```python
import numpy as np
from astropy.io import fits
from astropy.table import Table
from astropy.wcs import WCS
#from findobjects_solution import findObjects
from findobjects import findObjects
from astropy.coordinates import SkyCoord
import matplotlib.pyplot as plt
import warnings
import glob
warnings.filterwarnings('ignore')


import os
home = os.getenv('HOME')

# This is the only thing that needs to switch depending on if you're working at the lab
# or on your own computer
onedrive = "C:\\Users\\aka188\\OneDrive - Simon Fraser University (1sfu)"

# Directory that has your output from Assignment 5
data_dir = onedrive + "/PHYS391-Spring-2025-private/Assignment5"

# This is your private directory.  You should save this python script there too.
working_dir = onedrive + '/PHYS391-Spring-2025-private/Assignment5'

#------------------------------------------------------------------------------------------
gain = 0.3360284075579739 # What you found in Assignment 2
platescale = 0.17082740088105727 # What you found in Assignment 1
## arxsec / pixel , plate scale for this binning from platescale proejct

bands = ['PhotB','PhotV','PhotR'] # We only need B and V today

# Coordinates of reference stars that I found for you in the image
# TYC 2410-1181-1, NGC 2099 391, NGC 2099 134, UCAC4 613-027641
ra = [88.06561167, 88.022623, 88.0629883,88.0730483]
dec = [32.5031286, 32.4756283, 32.5093561, 32.46334139]
coords_ref = SkyCoord(ra, dec , unit='deg', frame='fk5')
mag_ref_dict = {'PhotB': [11.99,12.51,11.82,12.183],
                'PhotV': [11.31,10.93,11.28,11.826],
                'PhotR': [11.81,10.413,11.93,11.747]}

#------------------------------------------------------------------------------------------
def getmags(band):
    # This function calls findObjects() that you wrote in Assignment 4 (and I modified)
    # and then computers magnitudes from the output fluxes

    file_list = glob.glob(os.path.join(working_dir, f"M37_{band}.fit*"))
    # Open the combined image that you created in align.py
    hdu = fits.open(file_list[0])
    image = hdu[1].data # Drizzle saves the data in the 2nd hdu item, ie hdu[1] not hdu[0]

    # For these items check align.py to find the header cards to use
    texp = hdu[0].header["EXPINDIV"] # exposure time of an individual image
    nexp = hdu[0].header["NDRIZIM"] # number of images that were combined ## ???
    skymean = hdu[0].header['SKYMEAN'] # the mean value of the sky
```

```
binning = hdu[0].header['BINNING'] # the binning

# Run findObjects(), using the propper binning, an edge_dist of 300.  Do not
# restrict the objects to non-blended sources, and make sure to use the sky, texp
# and nexp keywords
objects,_ = findObjects(image,binning=binning,
                        edge_dist=300,F=5,B=512,brightest=None,
                        nonblended=False, sky=None,texp=texp , nexp=nexp)

# Get the resulting "flux_auto" and error values
flux = objects['flux_auto']
fluxerr = objects['flux_auto_err']

# Compute the signal-to-noise ratio
sn = flux / fluxerr # Signal to Noise Ratio = FLux / FLux error

# Now compute the instrumental magnitudes and magnitude error
mag_inst = -2.5*np.log10(flux/texp)
mag_err =  (2.5 / np.log(10)) * (fluxerr / flux) # Use standard error propagation and fluxer

# get WCS of each object
w = WCS(hdu[0].header)
coords = w.pixel_to_world(objects['xwin_image'], objects['ywin_image'])

# Find matches to reference stars
idx, d2d, d3d = coords_ref.match_to_catalog_sky(coords)
mag_ref = np.array(mag_ref_dict[band]) # use mag_ref_dict and the correct key ### ???

# Compute the offset between the real mags and the instrument mags for the reference
# stars.  Remember idx is the list of indices that give you the matching stars in
# coords, and thus mag_inst
offset = mag_ref - mag_inst[idx]  # Use reference mag minus instrument mag
meanoffset = np.mean(offset) # the mean offset

print(f"{band}'s Offset {offset}, expected to be 23 for all bands")

# Now using the meanoffset, correct all the instrumental mags to real mags.
mag = mag_inst + meanoffset

# Compute the mean surface brightness of the sky for this band using the mean value
# of the sky and angular size of the binned pixel (from the platescale and binning)
# Compute the mean surface brightness of the sky using the correct formula
mu_sky_inst = -2.5 * np.log10((skymean * gain) / texp)  # Convert sky flux to instrumental m
mu_sky = mu_sky_inst + 2.5 * np.log10((platescale * binning) ** 2) \
    + meanoffset  # Convert per pixel to per arcsec²


# Add the magnitudes and RA and DEC as extra columns in the objects table
objects['ra'] = coords.ra
objects['dec'] = coords.dec
objects[f'mag_inst_{band}'] = mag_inst
objects[f'mag_err_{band}'] = mag_err
objects[f'mag_{band}'] = mag
objects[f'S/N_{band}'] = sn
objects[f'npixels_{band}'] = np.pi*(2*objects['flux_radius'])**2
```

```
    print(f"Number of stars found in band {band}: {len(objects)}")
    print(f"Mean surface brightness of the sky for {band}: {mu_sky}")

    return objects


#--------------------------------------------------------------------------------------
# Now that we've defined the getmags function above, run it on for PhotB and PhotV to
# find the stars in those combined images
objectsB = getmags('PhotB')
objectsV = getmags('PhotV')
objectsR = getmags('PhotR')

