (x, y)
        Positions.
    distance: float
        Min allowed distance from the center.

    Returns
    -------
    list of (x, y)
        Positions that are further than `distance` pixels from each other.
```

```
    """

    return [
        (x, y) for x, y in positions
        if len(away_from_point(positions=positions, distance=distance,
                               point=(x, y))) == len(positions) - 1
    ]


def find_star_index(positions, distance, point):
    """
    Calculate the index of the object in the positions that is closer than␣
 ↪distance to `point`.

    Parameters
    ---------
    positions: list of (x, y)
        List of positions
    distance: float
        Distance.
    point: (x, y)
        A position.

    Returns
    -------
    int
        Index of the object in `positions`. Return None if none or more than one␣
 ↪objects found.
    """

    close_points = [
        i for i, position in enumerate(positions)
        if get_distance(point, position) < distance
    ]

    if len(close_points) == 1:
        return close_points[0]
    else:
        return None


def find_stars(image, aperture_radius, reference_stars, star_finder_threshold,
               min_distance_from_center, min_distance_between_stars):
    """
    Find positions of start in the `image`.

    Parameters
```

```
    ----------

    image: astropy.nddata.ccddata.CCDData
        An image to the stars in.

    aperture_radius: flost
        Radius of the star's aperture.

    reference_stars: pandas.core.frame.DataFrame
        A table containing positions of reference stars.

    star_finder_threshold: float
       The number of standard deviations of image brighness, the value is used in
        `threshold` parameter of DAOStarFinder. Lower values (like 5) allow to␣
↪detect fainter stars,
        but might produce unrealiable fluxes due to lower signal to noise.

    min_distance_from_center: float
       The minimum distance form the center of the image (in pixels)of the stars.
        This is needed to exclude stars that are in the very cetener, because it␣
↪is oversaturated
        and it's hard to distinguish stars.

    min_distance_between_stars: float
        The minimum distance between the stars, in pixels.
        This is needed to avoid including stars that are too close together, and
        therefore have overlaping signal.

    Returns
    -------
    list of (x, y)
        Positions of stars in the image.
    """

    # Calculate mean, meadian and standard deviation of pixel values,
    # excluding extreme values that are less or more than 3 standard deviations␣
↪from
    # a venter value.
    mean, median, std = sigma_clipped_stats(image.data, sigma=3.0, maxiters=5)

    # Locate sources
    daofind = DAOStarFinder(fwhm=aperture_radius,
                            threshold=star_finder_threshold * std,
                            exclude_border=True)

    # Find the sourcesdistance
    sources = daofind(image.data - median)
```

```
    # Keep only stars away from crowded center of the cluster
    positions_all = [ (x, y) for x, y in zip(sources['xcentroid'],␣
↪sources['ycentroid']) ]

    # Remove stars in the center of the cluster, since it is crowded and
    # it is hard to distinguish stars
    positions_all = away_from_center(positions=positions_all,
                                     distance=min_distance_from_center, image=image)

    # Remove stars with overlaping apertures. This is needed because stars
    # that are close to each other will have overlapping fluxed that we can't␣
↪distinguish.
    positions = away_from_each_other(positions=positions_all,
                                       distance=min_distance_between_stars)

    # Find indices of the reference stars in the array of all stars
    reference_stars_map_all =␣
↪find_reference_stars(reference_stars=reference_stars,
                                                  positions=positions_all)

    reference_stars_map_filtered =␣
↪find_reference_stars(reference_stars=reference_stars,
                                                      positions=positions)

    # Check that all reference stars are still present
    for star_number, index in reference_stars_map_filtered.items():
        if index is None:
            # The reference star was filtered, add it back
            positions.append(positions_all[reference_stars_map_all[star_number]])


    return positions


def plot_stars(image, filter_name, positions, figure_number, aperture_radius,
               reference_stars, plot_dir):
    """
    Plot the image and show positions of the stars on it.

    Parameters
    ----------

    image: astropy.nddata.ccddata.CCDData
        An image to the stars in.

    filter_name: str
```

```
        Name of the liter, i.e. "V".

    positions: list of (x, y)
        Position of all stars in the image.

    figure_number: int
        The figure number shown in the plot's caption.

    aperture_radius: flost
        Radius of the star's aperture.

    reference_stars: pandas.core.frame.DataFrame
        A table containing positions of reference stars.

    plot_dir: str
        Directory where the plot is saved.
    """

    apertures = CircularAperture(positions, r=aperture_radius)
    title = f"Figrue {figure_number}: Stars in non-photometric image,␣
↪{filter_name} filter."
    plot_file_name = f"selected_images_{filter_name.lower()}.pdf"

    fig, ax = show_image(image=image, apertures=apertures, title=title,␣
↪show=False)

    reference_positions = [
        (star[0], star[1])
        for star in reference_stars[["non_photometric_x", "non_photometric_y"]].
↪values
    ]

    reference_apertures = CircularAperture(reference_positions,␣
↪r=aperture_radius * 1.7)
    reference_apertures.plot(color='#ff7777', lw=3, alpha=1, axes=ax)

    # Save to file, show and close
    save_plot(fig, plot_dir=plot_dir, file_name=plot_file_name)
    plt.show(fig)
    plt.close(fig)

    # Print figure caption
    print(title)
    print()


def find_reference_stars(reference_stars, positions):
```

```
    """
    Find indices of the reference stars in the array of all stars.

    Parameters
    ----------

    reference_stars: pandas.core.frame.DataFrame
        A table containing positions of reference stars

    positions: list of (x, y)
        Position of all stars in the image.

    Returns
    -------
    dict
        key: int, deference star number
        value: index of the star in array of all stars
    """

    reference_stars_to_all_stars_map = {}

    for index, reference_star in reference_stars.iterrows():
        position = reference_star[["non_photometric_x", "non_photometric_y"]].
↪values
        reference_stars_to_all_stars_map[index] =␣
↪find_star_index(positions=positions, distance=2, point=position)

        if index is  None:
            raise RuntimeError(f"Can not find reference star at position␣
↪{position} for filter {filter_name}")

    return reference_stars_to_all_stars_map


def measure_magnitudes_with_single_reference_star(
    image, filter_name, fluxes, reference_star,
    reference_star_index, reference_flux_index,
    flux_uncertainty):
    """
    Measure magnitudes of all stars using single reference star.

    Parameters
    -----------
    image: astropy.nddata.ccddata.CCDData
        An image containing the stars.

    filter_name: str
```

```
        name of the filter

    fluxes: list of float
        List of fluxes for all stars.

    reference_stars: pandas.core.frame.DataFrame
        A table containing magnitudes of a single reference star.

    reference_star_index: int
        Reference star number

    reference_flux_index: int
        Index of the reference star in the list of all stars.

    flux_uncertainty: float
        Flux uncertainty (in ADU) in non-photometric image.

    Returns
    -------
    list of ufloat
        List of magnitudes and their uncertainties.
    """



    # Get magnitude and its uncertainty of the reference star
    m2 = reference_star[[f'{filter_name.lower()}_mag']].values[0]
    m2_error = reference_star[[f'{filter_name.lower()}_mag_err']].values[0]
    m2 = ufloat(m2, m2_error)

    # Get flux and its unceratainty of the reference star in non-photometric image
    f2 = fluxes[reference_flux_index]
    f2 = ufloat(f2, flux_uncertainty)

    exposure_time = image.header['EXPTIME']

    # Replace non-positive fluxes with small fluxes
    fluxes = [
        None if flux <= 0 else flux
        for flux in fluxes
    ]

    # Calculate the matnitues
#     magnitudes = [
#         m2 - 2.5 * log10(ufloat(flux, flux_uncertainty) / f2) if flux is not␣
 ↪None else None
#         for flux in fluxes
```

```
#     ]

    magnitudes = [
        m2.nominal_value - 2.5 * log10(flux / f2.nominal_value) if flux is not␣
↪None else None
        for flux in fluxes
    ]

    uncertainties = [
        math.sqrt(m2.std_dev**2 + ((2.5 * flux_uncertainty / math.log(10))**2) *␣
↪(1/(flux**2) + 1/(f2.nominal_value**2))) \
        if flux is not None else None
        for flux in fluxes
    ]

    magnitudes = [
        ufloat(mag[0], mag[1]) if mag[0] is not None else None
        for mag in zip(magnitudes, uncertainties)
    ]

    # Verify that m1 is equal to m2 for the reference star
    # ----------

    m1 = magnitudes[reference_flux_index]

    if round(m1.nominal_value, 5) != round(m2.nominal_value, 5):
        raise RuntimeError(
            (
                f"Reference magnitudes are different: {m1}, {m2}, "
                f"reference star {reference_star_index}, {filter_name} filter"
            )
        )

    return magnitudes



def measure_magnitudes_with_all_reference_stars(image, filter_name,␣
↪reference_stars,
                                                positions, aperture_radius,
                                                flux_uncertainty):
    """
    Measure magnitudes of all stars using all reference stars.

