Documentation

Below is a rough documentaiton of the code. I have only described the files in the folders that i think are necessary to run the code. I have left out the files in folders which i think are not really necessary for the code to work.

Folder: Arrays

The folder contains all the possible ALMA configurations that can be used in the code, but its not necessary as the python code lego_alma_eso_sid.py generates its own custom config file everytime the configuration of the lego-telescope is changed. The custom config file is stored as **lego_alma.config** in the folder **arrays_old**.

Folder: arrays_old

The folder contains the custom config file created for our use as **lego_alma.config.**

Folder: image

The folder contains the images we use for lego alma.

Folder: Imports

The folder contains the most important python files **vriCalc.py**. This is the actual python file that needs to be run to calculate the interferometer images. This python file takes in the image, antenna configuration, etc. More details when called.

CSV File: sid_ant_text_latest.csv

contains the antenna coordinates.

lego_alma_eso_sid.py

In the section below, I will be explaining the code in the python file lego_alma_eso_sid.py. This python file is how you interact with **vriCalc.py** .

Importing packages and setting parameter values

```
import serial
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import pandas as pd
from astropy.io import ascii
import time
from Imports.vriCalc import observationManager
from astropy.convolution import Gaussian2DKernel,convolve
from scipy.ndimage import gaussian_filter
import matplotlib.image as mpimg
import pickle
import os
import sys
import pyautogui
import matplotlib.pylab as pylab
import sys
```

In the code snippet above, I am importing all the necessary packages needed to run the code.

```
# antenna_filename = "/home/kiosk/alma_sid_07_05_2025/Alma_main/sid_ant_tex
antenna_filename = "sid_ant_text.csv"
```

```
# model_logo_filename = "/home/kiosk/alma_sid_07_05_2025/Alma_main/models
model_logo_filename = "./models/aifa-logo.png"

imagefile1 = "./image/agb_star.jpg"
imagefile2 = "./image/galaxy_gas.jpg"
imagefile3 = "./image/hltau.jpg"
imagefile4 = "./image/outflow.jpg"
```

The code snippet above is there to import all the antenna positions stored in the variable **antenna_filename.** The code also imports all the necessary images for the plotting the images from the interferometer.

```
### below sets all the parameters for use in plottting
LOOP_TIME = 0.1 #seconds.  ## time between each new plot
scale_array =40.0 ## scaling the distance of antenna position on board to real lif
FREQ = 3e5 #MHz. # frequency of the observation
DEC = -40 #declination. # DEC position of the observation
pixel_scale = 0.0055 #arcseconds #pixel scaling from image pixel to arcseconds


params = {'axes.titlesize':'small'}. # setting the titlesize of the image
pylab.rcParams.update(params)
colormap = 'inferno'. # sets the colorscale


## dictionary that connects the bitwise position of the buttons to its use
bitdict = {'hourangle_m6':     1,
           'hourangle_0':      2,
           'hourangle_p6':     3,
           'agb_star':      6,
           'galaxy_gas':       5,
           'hltau':       4,
           'outflow':   7}
```

```
## dividing the above dictionary into buttons that select position on the sky and t
bitdict_config = {'hourangle_m6':     1,
          'hourangle_0':      2,
          'hourangle_p6':     3}


bitdict_image = {'agb_star': 3,
          'galaxy_gas': 1,
          'hltau': 2,
          'outflow': 4}


#### some more parameters for plotting

#this is a parameter, for debugging the code when you don't have ardruino input
# if set to False means ardruino is connected and vice-versa
NoSerial = False.
webcam = False # parameter to show if webcam connected should always we Fa
verbose = True # parameter to print out the messages when the code is running.
screenZoom = 0.7
# The zoom value seems to control the fouriertransform image. Original value 24
# ---------------------------------------------------------------------------
ZOOM = 15
PLOT = True
```

In the code snippet above, we set the parameters. The comments with the code snippet should explain their use.

