In [3]:
```
from IPython.core.display import HTML
css_file = '../../styles/styles.css'
HTML(open(css_file, "r").read())
```

Out[3]:

Absolute Photometry

Learning Objectives

> - How to use the `photutils` library to detect sources on an image, and measure instrumental magnitudes for many stars
> - Use the `astroquery` and `astropy` librarys to cross match catalogs on sky position
> - How to compute zeropoints and uncertainties

In the lecture we learnt the theory behind precise calibration of photometry, enabling us to put photometric observations onto a standard scale with precisions of better than 1%. In this practical we are going to learn how to produce calibrated photometry using a **simplified method**. This method does not yield the same accuracy as an analysis using observations of primary or secondary standard stars, combined with the use of colour terms. On the other hand, it is much simpler to apply, and even works to produce calibrated photometry through thin cloud. If you are happy to accept accuracies on the order of a few percent, I recommend using the method described below whenever you need to produce calibrated photometry.

In this practical we will will produce a colour-magnitude diagrams using the stacked images you made of your open cluster in the previous practical. If you have not yet finished that practical, use the stacked images provided in the `data` folder, taken from the open cluster NGC 7789. If you have finished, upload your own stacked images to the `data` folder and use them instead.

Method

In the lecture we saw that, for any star, the difference between the calibrated magnitude, m and the above-atmosphere instrumental magnitude $m_{0,i}$ is given by:

$$m = m_{0,i} + m_{\rm zp},$$

where $m_{\rm zp}$ is known as the zero point. The above-atmosphere instrumental magnitude is given by:

$$m_{0,i} = m_i - kX,$$

where $m_i = -2.5 \log_{10}(N_t/t_{\rm exp})$ is the instrumental magnitude, k is the extinction coefficient, and X is the airmass. Therefore:

$$m = m_i - kX + m_{\rm zp}.$$

In other words, if we were to plot the calibrated magnitude m against instrumental magnitude m_i for all the stars in our image, we would expect a straight line, with a gradient of one, and an intercept equal to $-kX + m_{\rm zp}$. **This intercept could then be added to all of our instrumental magnitudes to produce calibrated magnitudes.**

What makes this technique possible, is the existence of large sky surveys, which have provided calibrated magnitudes for many, relatively bright stars over a very wide areas of the sky. If your data is covered by one of these surveys, and it provides calibrated magnitudes in the same filter as your data, you can apply this technique relatively quickly.

Accuracy

The accuracy this technique can achieve is limited by two factors. First of all we are not taking into account any secondary effects, such as 2nd-order extinction, or colour terms. Secondly, our photometry cannot be any more accurate than that in the sky survey we use. Typically, large sky surveys achieve accuracies of a few percent. Ignoring the secondary effects will probably introduce errors on a similar level. If you need calibrated photometry to better than a few percent, you will have to observe standard stars, and apply the rigorous method described in the lecture.

Steps

Applying this method requires the application of four steps. These steps are:

1. finding the stars in our image, and calculating instrumental magnitudes;
2. matching our stars against a sky-survey, so we know instrumental and calibrated magnitude for many stars;
3. calculating the offset between instrumental and calibrated magnitudes, and applying to all stars in our image.

We will carry out these steps in Python. As usual we will draw on several third-party Python libraries. I will explain their use, and provide links to detailed documentation. We will also use some code that I have written which will help ease the code writing.

Photometry with Photutils

We have done aperture photometry before, using AstroImageJ. Whilst this tool is excellent for relative photometry, and producing light curves, each aperture must be placed by hand,

which makes it onerous to use when we want to measure every star in an image.

There are a lot of steps to aperture photometry. We must detect stars in the image, and measure their positions accurately. Then we have to add up the counts within a *target aperture*. Finally we have to measure the sky background level in a *sky annulus*, and subtract off the sky background. You can imagine that writing the functions to do this from scratch is going to be hard. Thankfully, there is a Python library called photutils that is written to do exactly this.

Photutils is not installed by default on CoCalc. The code cell below will install it. Run this cell, and if there are any **errors**, ask for help (warnings are ok).

In [4]:
```
## IMPORTANT: YOU ONLY NEED TO RUN THIS CELL ONCE, BUT RE-RUNNING IS HARMLES
import sys
!{sys.executable} -m pip install photutils
```

```
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: photutils in /home/user/.local/lib/python3.1
0/site-packages (1.9.0)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.10/dis
t-packages (from photutils) (1.23.5)
Requirement already satisfied: astropy>=5.0 in /usr/local/lib/python3.10/di
st-packages (from photutils) (5.3)
Requirement already satisfied: pyerfa>=2.0 in /usr/lib/python3/dist-package
s (from astropy>=5.0->photutils) (2.0.0.1)
Requirement already satisfied: PyYAML>=3.13 in /usr/lib/python3/dist-packag
es (from astropy>=5.0->photutils) (5.4.1)
Requirement already satisfied: packaging>=19.0 in /usr/local/lib/python3.1
0/dist-packages (from astropy>=5.0->photutils) (23.0)
```

The cell above should install photutils into a directory within our project. The directory should be something like `/home/user/.local/lib/python3.10/site-packages`.

If the code cell below raises an error edit the code cell below, so the path in the call to `sys.path.append` is the same one as the install location for photutils. Ask a demonstrator to help you find out where this is.

In [5]:
```
## if this code cell runs without error, you have successfully installed pho
sys.path.append('/home/user/.local/lib/python3.10/site-packages')
import photutils as p
```

Read in your image

Using the code cell below, read in your 300s, V-band stacked image into an array called `data`. Read the header from the FITS file into a variable called `header`. It's important to use those variable names - I'll assume they exist later in the notebook.

You can look at the code from Session 5 about the `astropy.io.fits` library if you can't remember how to do this.

In [6]:
```
from astropy.io import fits

#path to the fits file
image_file = '../../assignments/Session6/data/calib_v300.fit'

# now to read the data and the header from the fit file
data = fits.getdata(image_file) #image data
header = fits.getheader(image_file) # fits header
```

Creating a Source List

Our first job is to detect our sources. We will do this using an algorithm called DAOFIND (Stetson 1987, PASP, 99, 191). DAOFIND looks for bright regions in the image that have a peak brightness greater than some threshold and that have a size and shape similar to a Gaussian of specified FWHM.

Stars in our image will stand out above the background, and DAOFIND will find them, but we need to know what threshold to use. One way of doing this is to measure the statistics of the **background** in our image. If we measure the average value of the background, and the amount the background varies, we can look for regions that are significantly brighter than background pixels.

Below I do that using a "sigma-clipped" mean - where I estimate the average background and the standard deviation. We then throw away all the pixels more than 3 standard deviations (sigma) away from the mean, and repeat the process. I carry on until no pixels are more than 3 standard deviations away from the average value. Then I calculate the mean, median and standard deviation of the remaining pixels. Sound complex? Don't worrry, there's already code to to it!

In [7]:
```
# import sigma_clipping function from astropy
from astropy.stats import sigma_clipped_stats

mean_background, median_background, background_standard_deviation = sigma_cl

print("The background has an average value of {:.1f} and a standard deviatic
    mean_background, background_standard_deviation))
```

```
The background has an average value of 8956.2 and a standard deviation of 5
7.0 counts
```

Finding Stars

Now we know how bright our background is, and how much it varies, let's look for stars that are brighter than the background plus 5 standard deviations. That should be enough that we don't identify bright background pixels as stars by accident. The DAOFIND algorithm needs a guess for how big the stars are - as a Gaussian FWHM - we'll guess at 3 pixels for now.