    Parameters
    ----------
    image: astropy.nddata.ccddata.CCDData
```

```
        An image containing the stars.

    filter_name: str
        name of the filter

    fluxes: list of float
        List of fluxes for all stars.

    reference_stars: pandas.core.frame.DataFrame
        A table containing magnitudes of all reference stars.

    positions: list of (x, y)
        Positions of all stars.

    aperture_radius: float
        Aperture radius for measuring fluxes.

    flux_uncertainty: float
        Flux uncertainty (in ADU) in non-photometric image.

    Returns
    -------
    numpy.ndarray
        A 2D array containing fluxes of all stars for all reference stars.
        First index is the reference star, and second corresponds to all stars.
    """

    # Find indices of the reference stars in the array of all stars
    reference_stars_map = find_reference_stars(reference_stars=reference_stars,␣
↪positions=positions)

    # Measure flux
    apertures = CircularAperture(positions, r=aperture_radius)
    fluxes = aperture_photometry(image.data, apertures)['aperture_sum']

    # For each reference star, calculate the magnitudes of all stars,
    # storing results in a nested list
    magnitudes = [
        measure_magnitudes_with_single_reference_star(
            image=image, filter_name=filter_name,
            fluxes=fluxes, reference_star=reference_star,
            reference_star_index=index,
            reference_flux_index=reference_stars_map[index],
            flux_uncertainty=flux_uncertainty)

        for index, reference_star in reference_stars.iterrows()
    ]
```

```
    # Convert nested list to a 2D array
    return np.array(magnitudes)


def find_filter_magnitudes(filter_name, non_photometric_dir, reference_stars,
                           aperture_radius, figure_number,
                           star_finder_threshold, min_distance_from_center,
                           min_distance_between_stars, plot_dir,
                           flux_uncertainty):
    """
    Measure magnitudes of all stars in non-photometric image for the given filter.

    Parameters
    ----------

    filter_name: str
        Filter name: B, V, I etc.

    non_photometric_dir: str
        Directory where non-photometric FITS files are located.

    reference_stars: pandas.core.frame.DataFrame
        A table containing positions and magnitudes of all reference stars.

    aperture_radius: float
        Aperture radius for measuring fluxes.

    figure_number: int
        The figure number that will be shown in plot's caption.

    star_finder_threshold: float
        The number of standard deviations of image brighness, the value is used in
        `threshold` parameter of DAOStarFinder. Lower values (like 5) allow to␣
↪detect fainter stars,
        but might produce unrealiable fluxes due to lower signal to noise.

    min_distance_from_center: float
        The minimum distance form the center of the image (in pixels)of the stars.
        This is needed to exclude stars that are in the very cetener, because it␣
↪is oversaturated
        and it's hard to distinguish stars.

    min_distance_between_stars: float
        The minimum distance between the stars, in pixels.
        This is needed to avoid including stars that are too close together, and
        therefore have overlaping signal.
```

```
    plot_dir: str
        Directory where the plots of selected stars are saved.

    flux_uncertainty: float
        Flux uncertainty (in ADU) in non-photometric image.

    Returns
    ------
    list of dict
        List of positions and magnitudes of the stars. Each element is a␣
↪dictionary with keys:
            position: (x, y)
                Position of the star in the image, in pixels.
            magnitude: ufloat
                The magnitude of the star.
            magnitude_stdev: float
                Starndard deviations of the magnitude estimates for the star␣
↪using the reference stars.
    """

    # Set path to non-photometric image
    image_path = os.path.join(non_photometric_dir,
                                    f'NGC_3201_{filter_name}_median_60.0s.fits')

    # Read non-photometric image
    image = CCDData.read(image_path)

    # Find positions of stars
    positions = find_stars(image=image, aperture_radius=aperture_radius,
                           reference_stars=reference_stars,
                           star_finder_threshold=star_finder_threshold,
                           min_distance_from_center=min_distance_from_center,
                          min_distance_between_stars=min_distance_between_stars)

    # Plot the apertures
    plot_stars(image=image, filter_name=filter_name,
               positions=positions, figure_number=figure_number,
               aperture_radius=aperture_radius,
               reference_stars=reference_stars,
               plot_dir=plot_dir)

    all_magnitudes = measure_magnitudes_with_all_reference_stars(
        image=image, filter_name=filter_name, reference_stars=reference_stars,
        positions=positions, aperture_radius=aperture_radius,
        flux_uncertainty=flux_uncertainty)
```

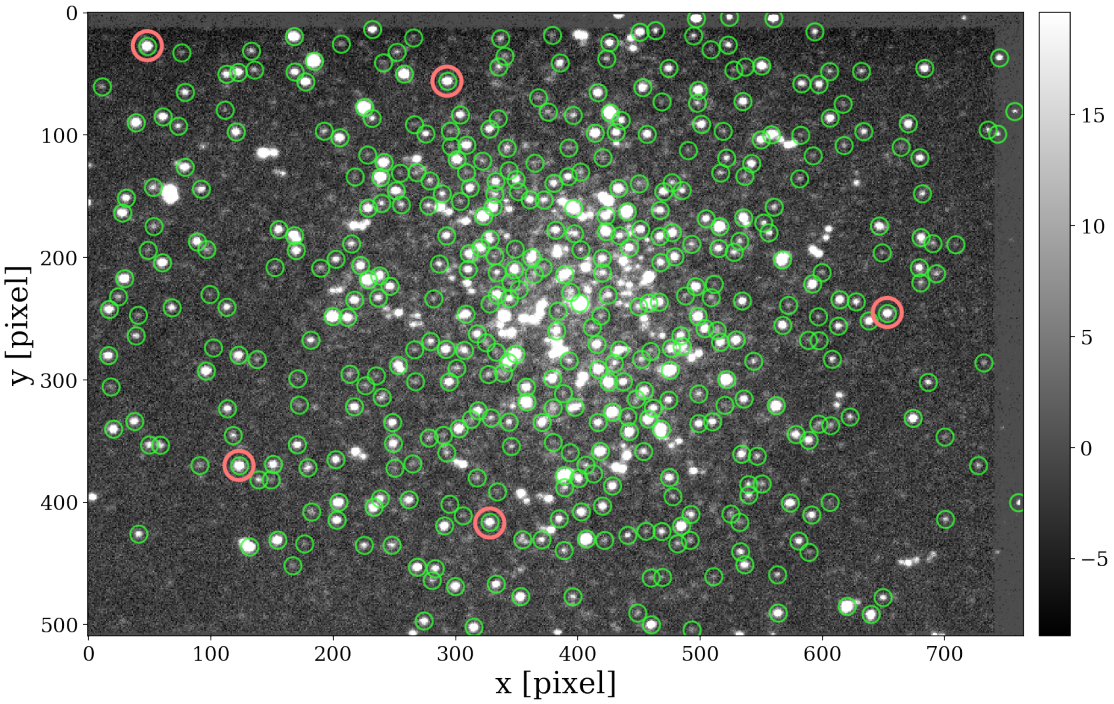
```
    positions_and_magnitudes = []

    # For each star, calcualte its the average and standard deviation
    # of its magnitude measurements for the reference stars
    for i, position in enumerate(positions):
        magnitudes = all_magnitudes[:, i]
        magnitudes = [magnitude for magnitude in magnitudes if magnitude is not␣
↪None]

        if len(magnitudes) == 0:
            continue

        magnitude_values = [magnitude.nominal_value for magnitude in magnitudes]
        magnitude_stdev = np.std(magnitude_values)

        # Get magnirude with middle value
        magnitudes = sorted(magnitudes, key=lambda value: value.nominal_value)
        magnitude_middle = magnitudes[int((len(magnitudes) - 1) / 2)]

        position_and_magnitude = dict(
            position=position,
            magnitude=magnitude_middle,
            magnitude_stdev=magnitude_stdev)

        positions_and_magnitudes.append(position_and_magnitude)

    return positions_and_magnitudes
```

1.15 Measure magnitudes of all stars for each filter

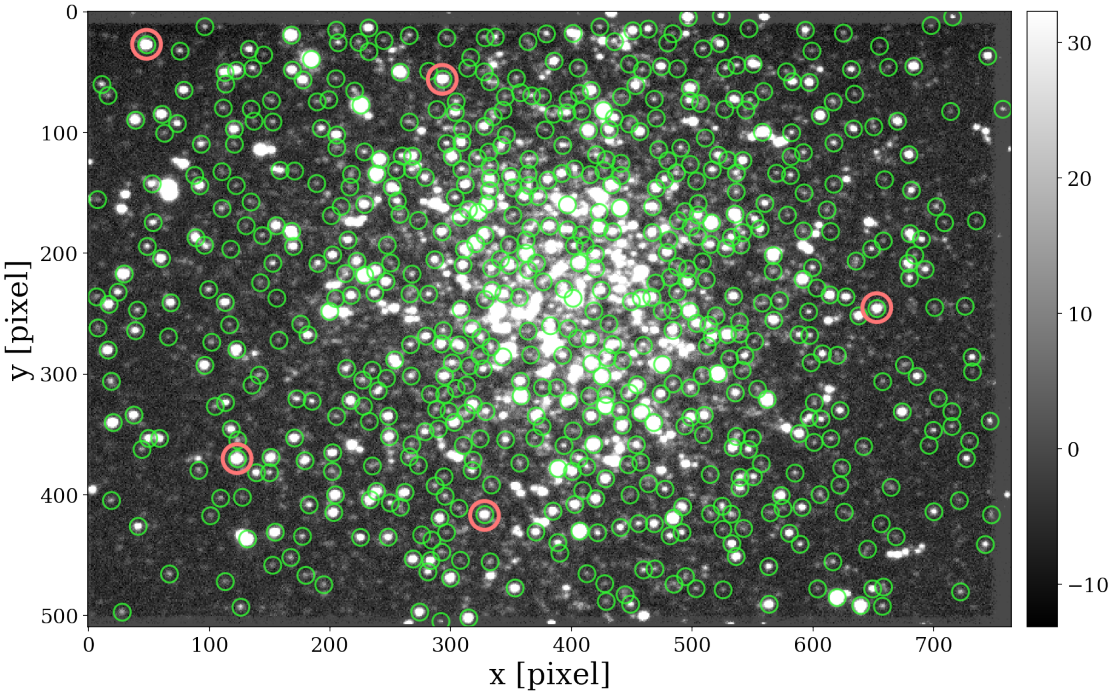
[12]:
```
def find_magnitudes(filter_names, non_photometric_dir,
                    reference_stars, aperture_radius,
                    star_finder_threshold, figure_number,
                    min_distance_from_center,
                    min_distance_between_stars,
                    flux_uncertainties):
    """
    Measure magnitudes of all stars for all filters.