Defining the functions used in the code

```
def get_antenna_dict(filename=antenna_filename):
```

```
    ant_database = pd.read_csv(filename, sep=";", header='infer')
    ant_bits = np.array(ant_database['bit']) +1
    ant_bits_posx = np.array(ant_database['posx'], dtype=float)
    ant_bits_posy = np.array(ant_database['posy'], dtype=float)

    position_array = (np.array([ant_bits_posx,ant_bits_posy]).T)*scale_array
    ant_dict = dict(zip(ant_bits,position_array))

    return ant_dict

ant_dict = get_antenna_dict()
```

The function above imports the antenna coordinates and multiplies them by the **scale_array** to convert it from size on the board to real life size.

```
def getserialinterface(deviceroot="/dev/ttyACM", maxdevice=3, boudrate=11520
    """returns the serial interface"""

    while True:
        count = 0
        while count < maxdevice:
            device = f"{deviceroot}{count}"
            try:
                ser = serial.Serial(device, baudrate=boudrate)
                print(f"Serial interface {device} found!")
                return ser
            except:
                print(f"Serial interface {device} not found!. Trying the next one ...")
                count += 1
                # for testing, use this line, but uncomment for real runs
                return None
```

```
        print("No serial device found. Please plug it in. ...")
        time.sleep(2)
```

The function above converts the output of the ardruino into strings so it can be used in the program.

```
def waitforserialchange(ser, bitdict, ant_dict, npadarray=45, verbose=False):
    """Waits for a change to the serial interface, i.e. any change of a contact.
       Pauses in case there is no contact on one of the hour angles
    Parameters:
    device (str)
       device to which the serial interface is connected

    npadarray (int)
       split position of the 64 input array of the microcontroller into the two arrays
       the first npadarray are for the antennas, the remaining for the buttons

    Return:
    list
       positions of antennas with closed contacts

    list
       positions of buttons with closed contacts

    """
    validhaselection = False
    while not validhaselection:
        if (NoSerial):  # this part is for testing if ardruino is not connected
            # buttons = "0110000"
            buttons = buttons_inp
            # attenas = "111111111111111111111111111111111111111111111"
            attenas = attenas_inp

            serialinput = attenas  + buttons
            # serialinput= "11111110001111100011010001110000000000000000 000000
```

```
    print('this is the length of serial input',len(serialinput))
else:           # this part reads the output of the arduino
    serialinput = str(ser.readline().decode("utf-8").strip())
    if ser.in_waiting > 0:
        # In case there were several triggers for redrawing recorded,
        # take the last one
        serialbuffer = ser.read(ser.in_waiting).decode("utf-8").strip().split('\n')
        if verbose:
            print(f"Inputs recorded in the last loop {len(serialbuffer)}. Taking the
        serialinput = serialbuffer[-1]
time.sleep(1)
        # the next two line of code divide the 7+45 length output of the arduinc
linein2  = serialinput[npadarray:]
linein   = serialinput[:npadarray]

print(len(linein))
print(len(linein2))

# the condition below is in case there is no antenna's on the board
count = 0
while int(linein) == 0:
    print("Waiting for a valid antenna selection ...")
    time.sleep(2)
    serialinput = str(ser.readline().decode("utf-8").strip())
    linein2  = serialinput[npadarray:]
    linein   = serialinput[:npadarray]
    print(linein)




# The array from the controller should have 45 positions for 45 antennas. (n
# Antenna list will have a list of antenna numbers and they should start as 1.
# This is why we add a "0" to the array.
# Below we decide which antennas are placed and which buttons are conne
# -------------------------------------------------------------------------
bit_pos1  = np.where(np.array([bit == '1' for bit in "0"+linein]) == True)[0]  #
```

```
bit_pos2 = np.where(np.array([bit == '1' for bit in "0"+linein2]) == True)[0] #
buttons_config = np.where(np.array([bit == '1' for bit in "0"+linein2[:2]]) ==
buttons_image = np.where(np.array([bit == '1' for bit in "0"+linein2[3:]]) ==
if verbose:
    print(serialinput)
    print((len(serialinput)))

    print(f"antennas:   {linein}")
    print(f"buttons:    {linein2}")

    print(f"antenna list:    {bit_pos1}")
    print(f"button  list:    {bit_pos2}      ({' '.join([key for key in bitdict if bitdict[

    print(f"buttons_config:    {buttons_config}")
    print(f"buttons_image:    {buttons_image}")




  # Selectinng where antennas are placed
ant_pos =  np.array([np.array(ant_dict[bb]) for bb in bit_pos1])

if len(ant_pos)>0 and (bitdict['hourangle_m6'] in bit_pos2) or (bitdict['houra

    xx_antpos, yy_antpos = ant_pos.T
    if len(ant_pos) > 1:
        create_config_file(ant_pos)
        singledish = False
    elif len(ant_pos)==1: #one antenna show a singledish image
        singledish = True

    validhaselection = True
else:
    print('Waiting for valid hourangle selection and at least one antenna ...')
```

```
    # TODO: convert everything into the usage of bit_pos1 and bit_pos2. Rename b
    return bit_pos1, bit_pos2, buttons_config,buttons_image,ant_pos, xx_antpos, yy
```