In [8]:
```
from photutils import DAOStarFinder

# make a star finder object to look for stars with FWHM~3 pixels that are mo
daofind = DAOStarFinder(fwhm=3.0, threshold=5*background_standard_deviation)

# use it to find stars. We'll subtract the background off first, so backgrou
sources = daofind(data - median_background)

print(sources)
```

```
/tmp/ipykernel_408/2966686396.py:1: DeprecationWarning: `photutils.DAOStarF
inder` is a deprecated alias for `photutils.detection.DAOStarFinder` and wi
ll be removed in the future. Instead, please use `from photutils.detection
import DAOStarFinder` to silence this warning.
  from photutils import DAOStarFinder
 id     xcentroid      ...        flux               mag
--- ------------------ ... ------------------ --------------------
  1 104.54137254141901 ... 1.7910758700600866  -0.6327849574346206
  2  627.8937554207942 ... 16.012383135155222  -3.0111399327690656
  3  787.1413866740155 ... 24.187878712599023   -3.458994455546163
  4 155.96555975329753 ...  2.035707823580332  -0.7717886142791887
  5 244.07525649518763 ... 2.9261052390824474   -1.165724854311643
  6  944.0744028847956 ... 1.2694506729315989   -0.259039575372095
  7 455.57477793056165 ...  6.835065504674937   -2.086856702593379
  8  400.8282875821198 ...  1.554770988148444 -0.47916607037177567
  9  606.9901918435214 ... 1.5021301396541191  -0.4417689003875544
 10 247.28877420821863 ... 1.2061944239999436 -0.20354329120358244
...                ... ...                ...                  ...
631  706.2128259167281 ...  2.928102029716752  -1.1664655141036582
632 53.875214186820266 ...   3.73227061683214   -1.429932814331392
633  87.87211856369967 ...  2.371910898428619  -0.9377459264780431
634  700.6631410801001 ... 1.5959055880782134  -0.5075179885432516
635  333.4405535970754 ... 2.5954442139290697   -1.035529246644238
636 1079.5484276189868 ...  2.480683111198473  -0.9864282248203188
637  732.9483176527319 ...  5.284261586841316  -1.8074607708941512
638  893.9291427171465 ... 2.9437299924186555  -1.1722449318094323
639 311.55357784173947 ... 1.9495539087265603  -0.7248381218143364
640  634.3217732162709 ... 1.7875032441580196   -0.630617096698387
641  654.0153859078961 ... 1.1082522012419436  -0.1115965064506049
Length = 641 rows
```

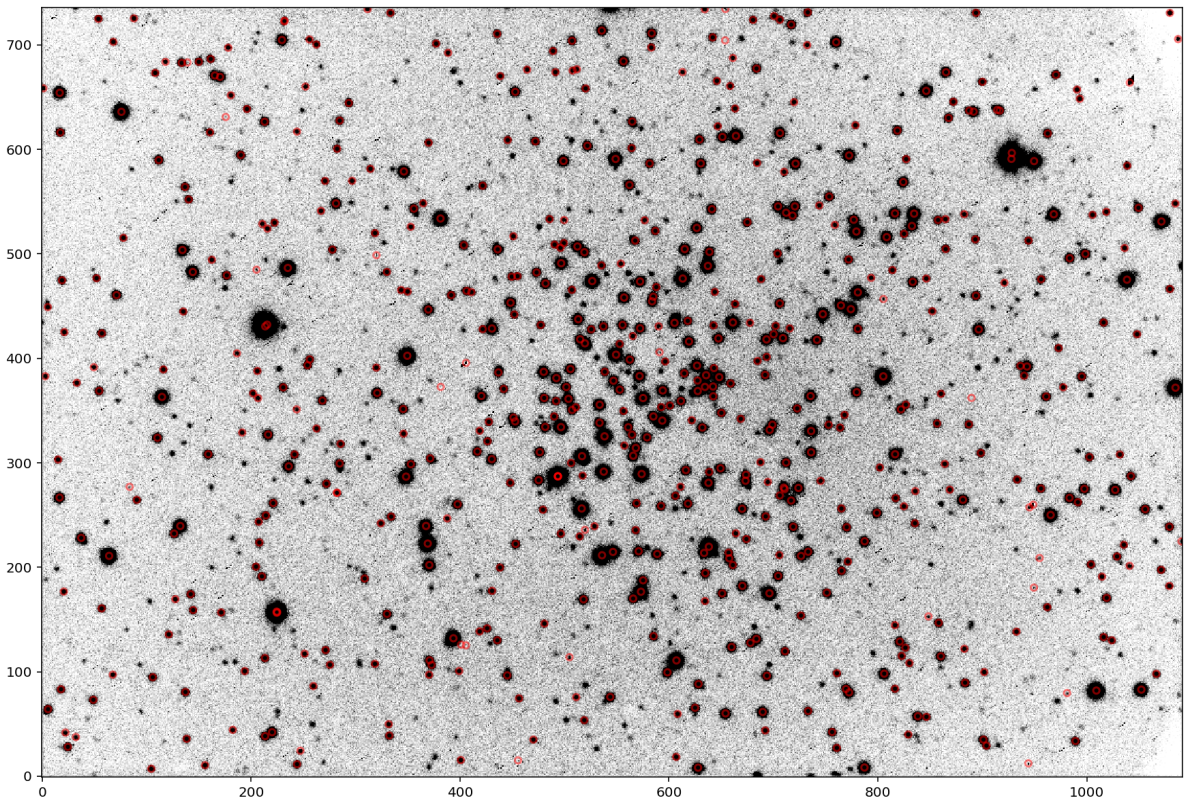
The `sources` variable contains a table of all the detected stars. There are various columns, but the ones we are interested in is the X and Y positions of the stars, which you can find with `sources['xcentroid']` and `sources['ycentroid']`.

But how do we know we've found most of the stars? Or if we are mistakenly identifying bright background pixels as stars? We can inspect our sources by-eye. To make this easier, I've written a little helper function to plot your image with the sources overlaid. The function is in the file `photometry_helpers.py`

In [9]:
```
from photometry_helpers import plot_sources
help(plot_sources)

plot_sources(data, sources, 3)
```

```
Help on function plot_sources in module photometry_helpers:

plot_sources(data, sources, radius)
    Plot positions of detected sources on image.

    Parameters
    ----------
    data:  `np.ndarray`
        A 2D array of pixel values of your data. From a FITS file, you can
create this array with
        `fits.getdata`.

    sources: `~astropy.table.Table`
        A table of detected sources for the image. Usually, `photutils.DAOS
tarFinder` would be
        used to create this list.

    radius: float
        The size of apertures to plot, in pixels
```

Out[9]:



Tweak the detection settings

> The DAOFIND algorithm requires a threshold for star detection, and a typical FWHM of the stars in the images. Try different settings for these values, and see how they affect the detection of stars in your data. Make a decision about what values to use for this image.

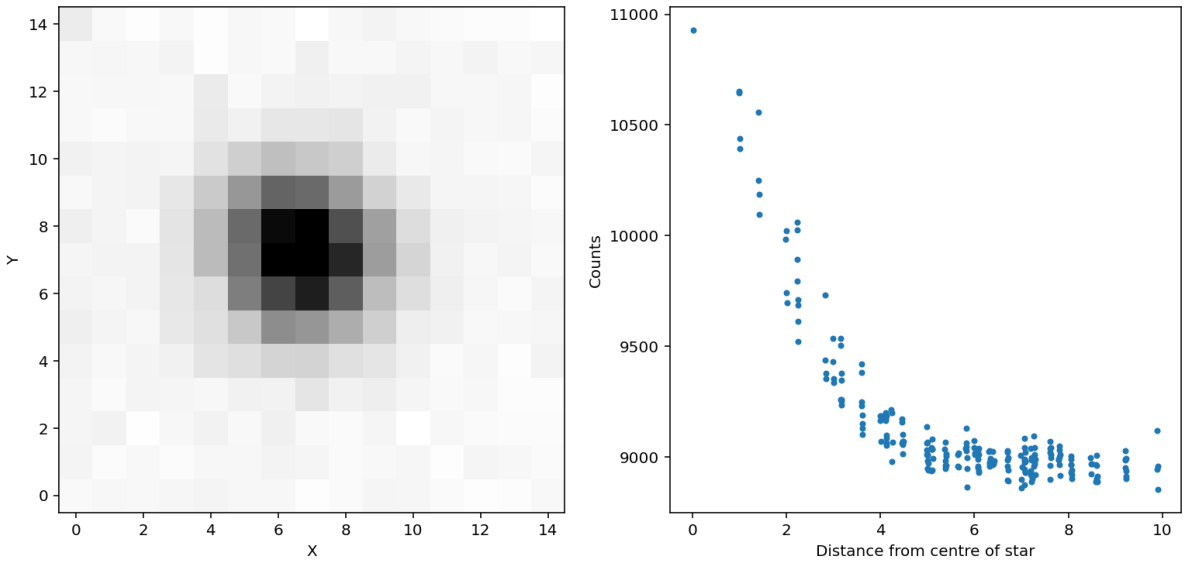
FWHM of stars in the image

We will also need an estimate for the FWHM of stars in the image. We can estimate this by-eye by plotting a bright, isolated, star and plotting the brightness against distance from the star's centre. I've written a clever little routine to do this.

Using the plot created by the code below, estimate the FWHM of the star in pixels. You should be able to convert this to a value in arcseconds, using the information you recorded during observing, and the specifications of the Hicks Observatory

In [10]:
```
from photometry_helpers import measure_FWHM

measure_FWHM(data, sources)
```

Out[10]:



Aperture Photometry

So, we have a list of detected sources and positions in the image, and an idea of the FWHM of stars in the image. Now we can perform aperture photometry on all of these sources. I've written a function to do this for us. The comments in the function make it relatively easy to understand, so by all means take a look in the file `photometry_helpers.py` to see what the function does if you like. In brief the function performs the following steps:

1. Add up the counts from each source within a target aperture

2. Measure the sky brightness around each source using the sky annulus
3. Subtract the sky contribution from the counts in step 1.
4. Calculate instrumental magnitude from the counts and exposure time.