    Parameters
    ----------

    filter_names: list of str
        Names of the image filters, i.e. ["V", "R"].

    non_photometric_dir: str
```

```
        Path to directory where the non-photometric FITS files are located.

    reference_stars: pandas.core.frame.DataFrame
        A table containing magnitudes of all reference stars.

    aperture_radius: float
        Radius of the aperture used to measure fluxes of the stars.

    star_finder_threshold: float
       The number of standard deviations of image brighness, the value is used in
        `threshold` parameter of DAOStarFinder. Lower values (like 5) allow to␣
↪detect fainter stars,
        but might produce unrealiable fluxes due to lower signal to noise.

    figure_number: int
        The number used in plot's caption.

    min_distance_from_center: float
       The minimum distance form the center of the image (in pixels)of the stars.
        This is needed to exclude stars that are in the very cetener, because it␣
↪is oversaturated
        and it's hard to distinguish stars.

    flux_uncertainties: dict
        Key: str
           Filter name ("V", "B", etc.)
        Value: float
            Flux uncertainty (in ADU) in non-photometric images.

    Returns
    -------
    dict
        key: filter name name, i.e. "V"
        value: list of dict
            List of positions and magnitudes of the stars. Each element is a␣
↪dictionary with keys:
                position: (x, y)
                    Position of the star in the image, in pixels.
                magnitude: ufloat
                    The magnitude of the star.
                magnitude_stdev: float
                   Starndard deviations of the magnitude estimates for the star␣
↪using the reference stars.
    """


    return {
```

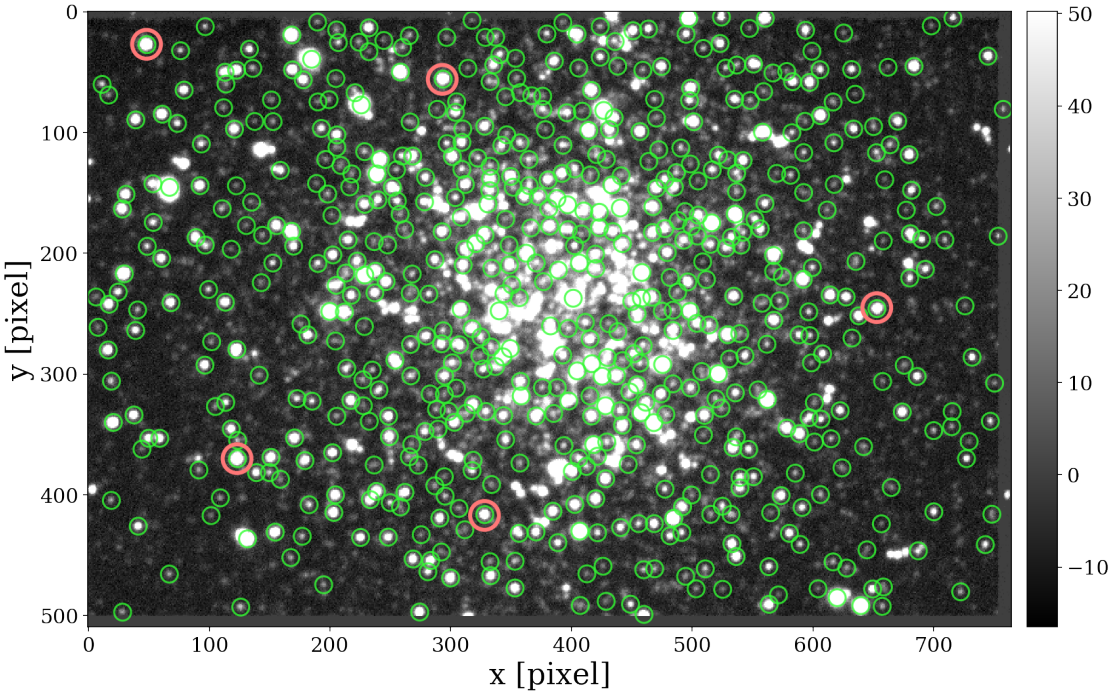
```
        filter_name: find_filter_magnitudes(
                        filter_name=filter_name,
                        non_photometric_dir=non_photometric_dir,
                        reference_stars=reference_stars,
                        aperture_radius=aperture_radius,
                        figure_number=figure_number + i,
                        star_finder_threshold=star_finder_threshold,
                        min_distance_from_center=min_distance_from_center,
                        min_distance_between_stars=min_distance_between_stars,
                        plot_dir="images",
                       ␣
 ↪flux_uncertainty=non_photometric_flux_uncertainties[filter_name])

        for i, filter_name in enumerate(filter_names)
    }


non_photometric_dir = "../050_scaling_and_combining/march_09_2018_stacked"
filter_names = ["B", "V", "R", "I"]

figure_number += 1

filter_magnitudes = find_magnitudes(filter_names=filter_names,
                                    non_photometric_dir=non_photometric_dir,
                                    reference_stars=reference_stars,
                                    aperture_radius=non_photometric_aperture,
                                    star_finder_threshold=5,
                                    figure_number=figure_number,
                                    min_distance_from_center=1,
                                    min_distance_between_stars=8.4,
                                   ␣
 ↪flux_uncertainties=non_photometric_flux_uncertainties)

print("Finished measuring magnitudes")
```



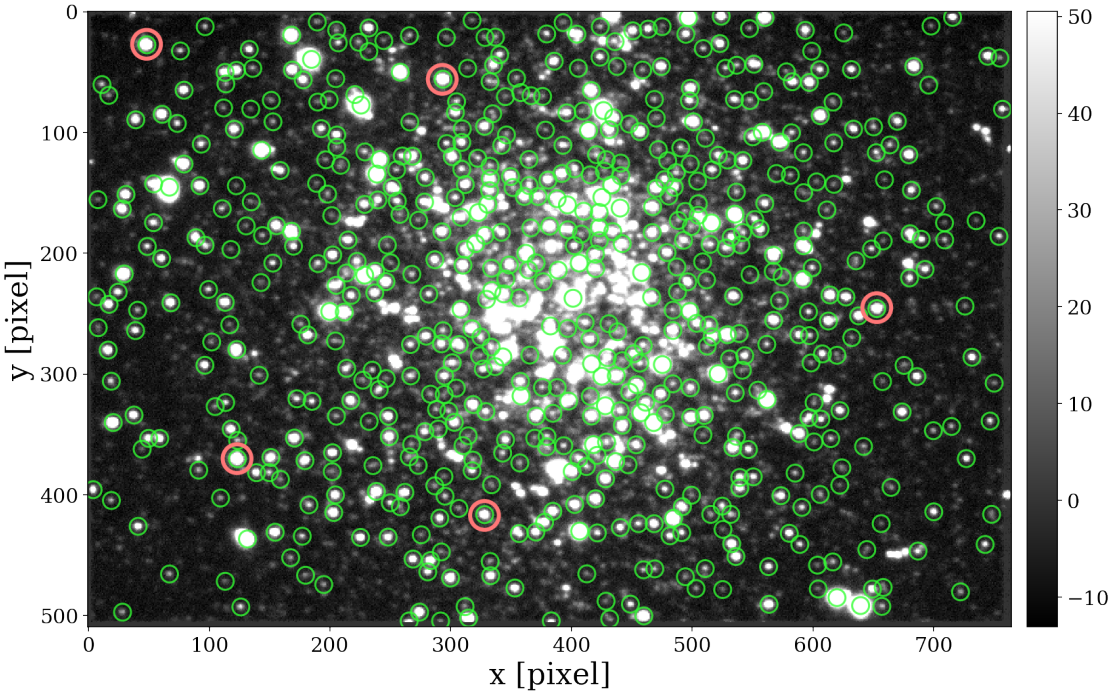
Figrue 17: Stars in non-photometric image, B filter.



Figrue 18: Stars in non-photometric image, V filter.



Figrue 19: Stars in non-photometric image, R filter.