The function above connects the string output from the ardruino to the information stored in the **ant_pos,** and **bitdict.** The main job of the function is to figure out how many antennas are placed and where they are placed and also convert the information from the ardruino into determining which buttons are pressed.

```


def select_model_and_hourangle(bitdict, bit_pos2,bitdict_config,bitdict_image,bu
    """Select the images and return the corresponding pixel scale and integration 
    """


    webcam = False


    pixel_scale = 0.05



# bitdict = {'hourangle_m6':     1,
#           'hourangle_0':      2,
#           'hourangle_p6':     3,
#           'agb_star':      4,
#           'galaxy_gas':       5,
#           'hltau':       6,
#           'outflow':   7}





    #load the imagefiles based on the bit values from the box
    if len(buttons_image) == 0:
        imagefile = imagefile1
    else:
        if  bitdict_image['agb_star'] == buttons_image[0]:
            imagefile = imagefile1
        elif bitdict_image['galaxy_gas']  == buttons_image[0]:
            imagefile = imagefile2
```

```
    elif bitdict_image['hltau']  == buttons_image[0]:
        imagefile = imagefile3
        # pixel_scale = 0.1
        # webcam = True
    elif bitdict_image['outflow']  == buttons_image[0]:
        imagefile = imagefile4
    else:
        imagefile = imagefile1
    # pixel_scale = 0.1
    # webcam = True
# print(imagefile)
# if not bitdict['agb_star'] in bit_pos2:
#     imagefile = imagefile1
# elif not bitdict['galaxy_gas'] in bit_pos2:
#     imagefile = imagefile2
# elif not bitdict['hltau'] in bit_pos2:
#     imagefile = imagefile3
#     pixel_scale = 0.1
#     webcam = True
# elif not bitdict['outflow'] in bit_pos2:
#     imagefile = imagefile4
#     pixel_scale = 0.1
#     webcam = True

# if (not bitdict['sgal'] in bit_pos2) and not (bitdict['cam'] in bit_pos2) and (not
#     imagefile = "models/marilyn-einstein.png"
#     pixel_scale = 0.1

# if bitdict['sgal'] in bit_pos2 and bitdict['m51'] in bit_pos2 and bitdict['cam'] in
#     imagefile= "models/mistery_med.png"

# which hourangle are we observing, is it a full track of hourangle or only dawn
# if bitdict['fulltrk'] in bit_pos2:
integration_time = 6
if bitdict['hourangle_0'] in bit_pos2:
    hourangle = 0
```

```
    elif bitdict['hourangle_m6'] in bit_pos2:
        hourangle = -5
    elif bitdict['hourangle_p6'] in bit_pos2:
        hourangle = +5
    # else:
    #     integration_time = 12
    #     hourangle = 0

    hourangle_start = hourangle - integration_time * 0.5
    hourangle_end   = hourangle + integration_time * 0.5

    if verbose:
        print(imagefile, integration_time, hourangle_start, hourangle_end)

    return webcam, imagefile, pixel_scale, integration_time, hourangle, hourangle_
    # return None
```