By now you should be getting quite good at reading documentation for functions you import, so let's import my function at look at the documentation.

In [11]:
```
from photometry_helpers import aperture_photometry
help(aperture_photometry)
```

```
Help on function aperture_photometry in module photometry_helpers:

aperture_photometry(data, header, sources, aperture_radius, sky_inner_radiu
s, sky_outer_radius)
    Calculate Aperture Photometry on a list of sources

    Parameters
    ----------
    data:  `np.ndarray`
        A 2D array of pixel values of your data. From a FITS file, you can
create this array with
        `fits.getdata`.

    header: `~astropy.fits.Header`
        A FITS Header object. From a FITS file, you can create this object
with
        `fits.getheader`.

    sources: `~astropy.table.Table`
        A table of detected sources for the image. Usually, `photutils.DAOS
tarFinder` would be
        used to create this list.

    aperture_radius: float
        Radius of the target aperture, in pixels

    sky_inner_radius: float
        Radius of the inner aperture that makes up the sky annulus

    sky_outer_radius: float
        Radius of the outer aperture that makes up the sky annulus

    Returns
    -------
    phot_table: `~astropy.table.Table`
        A table of measurements for each source, including instrumental mag
nitude and error.
```

Perform Aperture Photometry

Using the help above, use this function to perform aperture photometry on all of your sources. You'll need to pick values for the aperture radii, remembering everything we've

discussed in the course about how these choices affect the quality of your measurement.

Target apertures want to be big enough to accept a decent fraction of the flux, but not so large that the measurements are very noisy, or contaminated by nearby stars. As a rule of thumb this aperture might have a radius of 1.5-2x the FWHM.

Sky Annuli want to be wide enough to accurately measure the sky, but not so large that the annuli overlap nearby stars.

To get started, try values around 5, 10, 20 pixels for these apertures. We will tweak them later on.

In [12]:
```
# YOUR CODE HERE. USE THE FUNCTION ABOVE TO PERFORM APERTURE PHOTOMETRY

# Define aperture and sky annulus radii
aperture_radius = 5.0  # Adjust based on your source characteristics
sky_inner_radius = 10.0
sky_outer_radius = 20.0

# Perform aperture photometry
V300 = aperture_photometry(data, header, sources, aperture_radius, sky_inner
```

Photometric Calibration

Now we have a table of instrumental magnitudes (and much besides) in it. The next step is photometric calibration. As a reminder, this involves fitting a straight line to a graph of *instrumental magnitude* against *calibrated magnitude*, to find the offset between the two. Equivalently, we can find the average value of the *difference* between the calibrated and instrumental magnitudes for all our stars.

Since we have instrumental magnitudes and sky positions (RA, Dec) for a number of stars, we must find the matching stars in an online catalog of calibrated magnitudes. We will use the APASS catalog; a catalog which combines several other sky surveys to provide data in many filters across much of the sky. Crucially, in this case it includes B and V magnitudes, the two filters used for our photometry.

To perform the cross-matching we will use the astroquery Python library, and we have our aperture photometry results in a Table object from astropy.

In [13]:
```
!pip install astropy-regions
```

```
Defaulting to user installation because normal site-packages is not writeab
le
ERROR: Could not find a version that satisfies the requirement astropy-regi
ons (from versions: none)
ERROR: No matching distribution found for astropy-regions
```

In [14]:
```
from astropy import units as u
from astroquery.xmatch import XMatch
```

```
Could not import regions, which is required for some of the functionalities
of this module.
```

Our photometry is stored as an astropy table, which is a handy object for reading and writing tabular data. These astropy tables play nicely with Jupyter notebooks, so you can simply type the name of the table in a code cell to see the table displayed in the browser. So, if you named your photometry table above `V300` the following cell will work. If not, replace the variable name below:

In [15]:
```
V300
```

Out[15]: *QTable length=641*

id	xcenter	ycenter	aperture_sum	aperture_sum_err	
	pix	pix			
int64	float64	float64	float64	float64	
1	104.54137254141901	7.088233791098361	718407.4367414076	972.2342140967972	5990.
2	627.8937554207942	8.187394898555173	876422.5868504382	1072.687771286411	6276.6
3	787.1413866740155	8.32544979285852	978944.7054532459	1133.1097934279899	6309.5
4	155.96555975329753	10.538952644506498	719774.8007327804	973.1479539021806	6927.
5	244.07525649518763	11.476120301044267	733632.1475929314	982.3601633013358	7258.
6	944.0744028847956	12.284980533424804	701494.7025427758	960.8604547012317	7469.
7	455.57477793056165	15.136791152241742	694206.7931449371	955.9176384704814	8958
8	400.8282875821198	15.438920799360066	716610.641298095	971.0321998109993	8957.
9	606.9901918435214	18.461009501868357	718474.863808099	972.2792923043772	8955.
...	...	...	...	...	
632	53.875214186820266	724.8500390557613	730051.7953618774	979.988281925749	6808.
633	87.87211856369967	725.3452452543411	718576.8677250567	972.347482812128	6632.
634	700.6631410801001	727.5462706037954	726344.7498430387	977.5264056351915	6150.
635	333.4405535970754	730.6097091293623	725755.9726257467	977.123571037232	5412.
636	1079.5484276189868	730.5618590201254	709904.9337402543	966.5282146175795	4107.
637	732.9483176527319	731.1780458837513	733338.4200201904	981.9967584657919	5339.
638	893.9291427171465	730.6673492598428	728450.7209626881	978.9046324861737	5396.
639	311.55357784173947	734.223529581349	472786.13099707314	788.7298431454681	471
640	634.3217732162709	734.0744593495212	486112.51122627826	799.7660450411163	4734.
641	654.0153859078961	734.3435500941304	456244.5317113177	774.8697020717477	4686.

The important columns for our uses are the measured centres of our stars in RA and Dec (**RA** and **DEC**) and the instrumental magnitude and uncertainty (**instrumental_mag** and **e_instrumental_mag**). Note that the magnitude uncertainty is calculated using the CCD signal-to-noise equation we saw in the lectures.

X-match with APASS

> We need to calibrate our photometry, which will involve comparing our instrumental magnitudes to calibrated magnitudes measured for the same stars. We need to match our detected stars with those catalogued in the sky survey APASS. We are looking for stars who's RA and Dec matches to within some radius. A service called **Vizier** hosts online versions of astronomical catalogs, and we can use the `Xmatch.query` function to match an astropy table with a table hosted by Vizier using the code below.

> Run the code cell below, and note carefully how we specify the columns that contain RA and Dec in our *local* table, and how we set the maximum distance for a valid match. **II/336/apass9** is the name of the APASS catalog on Vizier. If you need to find the names of other catalogs (perhaps APASS doesn't cover the patch of sky containing your open cluster), you can enter the catalog name in the search box here

> **The code cell below may take a while to run. Be patient...**

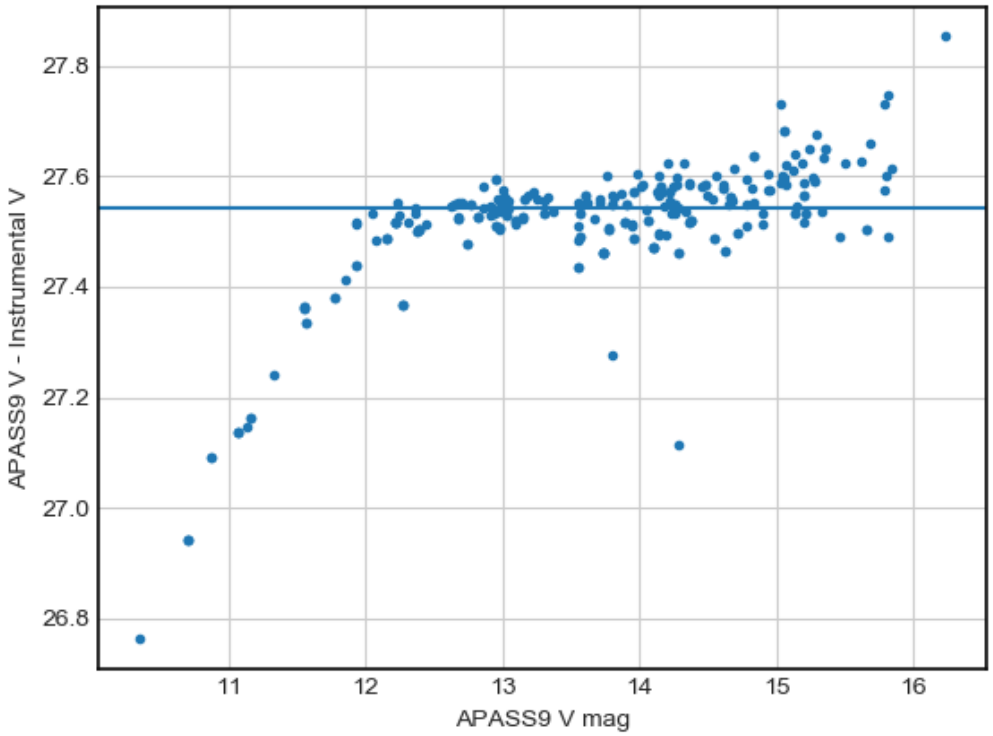
In [16]:
```
xmatch = XMatch.query(cat1=V300, cat2='vizier:II/336/apass9', max_distance=2
```

The result of this query (`xmatch`) is also an astropy `Table` . It has all the columns from both tables for each valid match. If you want to extract a column and save it into a variable, you can access the `Table` like a dictionary. So the code below extracts the **Magnitude** column (which is instrumental magnitude from our APT photometry) and **Vmag** column from UCAC4 and computes the difference.