```
Figrue 20: Stars in non-photometric image, I filter.

Finished measuring magnitudes
```

1.16 Are apertures any good?

I look at Fig. 17-20 and check these things:

1. Are the apertures (small green circles) match the stars well? Answer: yes
2. Are the apertures cover the stars completely? Answer: for most stars - yes. There are several fat oversaturated stars stars, especially in filter R and I. For those stars, the size of the aperture could be larger. It would be nice to have a way to programmatically detect those stars and make apertures larger for those.
3. Are the reference stars (larger red circles) match the green circles? Answer: yes.
4. There there multiple stars covered by the same aperture? Answer: no, for majority of stars.

Conclusion: I'm happy with star detection and aperture choice.

1.17 Verify standard deviations of magnitudes are the same for each filter

We measured magnitude of each star five times, using five reference stars. The standard devistion of these five numbers should be identical for all stars. Why? Ok, let's look at Eq. 5 again:

$m_1 = m_2 - 2.5\log(f_1/f_2) = m_2 + 2.5\log(f_2) - 2.5\log(f_1)$.

The measured flux of the star f_1 is the same between five measurements. So its five magnitudes m_1 will be equal to $m_2 + 2.5\log(f_2)$ plus constant value $-2.5\log(f_1)$. Therefore, the variation between five measurements is only due to variation of the $m_2 + 2.5\log(f_2)$ values. These values come from reference stars, which are the same for all stars. Thefore, all stars should have the same variation between five measurements. Or, in other words, the standard deviation of these five measurements will be the same.

Let's verify this in code.

[13]:
```
# Loop over all filters
for filter_name, positions_and_magnitudes in filter_magnitudes.items():
    standard_deviations = [
        position_and_magnitude['magnitude_stdev']
        for position_and_magnitude in positions_and_magnitudes
    ]

    # Raise error if standard deviations of magnitudes are different for each␣
↪stars
    if np.std(standard_deviations) > 1e-10:
        raise RuntimeError(f"ERROR: standard deviations vary for {filter_name}␣
↪filter")
```

1.18 Print standard deviations of five magnitudes

Now that we checked that standard deviations of five magnitudes for each filter are the same for all stars, let's print them.

[14]:
```
def print_standard_deviations(filter_magnitudes):
    """
    Print standard deviations of five measured magnitudes for each filter.

    Parameters
    ----------

    filter_magnitudes: dict
        key: filter name name, i.e. "V"
        value: list of dict
            List of positions and magnitudes of the stars. Each element is a␣
↪dictionary with keys:
                position: (x, y)
                    Position of the star in the image, in pixels.
                magnitude: ufloat
                    The magnitude of the star.
                magnitude_stdev: float
                   Starndard deviations of the magnitude estimates for the star␣
↪using the reference stars.
    """

    print("\nTable 3: Standard deviation of five magnitudes")

    # Loop over all filters
    for filter_name, positions_and_magnitudes in filter_magnitudes.items():
        # I only need standard deviation for one star
        # since they are identical for other stars, as I checked previously
        standard_deviation = positions_and_magnitudes[0]['magnitude_stdev']

        print(f'{filter_name}: {standard_deviation:.3f}')

        if standard_deviation > 0.2:
            raise RuntimeError(f"ERROR: magnitude standard deviation␣
↪{standard_deviation:.4f} mag is too large for {filter_name} filter")


print_standard_deviations(filter_magnitudes=filter_magnitudes)
```

```
Table 3: Standard deviation of five magnitudes
B: 0.085
V: 0.039
R: 0.035
```

```
I: 0.040
```

The standard deviations are 0.1 mag or smaller (Table 3), I'm happy with the results. The standard deviations is largest of B filter, probably because we used only one photometric image for it. Hmm, maybe I should not have removed that other blurry B image. I regret I did that.

> Maybe all one can do is hope to end up with the right regrets.
>
> Arthur Miller, The Ride Down Mt. Morgan

1.19 Save the magnitudes to a CSV file

The next step is to save the magnitudes of the stars in a CSV file. Here I will store the magnitudes in a table (Pandas data frame). Each row is a different star. The columns are, x-y positions, and B, V, R, I magnitudes.

The only nuiance here is that x-y positions of the same star will vary slightly between different filters, say (123, 42) for V image and (124, 41) for R image. But even though positions are diffrent, it's still the same star. So we want a single row in the table, not two rows. To achive that I compare coordinates of stars between filters, and consider them to correspond to the same star if the distance between the coordinates is smaller than two pixels.

[15]:
```
def save_magnitudes(filter_magnitudes, file_path):
    """
    Store positions and magnitudes of stars for all filters to a CSV file.

    Parameters
    ----------
    filter_magnitudes: dict
        key: filter name name, i.e. "V"
        value: list of dict
            List of positions and magnitudes of the stars. Each element is a␣
↪dictionary with keys:
                position: (x, y)
                    Position of the star in the image, in pixels.
                magnitude: ufloat
                    The magnitude of the star.
                magnitude_stdev: float
                   Starndard deviations of the magnitude estimates for the star␣
↪using the reference stars.

    file_path: str
        Path to the CSV file where the magnitudes are saved.

    Returns
    -------
    pandas.core.frame.DataFrame
        A table containing magnitudes of all stars.
    """
```

```
    filter_names = filter_magnitudes.keys()

    # Set column name that include positions and magnitudes for all filters
    filter_columns = [ [f'{name.lower()}_mag', f'{name.lower()}_mag_err'] for␣
↪name in filter_names ]
    filter_columns = sum(filter_columns, [])
    column_names = ["x", "y"] + filter_columns

    # Create empty table
    df = pd.DataFrame(columns=column_names)

    # Maximum distance between stars for different filters
    max_star_distance = 2

    # For each filter, store magnitudes of stars in the table
    for filter_name in filter_names:
        # Get the list of positions and magnitudes for the current filter
        positions_and_magnitudes = filter_magnitudes[filter_name]

        # Loop over all positions of the stars for the current filter
        for position_and_magnitude in positions_and_magnitudes:
            position = position_and_magnitude['position']
            mag = position_and_magnitude['magnitude']
            mag_std = position_and_magnitude['magnitude_stdev']

            # Check if the star is already present in the table
            # because its magnitude was measured for other filters
            # --------

            positions = df[["x", "y"]].values.tolist()

            star_index = find_star_index(positions=positions,
                                         distance=max_star_distance,
                                         point=position)

            if star_index is None:
                # This is a new star, add a row to the table
                # ------

                row = {
                    'x': int(round(position[0])),
                    'y': int(round(position[1])),
                    f'{filter_name.lower()}_mag': round(mag.nominal_value, 2),
                    f'{filter_name.lower()}_mag_err': round(mag.std_dev, 2),
                }
```

```
                df = df.append(row, ignore_index=True)
            else: # This star has already been added to the table
                # Find star's row in the table
                row = df.iloc[star_index]

                # Get stars's x-y position from the table
                # ------

                existing_xy = row[["x", "y"]].values.tolist()
                existing_x = existing_xy[0]
                existing_y = existing_xy[1]

                # Verify that star's position in the table is similar to
                # star's position for current filter
                if abs(existing_x - position[0]) > max_star_distance or \
                    abs(existing_y - position[1]) > max_star_distance:

                    raise RuntimeError(
                        (
                            f"Non matching positions in magnitudes table: "
                            f"{existing_xy} and {position}"
                        )
                    )

                # Update existing table row with star's magnitude
                row[f'{filter_name.lower()}_mag'] = round(mag.nominal_value, 2)
                row[f'{filter_name.lower()}_mag_err'] = round(mag.std_dev, 2)

    # Save coordinates as integers, not floats
    df.x = df.x.astype(int)
    df.y = df.y.astype(int)

    # Save magnitudes to a CSV file
    df.to_csv(file_path, index=False)

    return df


# Save magnitudes to a CSV file
magnitudes_path = os.path.join(data_dir, "magnitudes.csv")
df_magnitudes = save_magnitudes(filter_magnitudes=filter_magnitudes,␣
 ↪file_path=magnitudes_path)

print(f"Magnitudes are saved to: {magnitudes_path}")
```

```
Magnitudes are saved to: data/magnitudes.csv
```

1.20 Magnitudes I calculated

The magnitudes data are available here:

https://github.com/evgenyneu/asp3231_project/blob/master/code/060_find_magnitudes/data/magnitudes.csv

1.21 Check difference in magnitudes of reference stars from photometric and non-photometric images

Ok, now for each of five photometric stars, I have magnitudes measured both from photometric and non-photometric images. These measurements should be similar. Let's check that this is true.

[16]:
```
def verify_reference_magnitudes(data_dir, filter_names,␣
↪max_magnitude_difference):
    # Read the magnitudes from CSV files
    # ------

    df_non_photometric = pd.read_csv(os.path.join(data_dir, "magnitudes.csv"))

    df_photometric = pd.read_csv(os.path.join(data_dir,␣
↪"reference_stars_magnitudes.csv"),
                                 index_col="star_number")

    positions = df_non_photometric[["x", "y"]].values.tolist()
    max_star_distance = 2