This function’s main job is to select which image needs to be displayed and worked on based on the output of ardruino. It also generates the time-period of observation based on the button selected on the kiosk.

```



def write_alma_config_file(filename, antenna_coords):
    filename = './arrays/' +filename
    with open(filename, 'w') as f:
        f.write("#-----------------------------------------------------------------
        f.write("#                                                                #\n")
        f.write("# Array definition file for ALMA, Cycle 6, Config 5, 12-m antennas.
        f.write("#                                                                #\n")
        f.write("#-----------------------------------------------------------------
        f.write("# Baseline Range: 15m-1.4km\n\n")
        f.write("# Name of the telescope\n")
        f.write("telescope = ALMA\n\n")
        f.write("# Name of the configuration\n")
        f.write("config = Custom-lego-alma\n\n")
```

```
        f.write("# Latitude of the array centre\n")
        f.write("latitude_deg = -23.0229\n\n")
        f.write("# Antenna diameter\n")
        f.write("diameter_m = 12.0\n\n")
        f.write("# Antenna coordinates (offset E, offset N)\n")
        for e, n in antenna_coords:
            f.write(f"{e}, {n}\n")
```

The function above creates the custom config file based on the configuration of the lego antennas on the kiosk

```
if (not NoSerial):
    ser = getserialinterface()
else:
    ser = None

matplotlib.rcParams['toolbar'] = 'None'
plt.style.use('dark_background')

# all images should have the same pixel size. Depending on the screen size.
# For example 400×400. This should speed up the processing.



starttime = time.time()
lasttime  = starttime

imglogo=mpimg.imread(model_logo_filename)




print((plt.get_backend()))

# This resizes the figure to the size of the screen
# Comment out lines below to remove.
# The size of the window is controlled by the screenZoom variable.
# screenZoom = 1 is fullscreen. 0< screenZoom <=1
```

```
# ---------------------------------------------------------------
scrsize = pyautogui.size()
mng = plt.get_current_fig_manager()
# mng.resize(int(scrsize[0]*screenZoom), int(scrsize[1]*screenZoom))
#mng.resize(*mng.window.showMaximized())
#mng.window.showMaximized()
mng.full_screen_toggle()
# KDE: title-bar, right click, set 'apply initially' to remove title, to maximise etc

#-------------------------------------------------------------
#---------------------------------------------------------------



########################### Start of the main program ##############
#start of main prgoram loop
serialinput="0"
serialinputlast=""
imglogo = mpimg.imread(model_logo_filename)
```

The code snippet above starts the plotting process. Its just setting it to have a black background and setting it to fullscreen.