# Note: you can use the magnitudes and S/N values, which are obtained after 510 seconds,
# to estimate how much time you need to expose your target for your project.
# Likewise, you can use the npixels parameter to estimate the "surface brightness" of
# the stars and plot that against S/Nt (after 510 seconds) if you want to know how long
# you need to expose an extended object (like a galaxy) in order to achieve the S/N that
# you want.

#--------------------------------------------------------------------------------------
# Now we have to match the stars from objectsV to objectsB (which had fewer stars).
# Let's define a final "master" astropy table that only contains the stars from the
# PhotB catalogue and fill it with data from objectsB
masterTable = Table(objectsB)

# Let's get the WCS of the PhotB file (which should be same for PhotV)
hdu = fits.open(working_dir+ "/M37_PhotB.fit")
wcs = WCS(hdu[0].header)

# Define the world coordinates of each object table
coordsB =  wcs.pixel_to_world(masterTable['xwin_image'], masterTable['ywin_image'])
coordsV =  wcs.pixel_to_world(objectsV['xwin_image'], objectsV['ywin_image'])

# For the B catalogue find the star in the V catalogue that is the closest match
idx, d2d, _ =  coordsB.match_to_catalog_sky(coordsV)

print(f"Matched {len(idx)} stars, median separation: {np.median(d2d.arcsec):.2f} arcsec")


# Get the columns from objectV and assign them to the masterTable
band = 'PhotV'
masterTable[f'mag_inst_{band}'] = objectsV[f'mag_inst_{band}'][idx]
masterTable[f'mag_err_{band}'] = objectsV[f'mag_err_{band}'][idx]
masterTable[f'mag_{band}'] = objectsV[f'mag_{band}'][idx]
masterTable[f'S/N_{band}'] = objectsV[f'S/N_{band}'][idx]
masterTable[f'npixels_{band}'] = objectsV[f'npixels_{band}'][idx]
masterTable[f'match_sep_{band}'] = d2d.arcsec

# You could do the same for PhotR if you wanted to, but we won't need it for this
# Assignment.

# Let's clean the table of stars where the closest match in the V catalogue was more
# than 3 arcsec away
mask = (masterTable['match_sep_PhotV'] < 3)
masterTable = masterTable[mask]
```

```
# Now save the table to a fits file (yes, fits files can contain tables as well as
# images)
masterTable.write('masterTable.fits',overwrite=True)

#---------------------------------------------------------------------------------------
# Let's make a quick plot of B-V on the x-axis and V and on the y-axis.
# Make sure "bright" is at the top of the plot.  Check my plot in Lecture 6 to make sure
# yours looks similar.  No need to upload this one.  ac

print("PhotB min/max:", masterTable['mag_PhotB'].min(), masterTable['mag_PhotB'].max())
print("PhotV min/max:", masterTable['mag_PhotV'].min(), masterTable['mag_PhotV'].max())


# Define B-V and V from the quantities in the masterTable
BminusV = masterTable['mag_PhotB'] - masterTable['mag_PhotV']
V = masterTable['mag_PhotV']

# Make the plot and label the axes
plt.plot(BminusV,V,'o')
plt.xlabel("BminusV")
plt.ylabel("V")
plt.gca().invert_yaxis()


plt.figure()
plt.hist(BminusV, bins=50, color='orange', alpha=0.7)
plt.xlabel("B-V")
plt.ylabel("Frequency")
plt.ylim(0, max(np.histogram(BminusV, bins=50)[0]) + 50)  # Prevents flipping
plt.title("B-V Histogram (Uninverted)")

plt.ylim([16.5, 7.5])
plt.xlim([0, 2.2])
plt.show()
'''
# Debugg
print(len(BminusV), len(V))  # Should be the same!
print(f"B-V range: {BminusV.min()} to {BminusV.max()}")
print(f"V magnitude range: {V.min()} to {V.max()}")
print(f"CoordsB min/max RA: {coordsB.ra.min()}, {coordsB.ra.max()}")
print(f"CoordsB min/max DEC: {coordsB.dec.min()}, {coordsB.dec.max()}")
print(f"Mag_PhotB min/max: {masterTable['mag_PhotB'].min()} to {masterTable['mag_PhotB'].max()}"
print(f"Mag_PhotV min/max: {masterTable['mag_PhotV'].min()} to {masterTable['mag_PhotV'].max()}"
print(f"BminusV min/max: {BminusV.min()} to {BminusV.max()}")
print(f"V min/max: {V.min()} to {V.max()}")

print("PhotB min/max:", masterTable['mag_PhotB'].min(), masterTable['mag_PhotB'].max())
print("PhotV min/max:", masterTable['mag_PhotV'].min(), masterTable['mag_PhotV'].max())
print("BminusV min/max:", BminusV.min(), BminusV.max())
'''
```