    # Create a table to store magnitude differences between photometric and␣
↪non-photometric images
    # ------

    delta_column_names = [f'{name.lower()}_delta' for name in filter_names]
    diff_column_names = ["x", "y"] + delta_column_names


    df_delta = pd.DataFrame(columns=diff_column_names, index=df_photometric.
↪index)
    df_delta.index.name = "star_number"

    # Iterate over reference stars in photometric table
    for star_number, row_photometric in df_photometric.iterrows():
        # Get x-y coordinates of the star
        x_from_photometric = row_photometric['non_photometric_x']
        y_from_photometric = row_photometric['non_photometric_y']
        position = (x_from_photometric, y_from_photometric)

        # Find the position of this star in non-photometric table
```

```
        star_index = find_star_index(positions=positions,
                                     distance=max_star_distance,
                                     point=position)

        if star_index is None:
            raise RuntimeError(f"ERROR: Can't find reference star {position}")

        # Get the table row for the star from non_photometric table
        row_non_photometric = df_non_photometric.iloc[star_index]

        # List of column names for all magnitudes
        column_names = [f'{name.lower()}_mag' for name in filter_names]

        # Calculate difference of magnitudes betwen the photometric
        # and non-photometric images

        delta_mags = [
            row_photometric[column_name] - row_non_photometric[column_name]
            for column_name in column_names
        ]

        # Store the star's position and magnitude differences in the table
        # -------

        df_delta.loc[star_number]['x'] = x_from_photometric
        df_delta.loc[star_number]['y'] = y_from_photometric

        for filter_name, delta_mag in zip(filter_names, delta_mags):
            # Raise error if magnitude difference is too large
            if abs(delta_mag) > max_magnitude_difference:
                raise RuntimeError((
                    f"ERROR: magnitude difference {delta_mag:.3f} for "
                    f"reference star {star_number} is too lage"
                ))

            df_delta.loc[star_number][f'{filter_name.lower()}_delta'] =␣
↪round(delta_mag, 2)

    # Save coordinates as integers, not floats
    df_delta.x = df_delta.x.astype(int)
    df_delta.y = df_delta.y.astype(int)

    # Print the table of magnitude differences
    print("\nTable 5: Differences of magnitudes of reference stars between␣
↪photometric and non-photometric images.\n")

    print(tabulate(df_delta,
```

```
                    headers=["#"] + diff_column_names,
                    floatfmt=".2f", tablefmt="github"))

    # Calculate mean and standard deviation of magnitude differences
    # --------

    print("\n\nTable 6: Mean standard deviations of differences of magnitudes of␣
↪reference stars between photometric and non-photometric images.")
    print("----------------------")

    for filter_name in filter_names:
        column_name = f'{filter_name.lower()}_delta'
        print(f"{filter_name}: {df_delta[column_name].mean():>7.2f} ±␣
↪{df_delta[column_name].std():.2f} mag")


    # Save magnitudes to a CSV file
    file_path = os.path.join(data_dir, "magnitude_differences.csv")
    df_delta.to_csv(file_path, index=False)
    print(f'\nMagnitude differences saved to {file_path}')


verify_reference_magnitudes(data_dir=data_dir, filter_names=filter_names,␣
↪max_magnitude_difference=0.2)
```

Table 5: Differences of magnitudes of reference stars between photometric and non-photometric images.

#	x	y	b_delta	v_delta	r_delta	i_delta
1	48	27	-0.12	-0.01	0.00	0.00
2	293	56	-0.06	0.00	-0.04	-0.02
3	123	370	0.08	-0.01	-0.06	0.00
4	328	417	0.11	0.05	0.03	0.03
5	653	245	0.00	0.09	0.02	0.09

Table 6: Mean standard deviations of differences of magnitudes of reference stars between photometric and non-photometric images.

```
----------------------
B:    0.00 ± 0.10 mag
V:    0.02 ± 0.04 mag
R:   -0.01 ± 0.04 mag
I:    0.02 ± 0.04 mag
```

Magnitude differences saved to data/magnitude_differences.csv

1.21.1 Are my magnitude differences any good?

From Table 6 I can see that standard deviation of magnitude differences is 0.04 mag for all filter except B, where it is 0.10 mag. I don't know it these value are good or bad, will ask my teachers. These numbers are similar to standard deviations of magnitudes from 5 measurements I calculated earlier in Table 3. The magnitudes are more different for B filter probably because I only used one photometric B image.

1.22 Plot magnitude uncertainties

Next, I plot the magnitude uncertainties.

[17]:
```
df_magnitudes = pd.read_csv(os.path.join(data_dir, "magnitudes.csv"))
figure_number = 20

for filter_name in ["I", "R", "V", "B"]:
    err_column = f"{filter_name.lower()}_mag_err"
    df = df_magnitudes[df_magnitudes[err_column] < 2]

    plot_mag_vs_uncertainties(filter_name=filter_name,
                              magnitudes=df[f"{filter_name.lower()}_mag"],
                              uncertainties=df[err_column])