In [17]:
```
delta_mag = xmatch['Vmag'] - xmatch['instrumental_mag']
```

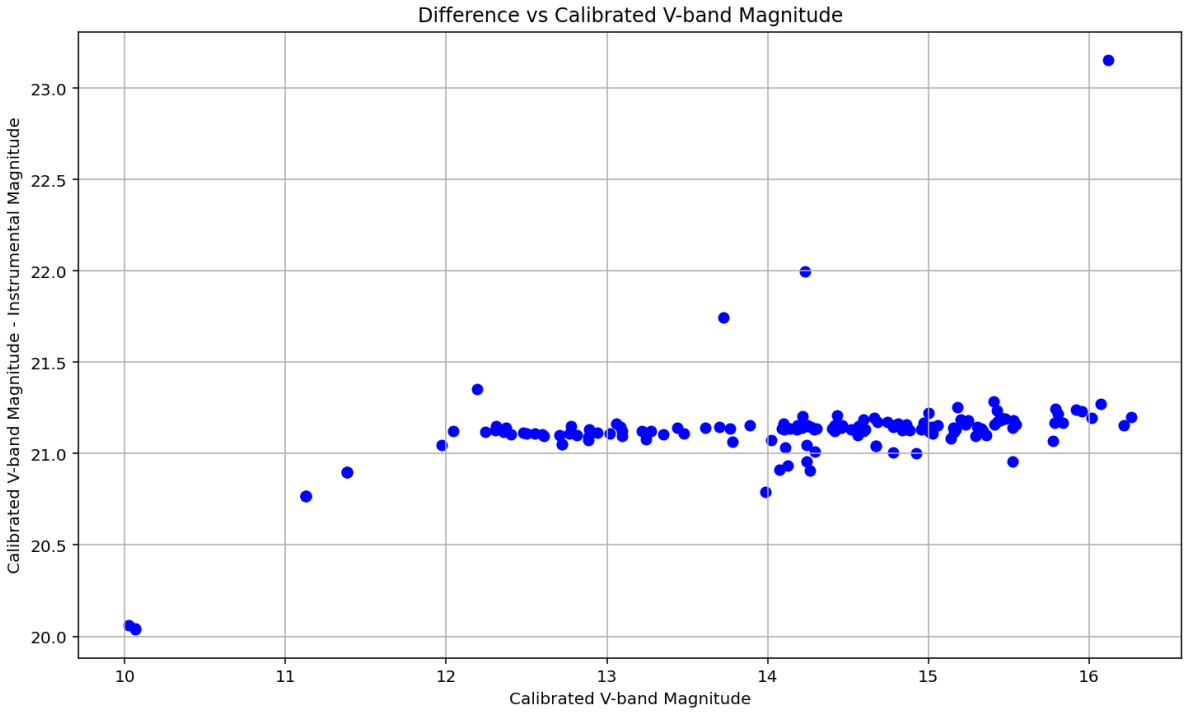
Plot the difference and find the zero-point

> Plot the difference between instrumental and calibrated magnitude found above, against the calibrated V-band magnitude on the X-axis. You should see something like the figure below:



In [18]:
```
# YOUR CODE HERE - MAKE A PLOT LIKE THE ONE ABOVE. DON'T WORRY ABOUT ADDING
import matplotlib.pyplot as plt

# Assuming you have a column 'Vmag' in your xmatch table
calibrated_mag = xmatch['Vmag']

# Assuming you have a column 'instrumental_mag' in your xmatch table
instrumental_mag = xmatch['instrumental_mag']

# Calculate the difference between instrumental and calibrated magnitude
mag_difference = calibrated_mag - instrumental_mag

# Plot the difference against calibrated V-band magnitude
plt.scatter(calibrated_mag, mag_difference, marker='o', color='blue')
plt.xlabel('Calibrated V-band Magnitude')
plt.ylabel('Calibrated V-band Magnitude - Instrumental Magnitude')
plt.title('Difference vs Calibrated V-band Magnitude')
plt.grid(True)
plt.show()
```

Out[18]:



The magnitude difference between instrumental and calibrated magnitude *should* be a constant, which is the value of $kX + m_{\rm zp}$. In the figure above, you can see there are plenty of outlying points, and the bright stars deviate from the constant. The outliers are either stars whose photometry is bad (poor sky estimation, or contaminated by very close stars), or spurious detections (i.e not stars). The deviation of the bright stars is caused because they are saturated, and so we cannot accurately measure their flux.

Since $m = m_i - kX + m_{\rm zp}$, we can find the value of $-kX + m_{\rm zp}$. - which I'll call the *zeropoint* from now on - by calculating the **median** difference between the instrumental and calibrated magnitude. The median will be robust against the outliers - but we only want to do it for the non-saturated stars!

Calculate the zero-point

> Calculate the median value of `delta_mag`. **If you have evidence for saturated stars**, use a NumPy *mask* to only calculate the median of stars that are not saturated (see notes on Fancy slicing in the NumPy notebook).

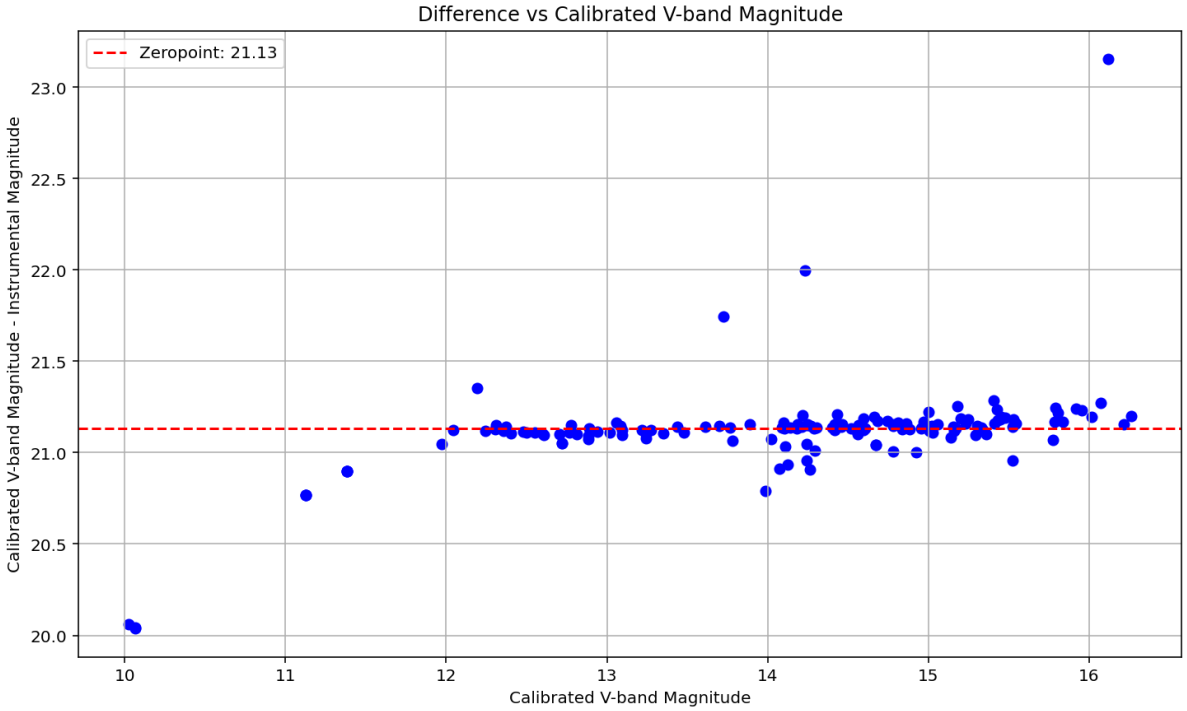
> The median value of `delta_mag` is our estimate of $-kX + m_{\rm zp}$ - i.e the value we want to add to our instrumental magnitudes to get a calibrated magnitude. You can add this value to the **instrumental_mag** column of the V300 table easily using `V300['calibrated_mag'] = V300['instrumental_mag'] + zp`, where `zp` is the median value you found.