Main program loop

```
Flag = True
while True:

#    Flag = False

    print("---------------------------------------------------------------------
    if verbose:
        starttime = time.time()

        lasttime = starttime
```

```
    thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistime-
#we fiddled with some constants here
#    hourangle_start = hourangle - 0.5
#    hourangle_end = hourangle + 0.5
#    hourangle+=1
#if hourangle > 6:
#        hourangle = -6
#        hourangle_start = -6
#        hourangle_end = 6
#FREQ += 1000
#if FREQ >= 3e4: FREQ = 5e3
try:
    if verbose:
        thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim
        computetimestart=time.time()
    #start the VRI with the latest antenna position config file, which we generate
    #bit strings from the boxes.

    ### linein = ser.readline()

    # moved above the serial input. Does not cost time anymore.
        thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim
    bit_pos1, bit_pos2, buttons_config,buttons_image,ant_pos, xx_antpos, yy_an
    # bit_pos1, bit_pos2, ant_pos, xx_antpos, yy_antpos, singledish = waitforseri
    write_alma_config_file("lego_alma.config", ant_pos)
    obsMan = observationManager(verbose=True, debug=True)

    if verbose:
        thistime = time.time(); print(("TIMING make observation manager (debugT
    obsMan.get_available_arrays()

    if verbose:
        thistime = time.time(); print(("TIMING make observation manager read arr
    # bit_pos1, bit_pos2, buttons_config,buttons_image,ant_pos, xx_antpos, yy_a
    # # bit_pos1, bit_pos2, ant_pos, xx_antpos, yy_antpos, singledish = waitforse
    # write_alma_config_file("lego_alma.config", ant_pos)
```

```
for i in range(1,9):
    pp = plt.subplot(2,4,i)
    xmin, xmax = pp.get_xlim()
    ymin, ymax = pp.get_ylim()
    rectangle = matplotlib.patches.Rectangle((xmin,xmin), xmax-xmin, ymax-
    pp.add_patch(rectangle)
# webcam, imagefile, pixel_scale, integration_time, hourangle, hourangle_sta
webcam, imagefile, pixel_scale, integration_time, hourangle, hourangle_start

if verbose:
    print(imagefile, integration_time,hourangle_start,hourangle_end,"<<<<<<
    print(("TIMING preparations: %f" %(time.time()-computetimestart)))
    computetimestart=time.time()
    thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim

# set up the VRI for this specific obs
# Select array configurations and hour-angle ranges.
obsMan.select_array('ALMA_Custom-lego-alma',haStart = hourangle_start,h
#obsMan.select_array('ALMA_Cycle6-C43-5',haStart = hourangle_start,haEr
#obsMan.select_array('ALMA_Cycle6-C43-1',haStart = hourangle_start,haEr
obsMan.get_selected_arrays()

if verbose:
    print(("TIMING select arrays: %f" %(time.time()-computetimestart)))
    computetimestart=time.time()
    thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim

# Set the observing frequency (MHz) and source declination (deg).
obsMan.set_obs_parms(FREQ, DEC)

if verbose:
    print(("TIMING set obs parmss: %f" %(time.time()-computetimestart)))
    computetimestart=time.time()

# Calculate the uv-coverage
```

```
obsMan.calc_uvcoverage()
if verbose:
    print(("TIMING uv coverage: %f" %(time.time()-computetimestart)))
    computetimestart=time.time()
    thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim

# XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
# XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

# if webcam is active do this

obsMan.load_model_image(imagefile)
obsMan.set_pixscale(pixel_scale)

if verbose:
    thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim
if PLOT:
    try:
        # Calculate the FFT of the model image
        obsMan.invert_model()

        if verbose:
            thistime = time.time(); print(("TIMING invert model %f  %f" %(thistim
            computetimestart=time.time()
        # if not singledish:
        #     # Grid the uv-coverage onto the same pixels as the FFT as the moc
        #     obsMan.grid_uvcoverage()

        #     if verbose:
        #         thistime = time.time(); print(("TIMING %f  %f  grid uv_coverage"
        #     # Create the beam image
        #     print("obsMan.calc_beam()")
        #     obsMan.calc_beam()

        #     if verbose:
        #         thistime = time.time(); print(("TIMING  %f  %f  calc beam" %(this
```

```
#     # Apply the uv-coverage and create observed image
#     obsMan.invert_observation()


    # Grid the uv-coverage onto the same pixels as the FFT as the mode
obsMan.grid_uvcoverage()

if verbose:
    thistime = time.time(); print(("TIMING %f  %f  grid uv_coverage" %(th
# Create the beam image
print("obsMan.calc_beam()")
obsMan.calc_beam()

if verbose:
    thistime = time.time(); print(("TIMING  %f  %f  calc beam" %(thistime
# Apply the uv-coverage and create observed image
obsMan.invert_observation()




if verbose:
    thistime = time.time(); print(("TIMING %f  %f  invert observations " %
plt.clf()

if verbose:
    thistime = time.time(); print(("TIMING %f %f  clf" %(thistime-starttim
plogo = plt.subplot(111)
plogo.imshow(imglogo)

plogo.set_position([0.0,0.0,0.1,0.1],which='both')
plogo.axes.get_xaxis().set_visible(False)
plogo.axes.get_yaxis().set_visible(False)

#plto antenna position
p0 = plt.subplot(2,4,1)
p0.set_title("Single Dish Antenna",fontsize = 15)
```

```
p0.scatter(0,0)
p0.set_xlim(-250,250)
p0.set_ylim(-250,250)
p0.set_xlabel("x (m)")
p0.set_ylabel("y (m)")
p0.set_aspect('equal')

pp0 = plt.subplot(2,4,5)
pp0.set_title("ALMA from source (perspective)",fontsize = 15)
pp0.text(4.5,-0.3,r"Powered by: Friendly VRI C.R. Purcell R. Truelove",h

hrangle_rad = np.radians(hourangle*15)
dec_rad = np.radians(DEC)
xx_earth_cen = -yy_antpos*np.sin(np.radians(-23.023))
yy_earth_cen = xx_antpos
zz_earth_cen = yy_antpos*np.cos(np.radians(-23.023))

xx_antpos_proj = -(xx_earth_cen*np.sin(hrangle_rad)+yy_earth_cen*np