    figure_number += 1
    print(f"Figure {figure_number}: Magnitudes and uncertainties of stars.")
```

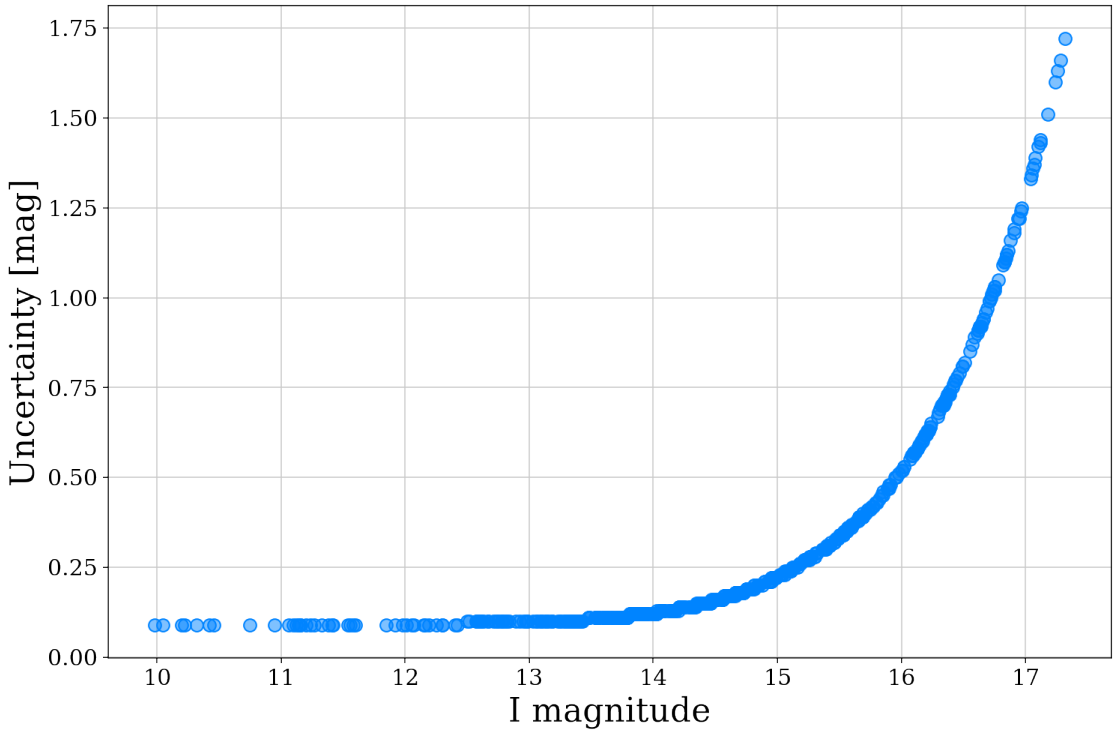


Figure 21: Magnitudes and uncertainties of stars.

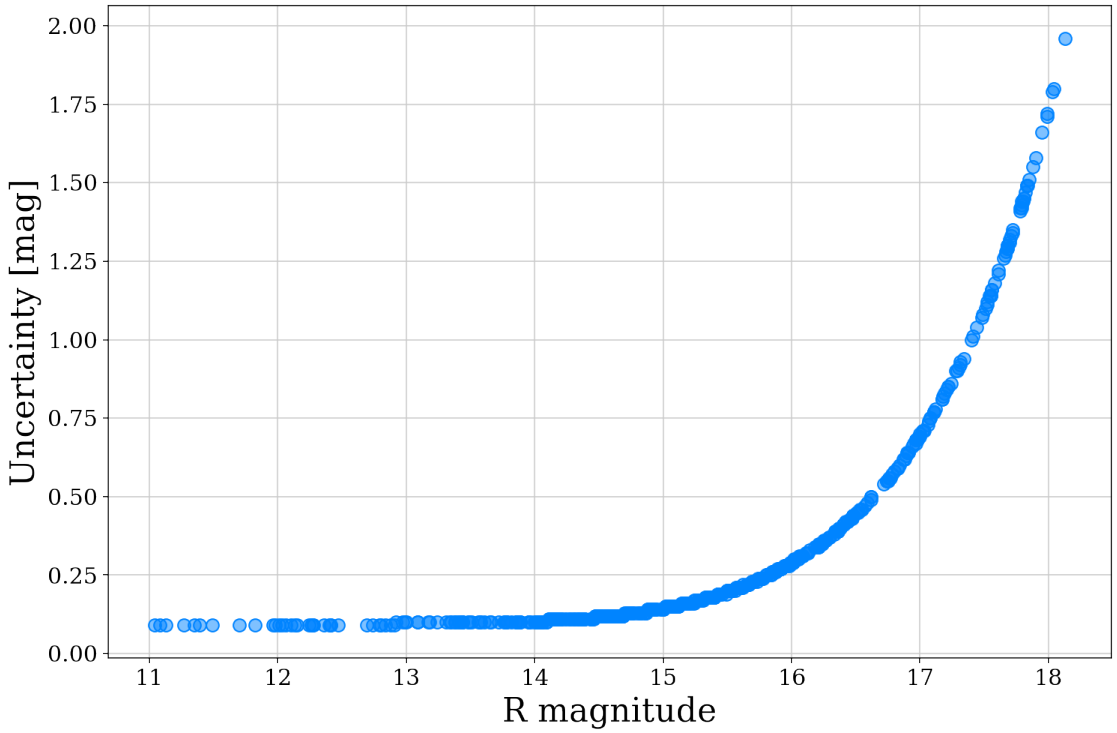


Figure 22: Magnitudes and uncertainties of stars.

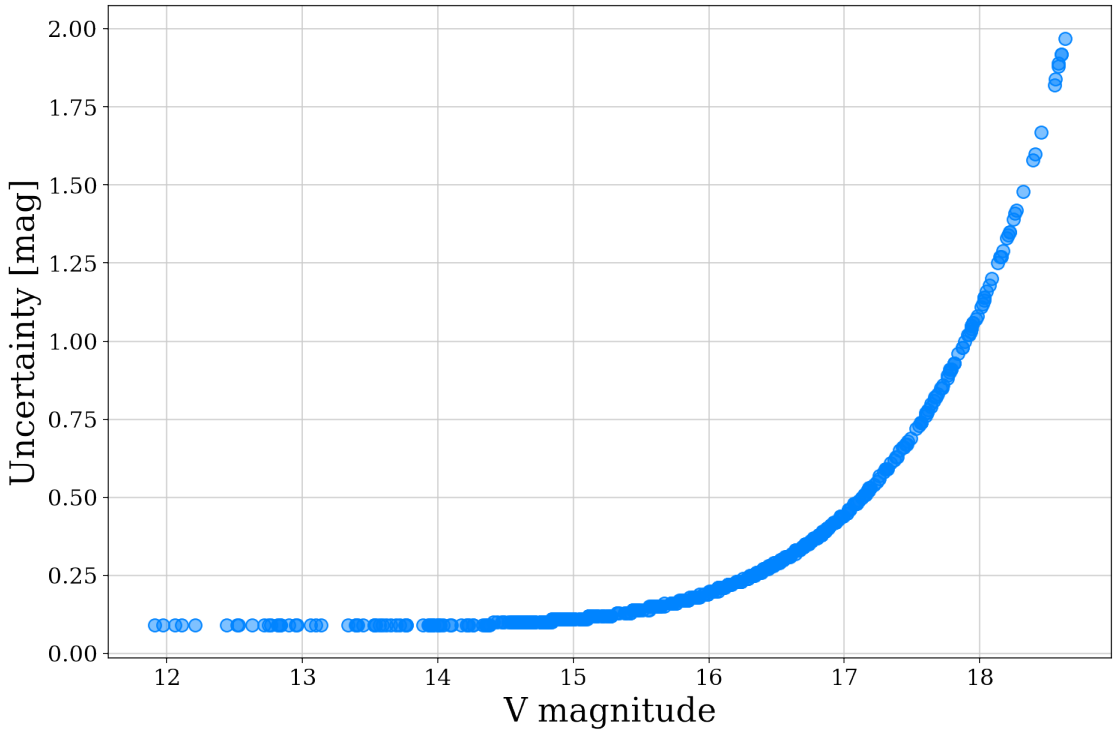


Figure 23: Magnitudes and uncertainties of stars.

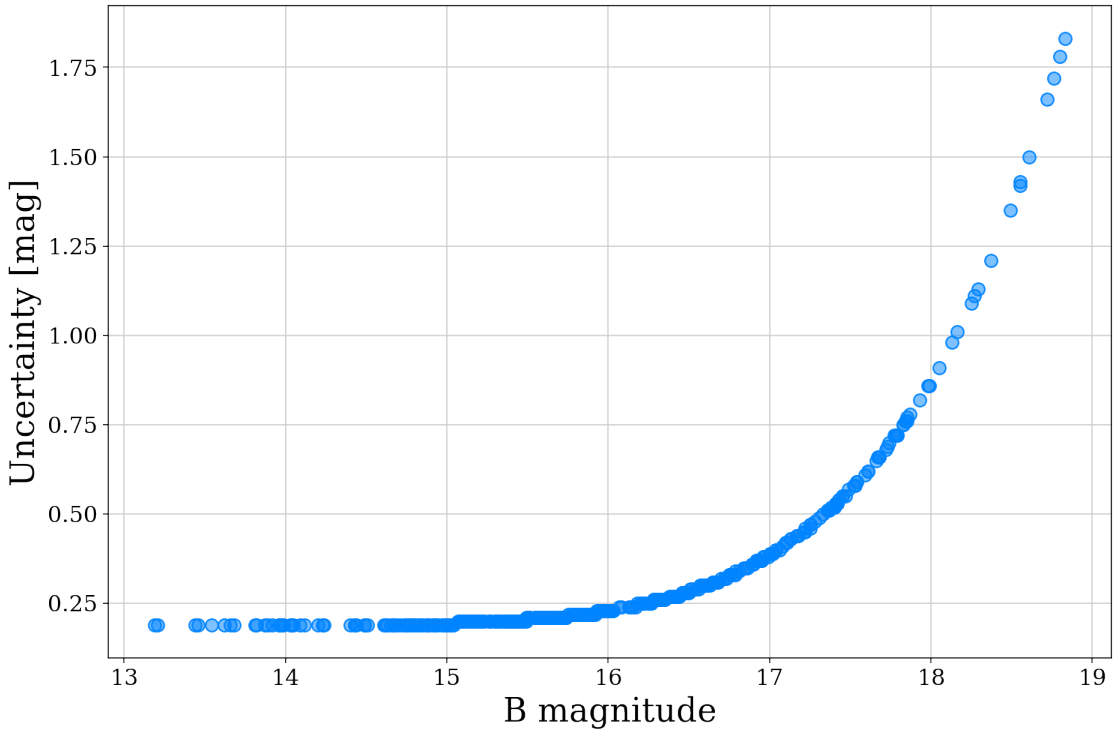


Figure 24: Magnitudes and uncertainties of stars.

1.23 How are magnitude uncertainties lookin'?

From Fig. 21-24 we can see that uncertainties are about 0.1 mag for brightest stars and then start to increase exponentially with magnitude. This makes sense, because failter stars should have larger magnitude uncertainties.

The small jumps in the data are due to

1.24 Plot color magnitude diagrams

As another check I plot a color-magnitude diagram. I expect to see the main sequence that turns into a red giant branch. Let's see if that's what's up.

[18]:
```
def make_cmd(plot_dir, file_name, data_path,
             blue_mag, blue_mag_err, red_mag, red_mag_err,
             x_label, y_label, title, xlims, ylims):
    """
    Make a plot of colour magnitude diagram.

    Parameters
    ----------
    plot_dir: str
        Directory where the plot file is placed.

    file_name: str
        Plot file name.

    data_path: str
        Path to the CSV file containing magnitudes for stars.

    blue_mag, red_mag: str
        Names of the column containing magnitudes for the bluer and redder␣
↪filters.

    x_label, y_label: str
        Axes labels.

    title: str
        Plot's title.

    xlims, ylims: (low, high)
        Limits for the axes.
    """
```

```
# Read magnitudes and colors from CSV file
df = pd.read_csv(data_path)

# Create a figure and axis object
fig, ax = plt.subplots(1, 1)

# Drop rows with missing values
df = df.dropna(subset=[blue_mag, red_mag])

x_values = df[blue_mag] - df[red_mag]

# Calculate the color uncertainty
x_errors = np.sqrt(df[red_mag_err]**2 + df[blue_mag_err]**2).values

# Error bars
ax.errorbar(x_values, df[red_mag], xerr=x_errors, yerr=df[red_mag_err],
            zorder=1, fmt='none', ecolor="#0084ff40")

# Show plot
ax.scatter(x_values, df[red_mag], zorder=2,
           color="#0084ff40",
           edgecolor="#0084ff")



# Show grid
ax.grid(zorder=-1)

# Set plot labels
ax.set_xlabel(x_label)
ax.set_ylabel(y_label)

# Set axes limits
# ------

if xlims is not None:
    ax.set_xlim(xlims)

if ylims is not None:
    ax.set_ylim(ylims)