In [19]:
```
# YOUR CODE HERE. CALCULATE THE ZEROPOINT AND ADD IT TO THE MAGNITUDE COLUMN
import numpy as np

# Calculate the median value of delta_mag
zp = np.ma.median(mag_difference)

# Add the median value to the instrumental_mag column in the V300 table
V300['calibrated_mag'] = V300['instrumental_mag'] + zp

# Print the calculated zeropoint
print("Zeropoint (zp):", zp)
# Print the first few rows of the V300 table
print(V300[['instrumental_mag', 'calibrated_mag']][:5])
```

```
Zeropoint (zp): 21.134499368410953
  instrumental_mag    calibrated_mag
------------------- ------------------
-7.2928864032276906 13.841612965183263
-7.7664948302471135  13.36800453816384
 -8.017944769062373  13.11655459934858
 -6.918995850258788 14.215503518152165
 -6.841189441735343  14.29330992667561
```

In [20]:
```
# Plot the difference against calibrated V-band magnitude
plt.scatter(calibrated_mag, mag_difference, marker='o', color='blue')
plt.xlabel('Calibrated V-band Magnitude')
plt.axhline(zp, color='red', linestyle='--', label=f'Zeropoint: {zp:.2f}')
plt.ylabel('Calibrated V-band Magnitude - Instrumental Magnitude')
plt.title('Difference vs Calibrated V-band Magnitude')
plt.grid(True)
plt.legend()
plt.show()
```

Out[20]:



In [21]:
```
V300
```

Out[21]: *QTable length=641*

id	xcenter	ycenter	aperture_sum	aperture_sum_err	
	pix	pix			
int64	float64	float64	float64	float64	
1	104.54137254141901	7.088233791098361	718407.4367414076	972.2342140967972	5990.
2	627.8937554207942	8.187394898555173	876422.5868504382	1072.687771286411	6276.6
3	787.1413866740155	8.32544979285852	978944.7054532459	1133.1097934279899	6309.5
4	155.96555975329753	10.538952644506498	719774.8007327804	973.1479539021806	6927.
5	244.07525649518763	11.476120301044267	733632.1475929314	982.3601633013358	7258.
6	944.0744028847956	12.284980533424804	701494.7025427758	960.8604547012317	7469.
7	455.57477793056165	15.136791152241742	694206.7931449371	955.9176384704814	8958
8	400.8282875821198	15.438920799360066	716610.641298095	971.0321998109993	8957.
9	606.9901918435214	18.461009501868357	718474.863808099	972.2792923043772	8955.
...	...	...	...	...	
632	53.875214186820266	724.8500390557613	730051.7953618774	979.988281925749	6808.
633	87.87211856369967	725.3452452543411	718576.8677250567	972.347482812128	6632.
634	700.6631410801001	727.5462706037954	726344.7498430387	977.5264056351915	6150.
635	333.4405535970754	730.6097091293623	725755.9726257467	977.123571037232	5412.
636	1079.5484276189868	730.5618590201254	709904.9337402543	966.5282146175795	4107.
637	732.9483176527319	731.1780458837513	733338.4200201904	981.9967584657919	5339.
638	893.9291427171465	730.6673492598428	728450.7209626881	978.9046324861737	5396.
639	311.55357784173947	734.223529581349	472786.13099707314	788.7298431454681	471
640	634.3217732162709	734.0744593495212	486112.51122627826	799.7660450411163	4734.
641	654.0153859078961	734.3435500941304	456244.5317113177	774.8697020717477	4686.

Homework #6

The B-band data and making a CMD

> Your task in the homework is to repeat the steps above for the rest of your data, and make two CMDs, one for the 30s data, and one for the 300s data.

The B-band data (4 points)

> Repeat the steps above for the 300-second B-band data. Make a plot of the difference between instrumental B-band magnitude and APASS B-band magnitude. Calculate the offset needed to correct your instrumental mags (zeropoint) and display it on your plot.

> Finally, add your zeropoint to the **instrumental_mag** column of the 300s B-band table to make a new **calibrated_mag** column.

In [22]:
```
from astropy.io import fits

#path to the fits file
image_file = '../../assignments/Session6/data/calib_b300.fit'

# now to read the data and the header from the fit file
data = fits.getdata(image_file) #image data
header = fits.getheader(image_file) # fits header
```

In [23]:
```
# import sigma_clipping function from astropy
from astropy.stats import sigma_clipped_stats

mean_background, median_background, background_standard_deviation = sigma_cl

print("The background has an average value of {:.1f} and a standard deviatic
    mean_background, background_standard_deviation))
```

```
The background has an average value of 3165.5 and a standard deviation of 2
8.2 counts
```

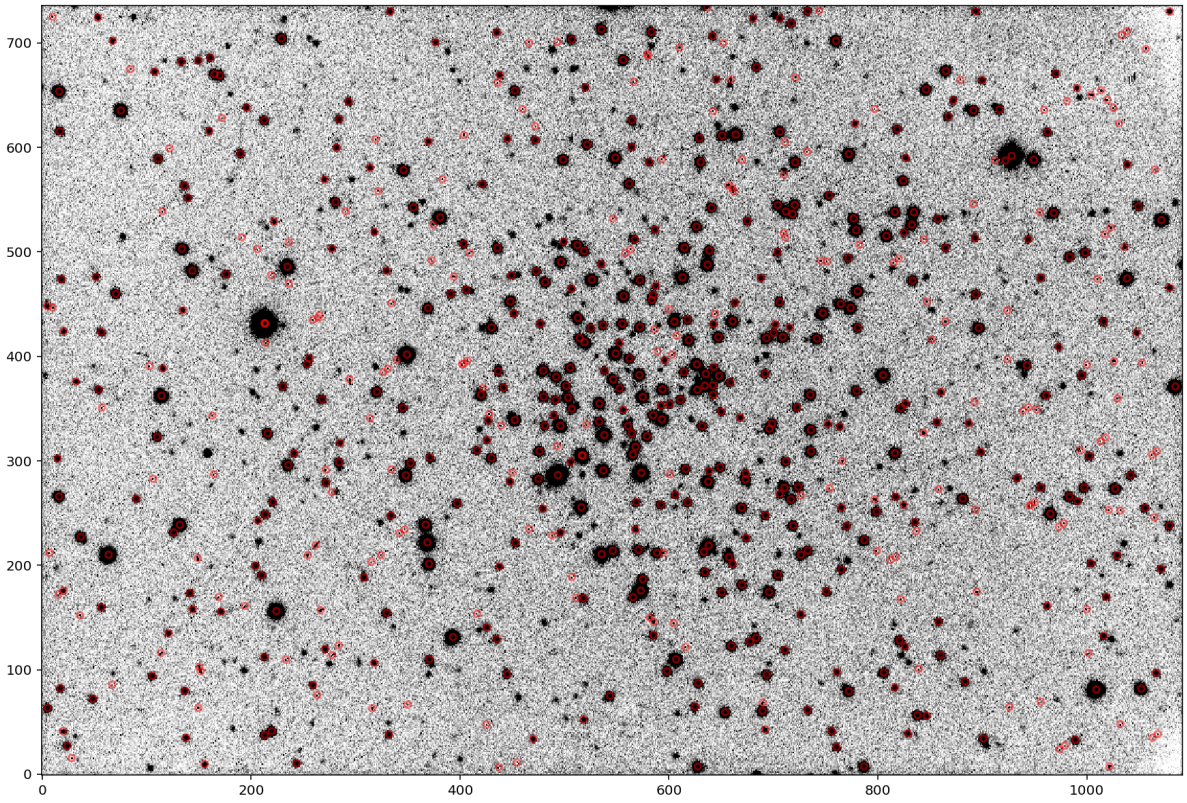
In [24]:
```
from photutils import DAOStarFinder

# make a star finder object to look for stars with FWHM~3 pixels that are mo
daofind = DAOStarFinder(fwhm=3.0, threshold=5*background_standard_deviation)

# use it to find stars. We'll subtract the background off first, so backgrou
sources = daofind(data - median_background)

print(sources)
```

```
 id     xcentroid      ...        flux                mag
--- ------------------ ... ------------------ --------------------
  1  437.2023544594674 ...  5.594184353903613  -1.8693419356410623
  2  627.6582618076158 ...  8.807985540313128   -2.362191482504784
  3 1021.7400269306551 ...  7.338545518220818   -2.164024980851261
  4  786.8464846588936 ...  7.422032919722416  -2.1763071907224543
  5  155.5138854438645 ... 1.0434352291185944 -0.04616373889555292
  6 243.72336234917987 ... 1.7288312309294163  -0.5943814984342017
  7  454.2887156801631 ...  8.536916844124084    -2.32825262773307
  8 28.492926284863707 ...  1.474130502135332 -0.42133983134565656
  9   974.049693949068 ... 1.8881304631101856  -0.6900799980133332
 10  760.4627628034586 ... 2.2523320762671775   -0.881581054544974
...                ... ...                ...                 ...
647  717.0102105054492 ...  3.888980257744378   -1.474589346185415
648  680.3922867623619 ...  1.382889328649818 -0.35196856328865167
649  706.0601353351024 ... 1.2749076298241868  -0.2636968005310938
650 10.146565593479595 ... 11.730927871322638    -2.67333091124753
651 53.587814765542916 ... 1.2474438261752152  -0.2400524948940544
652 1079.5065922176536 ... 1.1354878052438653  -0.1379561860112771
653 333.21143651529326 ... 1.1772047187158658  -0.1771299856504721
654  743.9761271052761 ... 1.5875492217727027  -0.5018179985523625
655   893.822829050948 ... 1.3295934912306333 -0.30929720110020453
656  732.7957169175983 ...  3.072372679405474    -1.21868473640043
657  636.6757110345097 ... 2.5601822992364904   -1.020677226496097
Length = 657 rows
```

```
/tmp/ipykernel_408/2966686396.py:1: DeprecationWarning: `photutils.DAOStarF
inder` is a deprecated alias for `photutils.detection.DAOStarFinder` and wi
ll be removed in the future. Instead, please use `from photutils.detection
import DAOStarFinder` to silence this warning.
  from photutils import DAOStarFinder
```