yy_antpos_proj = -xx_earth_cen*np.sin(dec_rad)*np.cos(hrangle_rad) +

if hourangle > 0:
   pp0.scatter(yy_antpos_proj,xx_antpos_proj)
elif hourangle < 0:
   pp0.scatter(-yy_antpos_proj,-xx_antpos_proj)
elif hourangle == 0:
   pp0.scatter(xx_antpos_proj,-yy_antpos_proj)

pp0.set_xlim(-250,250)
pp0.set_ylim(-250,250)
pp0.set_aspect('equal')
pp0.set_xlabel("x (m)")
pp0.set_ylabel("y (m)")

#plot original image
p1 = plt.subplot(2,4,2)
if imagefile == "/home/kiosk/alma_sid_07_05_2025/Alma_main/models/
```

```
    qmarkimg=mpimg.imread('/home/kiosk/alma_sid_07_05_2025/Alma_r
    p1.imshow(qmarkimg,cmap = colormap)
else:
    p1.imshow(np.real(obsMan.modelImgArr),origin = 'lower',cmap = colo
p1.axes.get_xaxis().set_visible(False)
p1.axes.get_yaxis().set_visible(False)
p1.set_title("Picture of the source", fontsize = 15)


# print("SINGLEDISH")
p2 = plt.subplot(2,4,3)
sigma = 50
x = np.linspace(-250, 250, 100)
y = np.linspace(-250, 250, 100)
x_grid, y_grid = np.meshgrid(x, y)
single_beam = np.exp(-((x_grid)**2 + (y_grid)**2)/(2*sigma**2))
p2.imshow(single_beam,origin = 'lower',cmap = colormap)
p2.text(-0.1,0.5,r"$\otimes$",ha='center', va='center',fontsize = 25, tran
p2.axes.get_xaxis().set_visible(False)
p2.axes.get_yaxis().set_visible(False)
p2.set_title("Beam of single dish",fontsize = 15)


#plot different final image, no beam
# with open(imagefile.split(".")[0]+"_SDOUT.pickle",'rb') as fin:
#     simage_sd = pickle.load(fin)
sgdish_image = gaussian_filter(obsMan.modelImgArr,sigma,mode = 'co
p3 = plt.subplot(2,4,4)
p3.imshow(sgdish_image,origin = 'lower',cmap = colormap)
p3.text(-0.1,0.5,r"$=$",ha='center', va='center',fontsize = 25, transform
p3.axes.get_xaxis().set_visible(False)
p3.axes.get_yaxis().set_visible(False)
p3.set_title("Single Dish View",fontsize = 15)
```

```
#fft of original image

p4 = plt.subplot(2,4,6)
# mm,ll = np.shape(obsMan.modelFFTarr)
#p4.imshow(np.log10(abs(obsMan.modelFFTarr)),cmap = 'gist_heat',int
# p4.imshow(np.log10(abs(obsMan.modelFFTarr)),cmap = 'gist_heat',in
p4.imshow(obsMan.modelImgArr,cmap = colormap,origin = 'lower')
# p4.set_xlim(ll/2-ll/ZOOM,ll/2+ll/ZOOM)
# p4.set_ylim(mm/2-mm/ZOOM,mm/2+mm/ZOOM)
p4.axes.get_xaxis().set_visible(False)
p4.axes.get_yaxis().set_visible(False)
p4.set_title("Picture of the source",fontsize = 15)



if not singledish:
    computetimestart = time.time()
    #plot uvcoverage
    p5 = plt.subplot(2,4,7)
    p5.scatter(obsMan.arrsSelected[0]['uArr_lam'],obsMan.arrsSelected
    p5.scatter(-1.*obsMan.arrsSelected[0]['uArr_lam'],-1*obsMan.arrsSe
    thistime = time.time(); print(("TIMING %f  %f uv coverage" %(thistim
    # p5.set_xlim(obsMan.pixScaleFFTX_lam*(-ll/ZOOM),(obsMan.pixSca
    # p5.set_ylim(obsMan.pixScaleFFTY_lam*(-mm/ZOOM),(obsMan.pix
    p5.set_aspect(p0.get_aspect())
    p5.text(-0.1,0.5,r"$\otimes$",ha='center', va='center',fontsize = 20, tr
    p5.axes.get_xaxis().set_visible(False)
    p5.axes.get_yaxis().set_visible(False)
    p5.set_title("Beam of the ALMA interferometer",fontsize = 15)



    #fft of final image
    p6 = plt.subplot(2,4,8)
    # p6.imshow(np.log10(abs(obsMan.obsFFTarr)+1e3)-3,cmap = color
    p6.imshow(np.real(obsMan.obsImgArr),origin='lower',cmap=colorma
    # p6.set_xlim(ll/2-ll/ZOOM,ll/2+ll/ZOOM)
    # p6.set_ylim(mm/2-mm/ZOOM,mm/2+mm/ZOOM)
```

```
            p6.text(-0.1,0.5,r"$=$",ha='center', va='center',fontsize = 20, transfo
            p6.set_aspect('equal')
            p6.axes.get_xaxis().set_visible(False)
            p6.axes.get_yaxis().set_visible(False)
            p6.set_title("ALMA view",fontsize = 15)
            computetimestart=time.time()
        else:
            p5 = plt.subplot(2,4,7)
            sigma = 80
            x = np.linspace(-250, 250, 100)
            y = np.linspace(-250, 250, 100)
            x_grid, y_grid = np.meshgrid(x, y)
            single_beam = np.exp(-((x_grid)**2 + (y_grid)**2)/(2*sigma**2))
            p5.imshow(single_beam,origin = 'lower',cmap = colormap)
            p5.text(-0.1,0.5,r"$\otimes$",ha='center', va='center',fontsize = 25, tr
            p5.axes.get_xaxis().set_visible(False)
            p5.axes.get_yaxis().set_visible(False)
            p5.set_title("Beam of single ALMA antenna",fontsize = 15)


            #plot different final image, no beam
            # with open(imagefile.split(".")[0]+"_SDOUT.pickle",'rb') as fin:
            #     simage_sd = pickle.load(fin)
            sgdish_image = gaussian_filter(obsMan.modelImgArr,sigma,mode =
            p6 = plt.subplot(2,4,8)
            p6.imshow(sgdish_image,origin = 'lower',cmap = colormap)
            p6.text(-0.1,0.5,r"$=$",ha='center', va='center',fontsize = 25, transfo
            p6.axes.get_xaxis().set_visible(False)
            p6.axes.get_yaxis().set_visible(False)
            p6.set_title("Single ALMA View",fontsize = 15)


    except Exception as e:
        print(e)
        pass
```

```
        else:
            print("Not plotting")
            pass

        if verbose:
            print(ant_pos)
            print(("Time for one loop: %s" % str((time.time()-starttime))))
        print("Pausing for   " ,LOOP_TIME)
        plt.pause(LOOP_TIME)

        #break
        # r=input()






    except KeyboardInterrupt:
        plt.clf()
        pl = plt.subplot(111)
        pl.imshow(np.real(obsMan.modelImgArr),origin='lower',cmap='gist_heat')
        pl.axes.get_xaxis().set_visible(False)
        pl.axes.get_yaxis().set_visible(False)
        if False:
            # This is for getting screenshots for debugging. However in production
            # we do not save any images for reasons of privacy in case the images
            # were done with the webcam.
            plt.savefig(imagefile.split(".")[0]+"_almaview_saved"+"."+ imagefile.split("

        break
# cam.release()
plt.close()
```