# Invert y axis
ax.invert_yaxis()

# Expand the plot to the edges
fig.tight_layout()
```

```
    save_plot(fig=fig, file_name=file_name, plot_dir=plot_dir)
    plt.show(fig)
    plt.close(fig)

    # Print image caption
    print(title)
    print()


data_path = os.path.join(data_dir, "magnitudes.csv")

all_plot_settings = [
    dict(magnitudes=["B", "V"], xlims=(-0.5, 3), ylims=(11.5, 18)),
    dict(magnitudes=["B", "R"], xlims=(-0.5, 3), ylims=(10, 18)),
    dict(magnitudes=["B", "I"], xlims=(-0.5, 5), ylims=(10, 18)),
    dict(magnitudes=["V", "R"], xlims=(-0.5, 2), ylims=(10, 18)),
    dict(magnitudes=["V", "I"], xlims=(-0.5, 4), ylims=(10, 18)),
    dict(magnitudes=["R", "I"], xlims=(-0.5, 2), ylims=(10, 18))
]

figure_number = 24

for plot_settings in all_plot_settings:
    axes_magnitudes = plot_settings['magnitudes']

    if 'xlims' in plot_settings:
        xlims = plot_settings['xlims']
    else:
        xlims = None

    if 'ylims' in plot_settings:
        ylims = plot_settings['ylims']
    else:
        ylims = None

    blue_mag = axes_magnitudes[0]
    blue_mag_lowcase = blue_mag.lower()
    red_mag = axes_magnitudes[1]
    red_mag_lowcase = red_mag.lower()
    figure_number += 1

    title = (
        f"Figure {figure_number}: Colours and magnitudes "
        "of stars in the direction of NGC 3201 globular cluster."
    )
```