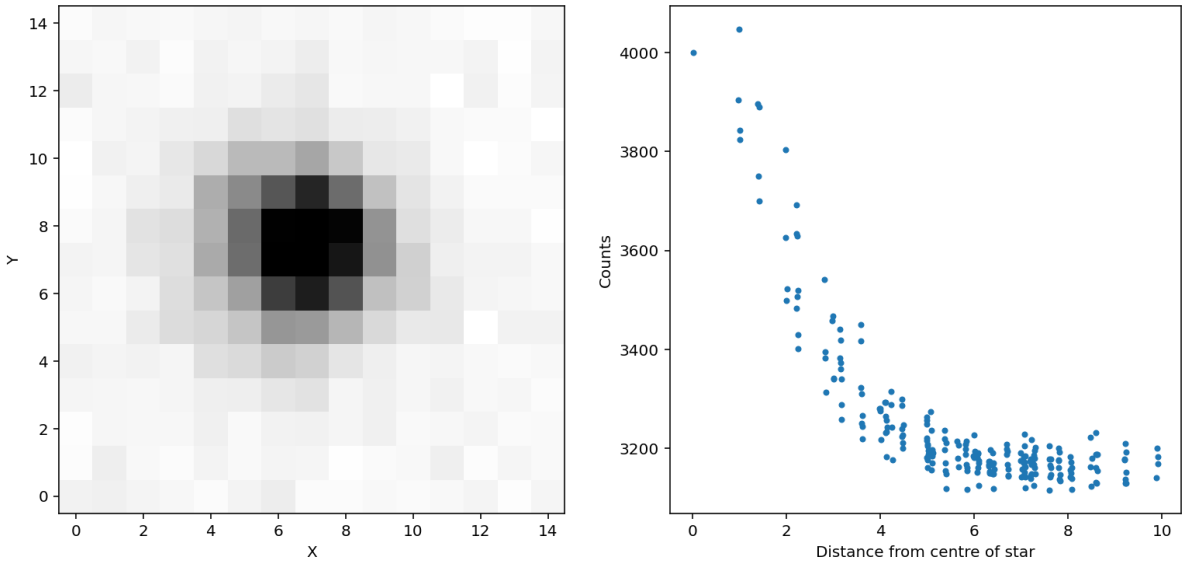
In [25]:
```
from photometry_helpers import plot_sources
plot_sources(data, sources, 3)
```

Out[25]:



In [26]:
```
from photometry_helpers import measure_FWHM
measure_FWHM(data, sources)
```

Out[26]:



In [27]:
```
from photometry_helpers import aperture_photometry
```

In [28]:
```
# Define aperture and sky annulus radii
aperture_radius = 4.5  # Adjust based on your source characteristics
sky_inner_radius = 9.0
sky_outer_radius = 18.0

# To perform aperture photometry for B filter
B300 = aperture_photometry(data, header, sources, aperture_radius, sky_inner
```

In [29]:
```
from astropy import units as u
from astroquery.xmatch import XMatch
```

In [30]:
```
B300
```

Out[30]: *QTable length=657*

id	xcenter	ycenter	aperture_sum	aperture_sum_err	
	pix	pix			
int64	float64	float64	float64	float64	
1	437.2023544594674	6.834275018761437	205633.9898128773	525.8184678472368	2166.9
2	627.6582618076158	7.315209203754148	250706.5066731625	578.8603725410483	2219.
3	1021.7400269306551	7.480199371272065	204978.71717603315	525.0078158775962	2224.4
4	786.8464846588936	7.43028562143857	243192.83261274872	570.3608988860502	2219.5
5	155.5138854438645	9.616138770626668	204993.83727670644	525.0265353891348	2456.86
6	243.72336234917987	10.643295093299365	209880.28170981462	531.0416559863785	2591.5
7	454.2887156801631	11.175953992973724	195656.3398259504	513.3362607706396	2680.3
8	28.492926284863707	15.394023994519426	200946.97499795345	519.9922519550737	3158.0
9	974.049693949068	23.928359431731415	202400.25888852827	521.8057215877938	3159.7
...	...	...	...	...	
648	680.3922867623619	723.3125938047142	207446.14901021493	528.0538495261104	2725.6
649	706.0601353351024	723.4904663177546	209526.3373925693	530.6082479543769	2730.84
650	10.146565593479595	724.8484949355263	207105.94638154935	527.6349160151373	1973.71
651	53.587814765542916	724.092009683259	206416.9779402018	526.7854835294929	2623.7
652	1079.5065922176536	729.8886748347428	204325.22013368885	524.1981119850371	1629.0
653	333.21143651529326	729.9794751482737	207684.9656229389	528.3477354777793	2011.2
654	743.9761271052761	730.9302356817799	203050.75332566674	522.6153976099635	1952.38
655	893.822829050948	730.2198775492471	208785.56146731038	529.7000133374548	1998.11
656	732.7957169175983	730.5709977954796	217172.98128959668	539.8942021347597	1968.59
657	636.6757110345097	734.1995644658568	140985.87159021574	435.343165399437	1680.25

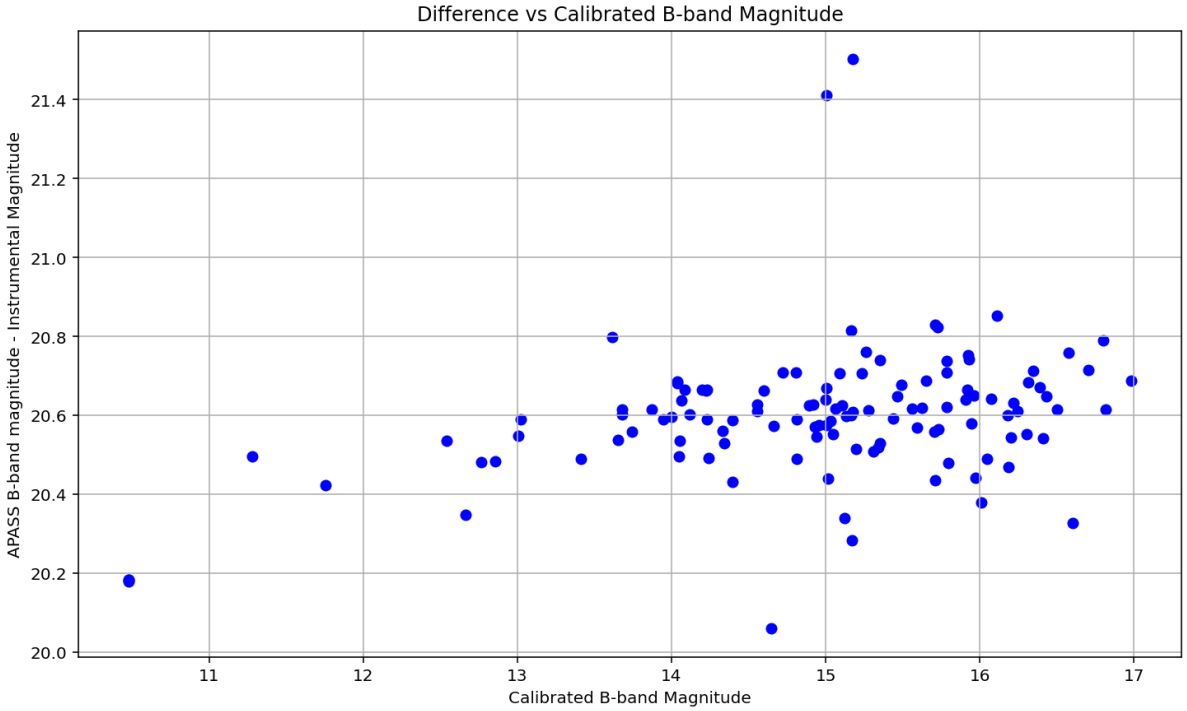
In [31]:
```
xmatch = XMatch.query(cat1=B300, cat2='vizier:II/336/apass9', max_distance=2
```