This is the main working part of the code. Now i will divide this into smaller parts to explain them in bit more detail.

Part 1

```
Flag = True
while True:

#    Flag = False

    print("----------------------------------------------------------------------------------
    if verbose:
        starttime = time.time()

        lasttime = starttime
        thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistime-

    try:
        if verbose:
            thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim
            computetimestart=time.time()


            thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim
        bit_pos1, bit_pos2, buttons_config,buttons_image,ant_pos, xx_antpos, yy_an
        write_alma_config_file("lego_alma.config", ant_pos)
        obsMan = observationManager(verbose=True, debug=True)

        if verbose:
            thistime = time.time(); print(("TIMING make observation manager (debugT
        obsMan.get_available_arrays()

        if verbose:
            thistime = time.time(); print(("TIMING make observation manager read arr
        for i in range(1,9):
            pp = plt.subplot(2,4,i)
            xmin, xmax = pp.get_xlim()
            ymin, ymax = pp.get_ylim()
            rectangle = matplotlib.patches.Rectangle((xmin,xmin), xmax-xmin, ymax-
```

```
    pp.add_patch(rectangle)
webcam, imagefile, pixel_scale, integration_time, hourangle, hourangle_start
if verbose:
    print(imagefile, integration_time,hourangle_start,hourangle_end,"<<<<<<
    print(("TIMING preparations: %f" %(time.time()-computetimestart)))
    computetimestart=time.time()
    thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim

# set up the VRI for this specific obs
# Select array configurations and hour-angle ranges.
obsMan.select_array('ALMA_Custom-lego-alma',haStart = hourangle_start,h
obsMan.get_selected_arrays()

if verbose:
    print(("TIMING select arrays: %f" %(time.time()-computetimestart)))
    computetimestart=time.time()
    thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim

# Set the observing frequency (MHz) and source declination (deg).
obsMan.set_obs_parms(FREQ, DEC)

if verbose:
    print(("TIMING set obs parmss: %f" %(time.time()-computetimestart)))
    computetimestart=time.time()

# Calculate the uv-coverage
obsMan.calc_uvcoverage()
if verbose:
    print(("TIMING uv coverage: %f" %(time.time()-computetimestart)))
    computetimestart=time.time()
    thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim

# XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
# XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

# if webcam is active do this
```

```
obsMan.load_model_image(imagefile)
obsMan.set_pixscale(pixel_scale)
```