```
    make_cmd(plot_dir="images",␣
↪file_name=f"cmd_{blue_mag_lowcase}_{red_mag_lowcase}.pdf",
             data_path=data_path,
             blue_mag=f"{blue_mag_lowcase}_mag",
             blue_mag_err=f"{blue_mag_lowcase}_mag_err",
             red_mag=f"{red_mag_lowcase}_mag",
             red_mag_err=f"{red_mag_lowcase}_mag_err",
             x_label=f"{blue_mag} - {red_mag} colour index",
             y_label=f"{red_mag} apparent magnitude",
             xlims=xlims, ylims=ylims,
             title=title)
```

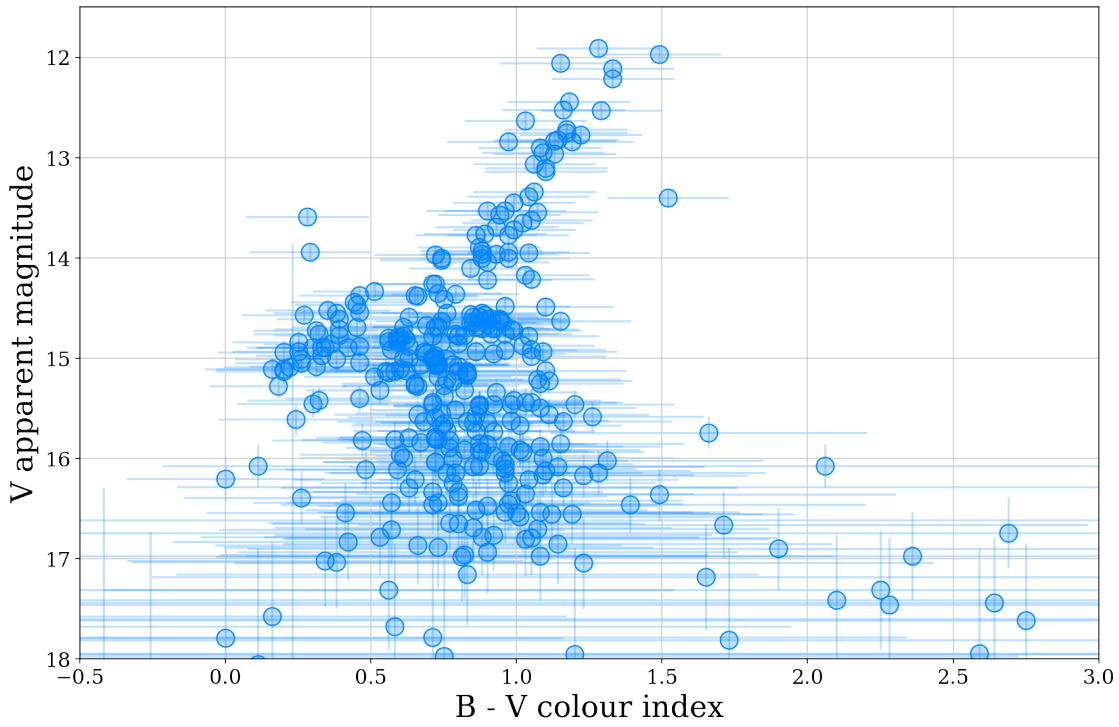


Figure 25: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

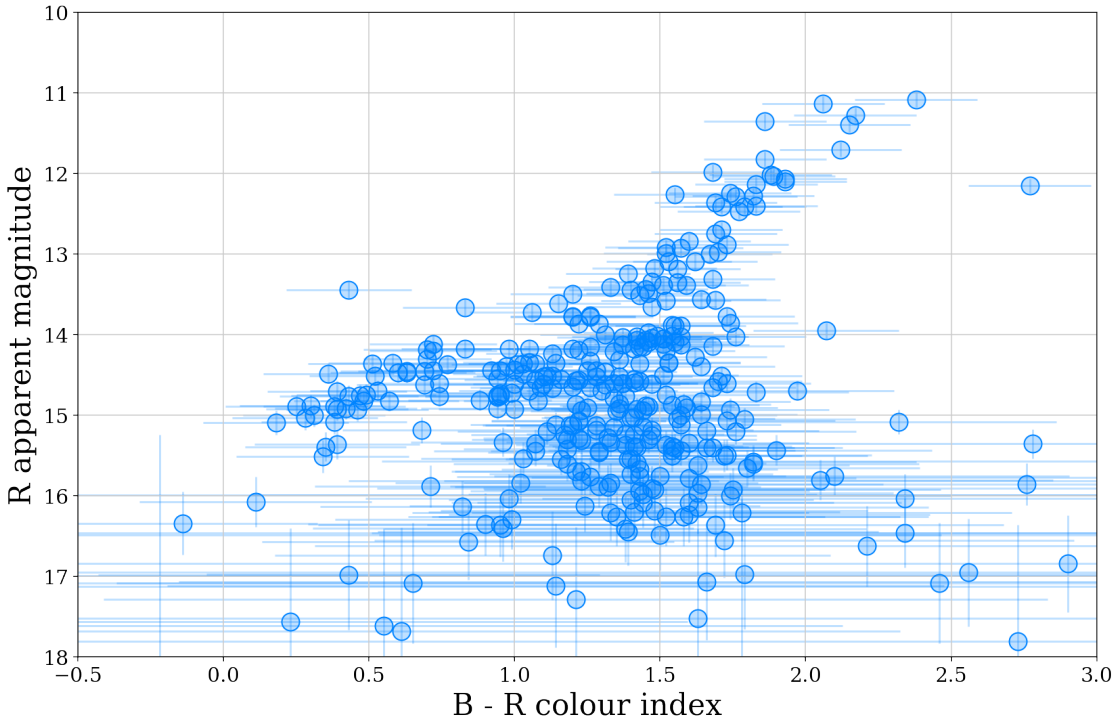


Figure 26: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

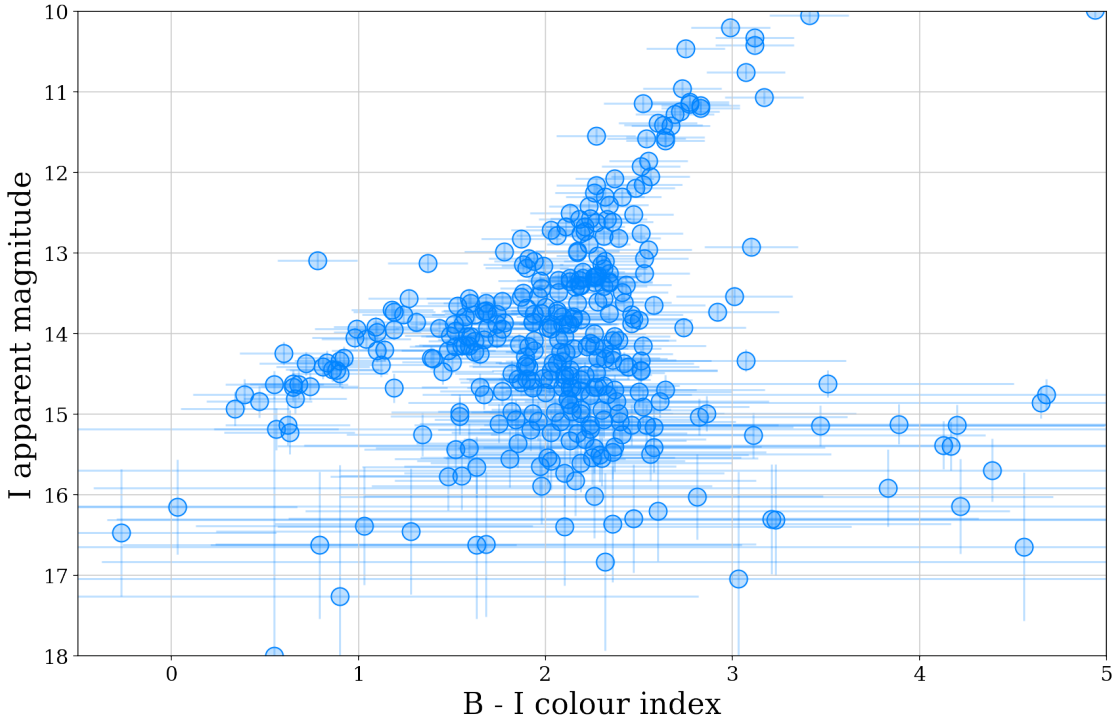


Figure 27: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

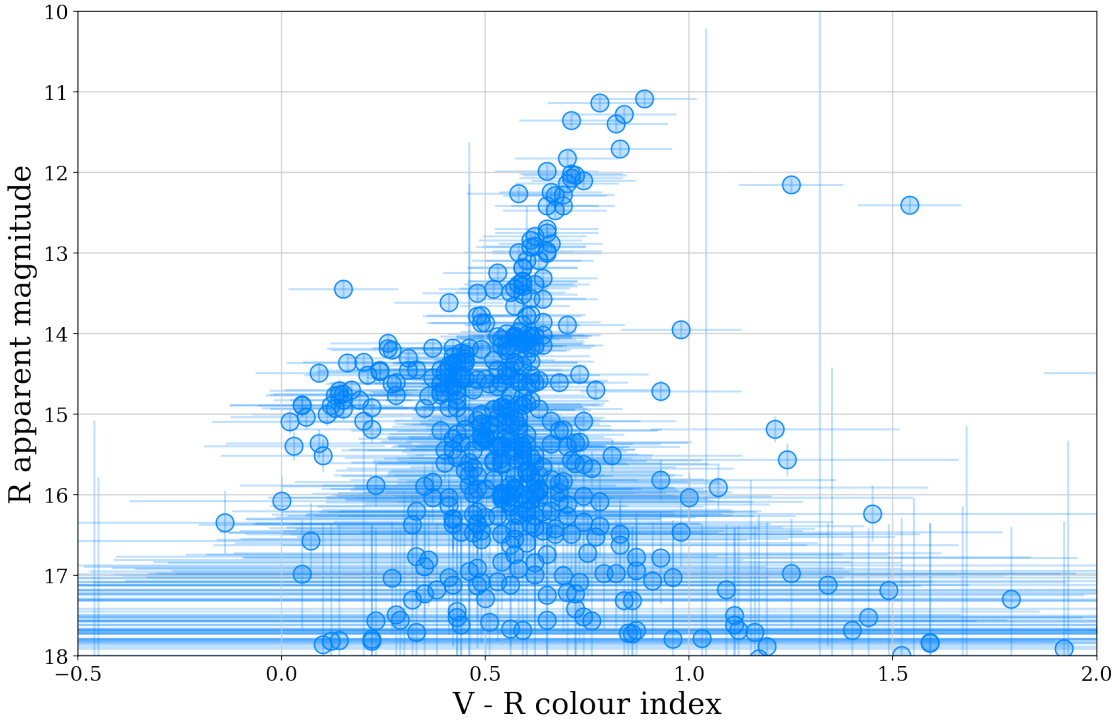


Figure 28: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

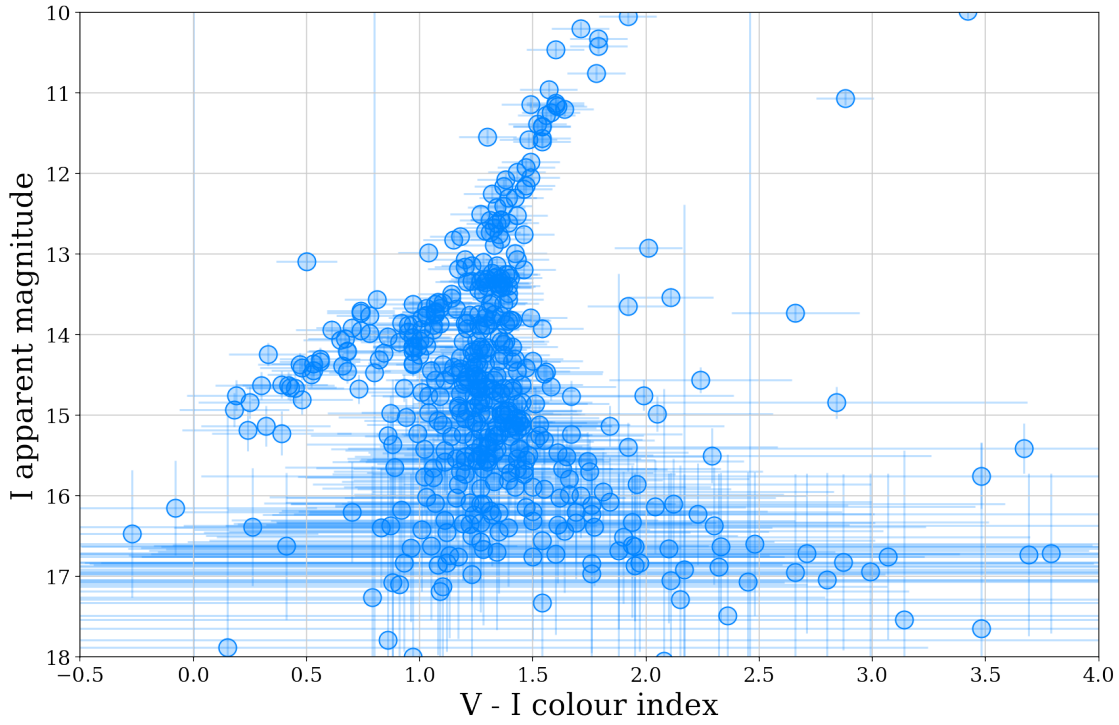


Figure 29: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

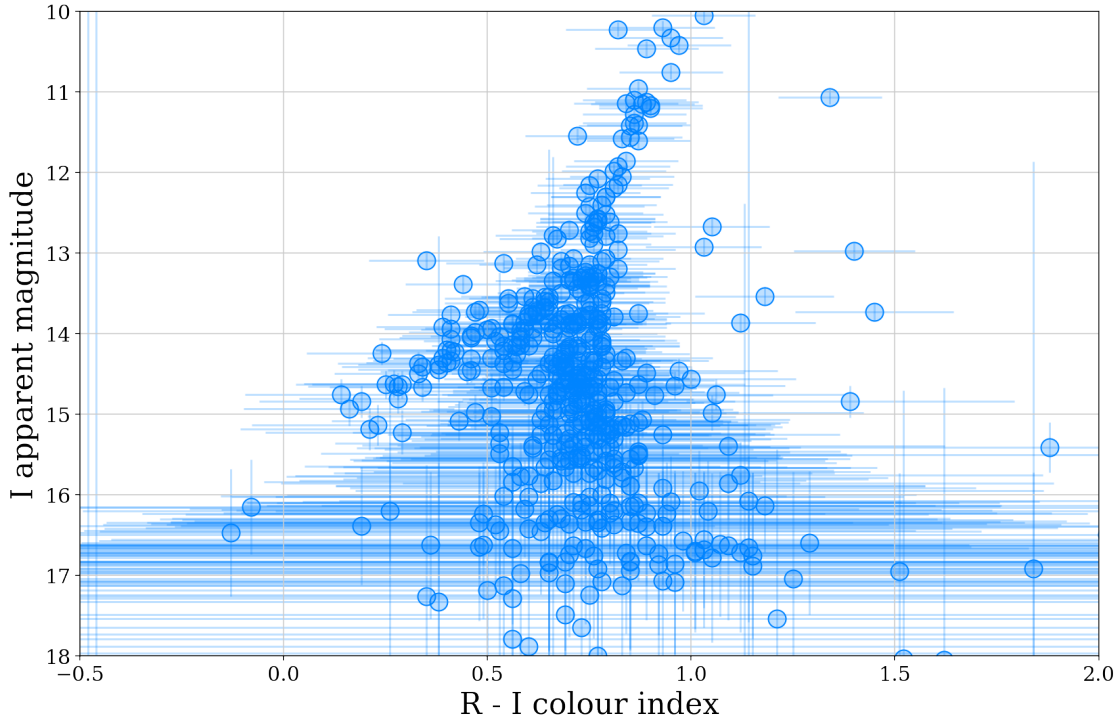


Figure 30: Colours and magnitudes of stars in the direction of NGC 3201 globular cluster.

1.25 Do color-magnitude diagrams make sense?

- We can see on B-V plot (Fig. 27) that V $>$ 16 magnitudes are very uncertain.
- We do not see main sequence, those stars are too faint and we have not detected them.
- We can see horizontal and RGB/AGB branches.

[19]:
```
print("We are done!")
```

```
We are done!
```

[ ]:
```

```