In [32]:
```
delta_mag = xmatch['Bmag'] - xmatch['instrumental_mag']
```

In [33]:
```
import matplotlib.pyplot as plt

# Assume having a column 'Bmag' in your xmatch table
calibrated_mag = xmatch['Bmag']

# Assume having a column 'instrumental_mag' in your xmatch table
instrumental_mag = xmatch['instrumental_mag']
```

```
# Calculate the difference between instrumental and calibrated magnitude
mag_difference = calibrated_mag - instrumental_mag

# Plot the difference against calibrated B-band magnitude
plt.scatter(calibrated_mag, mag_difference, marker='o', color='blue')
plt.xlabel('Calibrated B-band Magnitude')
plt.ylabel('APASS B-band magnitude - Instrumental Magnitude')
plt.title('Difference vs Calibrated B-band Magnitude')
plt.grid(True)
plt.show()
```

Out[33]:



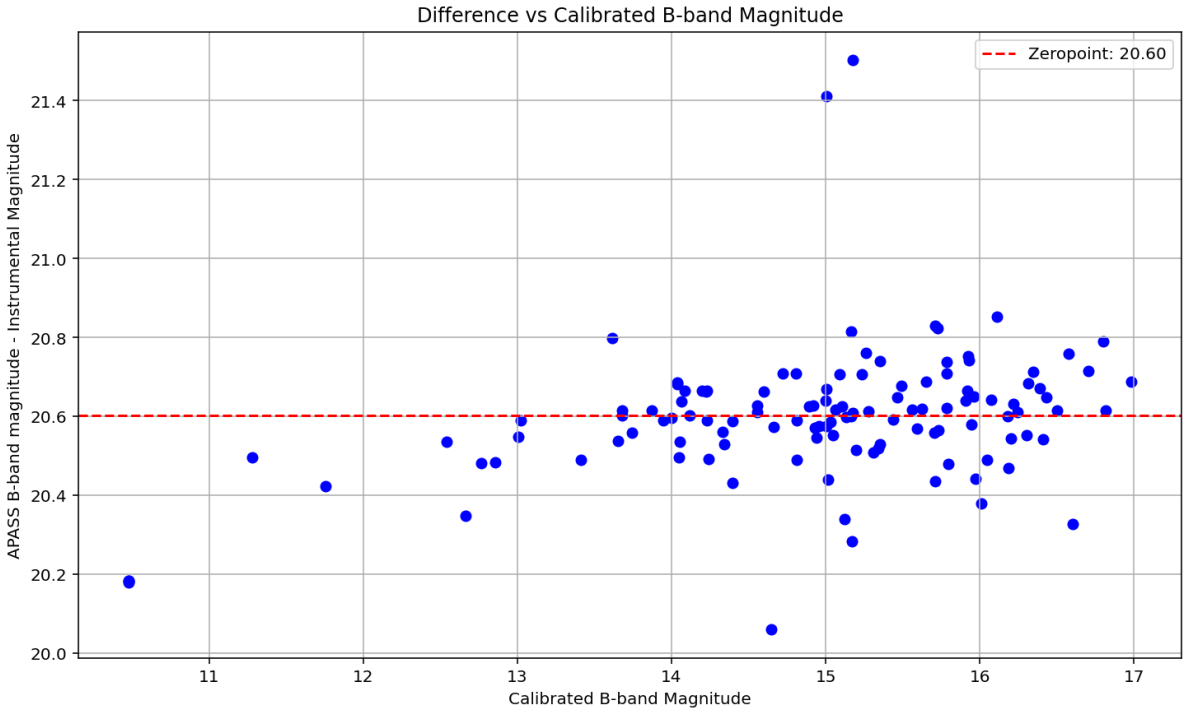
In [34]:

```
import numpy as np

# Calculate the zeropoint
zp = np.ma.median(mag_difference)

# Add the median value to the instrumental_mag column in the B300 table
B300['calibrated_mag'] = B300['instrumental_mag'] + zp

# Print the calculated zeropoint
print("Zeropoint (zp):", zp)

# Print the first few rows of the B300 table
print(B300[['instrumental_mag', 'calibrated_mag']][:5])
```

```
Zeropoint (zp): 20.602750127070266
 instrumental_mag    calibrated_mag
------------------ ------------------
-5.884904705870908 14.717845421199357
-6.405585828546734 14.197164298523532
 -5.81353019784269 14.789219929227576
-6.328576521708818 14.274173605361447
-5.525909315436362 15.076840811633904
```

In [35]:
```
# Plot the difference against calibrated V-band magnitude
plt.scatter(calibrated_mag, mag_difference, marker='o', color='blue')
plt.xlabel('Calibrated B-band Magnitude')
plt.axhline(zp, color='red', linestyle='--', label=f'Zeropoint: {zp:.2f}')
plt.ylabel('APASS B-band magnitude - Instrumental Magnitude')
plt.title('Difference vs Calibrated B-band Magnitude')
plt.grid(True)
plt.legend()
plt.show()
```

Out[35]:



In [36]:
```
B300
```

Out[36]: *QTable length=657*

id	xcenter	ycenter	aperture_sum	aperture_sum_err	
	pix	pix			
int64	float64	float64	float64	float64	
1	437.2023544594674	6.834275018761437	205633.9898128773	525.8184678472368	2166.9
2	627.6582618076158	7.315209203754148	250706.5066731625	578.8603725410483	2219.
3	1021.7400269306551	7.480199371272065	204978.71717603315	525.0078158775962	2224.4
4	786.8464846588936	7.43028562143857	243192.83261274872	570.3608988860502	2219.5
5	155.5138854438645	9.616138770626668	204993.83727670644	525.0265353891348	2456.86
6	243.72336234917987	10.643295093299365	209880.28170981462	531.0416559863785	2591.5
7	454.2887156801631	11.175953992973724	195656.3398259504	513.3362607706396	2680.3
8	28.492926284863707	15.394023994519426	200946.97499795345	519.9922519550737	3158.0
9	974.049693949068	23.928359431731415	202400.25888852827	521.8057215877938	3159.7
...	...	...	...	...	
648	680.3922867623619	723.3125938047142	207446.14901021493	528.0538495261104	2725.6
649	706.0601353351024	723.4904663177546	209526.3373925693	530.6082479543769	2730.84
650	10.146565593479595	724.8484949355263	207105.94638154935	527.6349160151373	1973.71
651	53.587814765542916	724.092009683259	206416.9779402018	526.7854835294929	2623.7
652	1079.5065922176536	729.8886748347428	204325.22013368885	524.1981119850371	1629.0
653	333.21143651529326	729.9794751482737	207684.9656229389	528.3477354777793	2011.2
654	743.9761271052761	730.9302356817799	203050.75332566674	522.6153976099635	1952.38
655	893.822829050948	730.2198775492471	208785.56146731038	529.7000133374548	1998.11
656	732.7957169175983	730.5709977954796	217172.98128959668	539.8942021347597	1968.59
657	636.6757110345097	734.1995644658568	140985.87159021574	435.343165399437	1680.25

Now you should have a two astropy tables in computer memory. One with corrected (calibrated) V-band magnitudes and one with corrected (calibrated) B-band magnitudes. We need to cross-match these tables with each other, to find which stars in the V-band table match with stars in the B-band table. We can't use `astroquery`'s Xmatch for this, since they are both local tables. Instead, we will use the `SkyCoord` object from astropy, which is meant to work with coordinates on the sky. You can create a `SkyCoord` object with positions of all of the stars in a table like so:

```
from astropy.coordinates import SkyCoord

coo_v = SkyCoord(V300['RA'], V300['DEC'], unit=u.deg)
```

The `SkyCoord` you made is a bit like a NumPy array, but it is an array of positions on the sky. It carries within it several useful functions, including one to match positions against another sky coordinate object - `match_coordinates_sky`. It's usage is shown below,

and the documentation is here. `match_coordinates_sky` returns an array of indices and the 2D and 3D separations of the matches. The array of indices can be used as a slice to sort the second set of coordinates, or the table from which it came, so that each row of the sorted coordinate/table contains the closest match of the corresponding row. Perhaps an example will explain:

```
# match every entry in coo_v with the nearest entry in coo_v
idx, distance_2d, distance_3d = coo_v.match_to_catalog_sky(coo_b)

# using idx as a slice for the table B300 will sort it so that
B300[0] is the closest match to V300[0]
B300 = B300[idx]
```

We are not quite done yet though - we have found the closest match to every object in the V300 table, but some of those matches may be quite far away. We could create a *fancy slicing mask* of True and False values by comparing the `distance_2d` array to some threshold separation, and use that mask to remove the rows where there is no close cross-match:

```
mask = distance_2d < 3 * u.arcsec
B300 = B300[mask]
V300 = V300[mask]
```

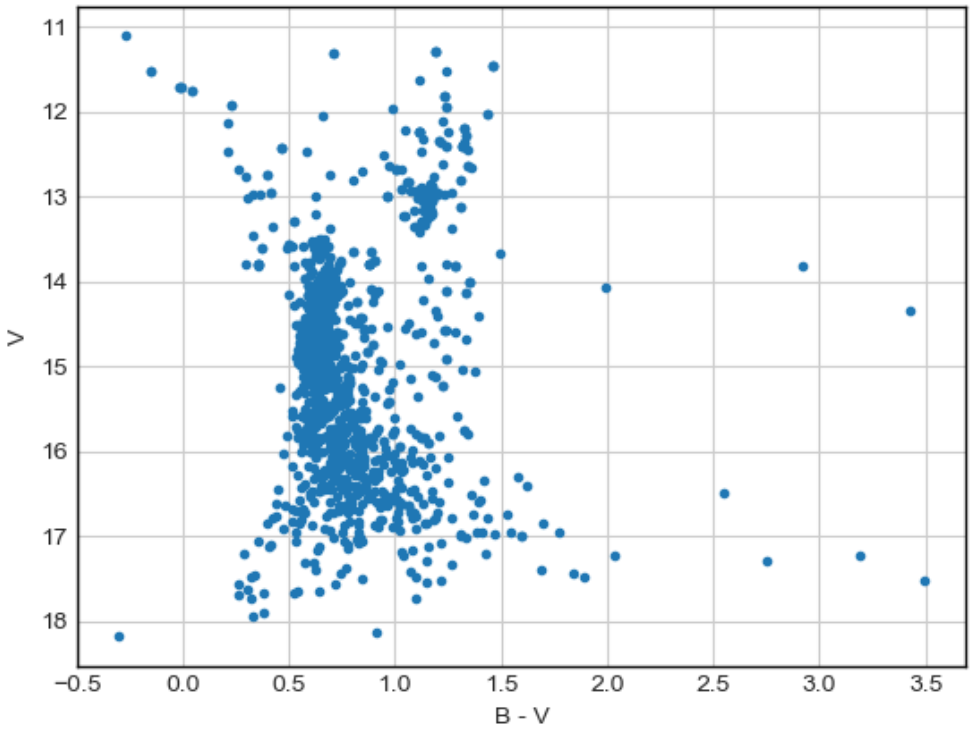
Cross-matching your V and B-band tables (2 points)

> Create `SkyCoord` objects from your 300s B and V tables. Match them using `match_coordinates_sky`, and use a mask to remove all rows where there is no good match.

In [37]:
```
# YOUR CODE HERE

from astropy.coordinates import SkyCoord

# Assume having columns 'RA' and 'DEC' in your B300 and V300 tables
coo_b = SkyCoord(B300['RA'], B300['DEC'], unit=u.deg)
coo_v = SkyCoord(V300['RA'], V300['DEC'], unit=u.deg)

# match every entry in coo_v with the nearest entry
idx, distance_2d, distance_3d = coo_v.match_to_catalog_sky(coo_b)

# using idx as a slice for the table B300 will sort it so that B300[0] is th
B300 = B300[idx]

# Create a mask based on the matching radius
mask = distance_2d  < 3 * u.arcsec
B300 = B300[mask]
V300 = V300[mask]
```

Plotting your CMD (3 points)

> With your matched tables in hand, extract the calibrated magnitude columns from each into two arrays called `V` and `B`. Calculate `B-V` and plot a colour-magnitude diagram of `B-V` against `V`. Your plot should look something like the one below, which is for the cluster NGC 7789.



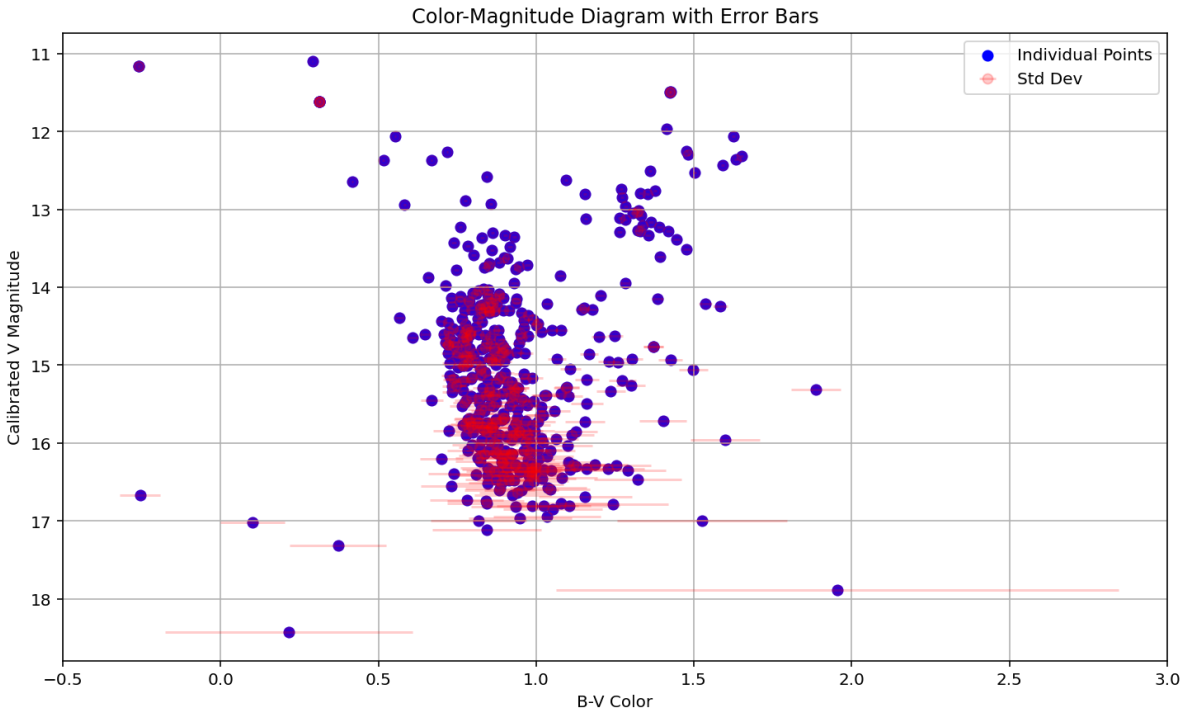
> Also calculate error bars on `B-V`: remember that
> $\Delta(x+y)^2 = \Delta x^2 + \Delta y^2$

> Hint: you can reverse the y-axis with `plt.gca().invert_yaxis()`

In [38]:
```
import matplotlib.pyplot as plt

# Assuming having columns 'calibrated_mag' in both B300 and V300 tables
V = V300['calibrated_mag']
B = B300['calibrated_mag']

# Assuming having columns 'e_instrumental_mag' in both B300 and V300 tables
error_V = V300['e_instrumental_mag']
error_B = B300['e_instrumental_mag']

# Calculate B-V color and its error using the provided formula
B_V = B - V
error_B_V = (error_V**2 + error_B**2)**0.5

# Plot the color-magnitude diagram
plt.errorbar(B_V, V, xerr=error_B_V, fmt='o', color='red', alpha=0.2, label=
```

```
plt.scatter(B_V, V, marker='o', color='blue', alpha=1.0, label='Individual P
plt.xlabel('B-V Color')
plt.ylabel('Calibrated V Magnitude')
plt.title('Color-Magnitude Diagram with Error Bars')
plt.xlim(-0.5, 3.0)
plt.gca().invert_yaxis()  # Reverse the y-axis
plt.legend()
plt.grid(True)
plt.show()
```

Out[38]:



Save your data (1 point)

> Save your `V` and `B-V` data, along with their uncertainties, to a text file, for use in the next lab!

> You might want to revisit Session1 for instructions for writng data to a file. Or, you could create an astropy Table using the instructions here and here

In [39]:

```
import numpy as np

# Combine data into a numpy array
data = np.column_stack((V, B_V, error_B_V))

# Save data to a text file
np.savetxt('color_magnitude_data.txt', data, header="V_mag B-V_color error_B
```