This is the part where you supply the values to **vriCalc.py for** calculation. The class observationManager from **vriCalc.py** is called to supply the values for observation, like the picture (model), position of the object in the sky, frequency of observation, duration of the observation, configuration of the antennas, etc.

Here are some importnat line in the code that do the calulcations, rest of it is just plotting.

```
obsMan = observationManager(verbose=True, debug=True) #initiates the observ
obsMan.get_available_arrays() #gets all the configurations, including our custom
obsMan.select_array('ALMA_Custom-lego-alma',haStart = hourangle_start,haEnd
obsMan.get_selected_arrays() # gets the specifics of our custom configuration
obsMan.set_obs_parms(FREQ, DEC) # sets observation parameters
obsMan.calc_uvcoverage() # calcualtes the UV coverage based on the observati
obsMan.load_model_image(imagefile) # loads our image
obsMan.set_pixscale(pixel_scale) # converts the image size from pixels to arcsec
```

Part 2

```
if verbose:
    thistime = time.time(); print(("TIMING %f  %f" %(thistime-starttime,thistim
if PLOT:
    try:
        # Calculate the FFT of the model image
        obsMan.invert_model()
        if verbose:
            thistime = time.time(); print(("TIMING invert model %f  %f" %(thistim
```

```
    computetimestart=time.time()
obsMan.grid_uvcoverage()

if verbose:
    thistime = time.time(); print(("TIMING %f  %f  grid uv_coverage" %(th
# Create the beam image
print("obsMan.calc_beam()")
obsMan.calc_beam()

if verbose:
    thistime = time.time(); print(("TIMING  %f  %f  calc beam" %(thistime
# Apply the uv-coverage and create observed image
obsMan.invert_observation()
if verbose:
    thistime = time.time(); print(("TIMING %f  %f  invert observations " %
plt.clf()
if verbose:
    thistime = time.time(); print(("TIMING %f %f  clf" %(thistime-starttim
plogo = plt.subplot(111)
plogo.imshow(imglogo)

plogo.set_position([0.0,0.0,0.1,0.1],which='both')
plogo.axes.get_xaxis().set_visible(False)
plogo.axes.get_yaxis().set_visible(False)

#plto antenna position
p0 = plt.subplot(2,4,1)
p0.set_title("Single Dish Antenna",fontsize = 15)
p0.scatter(0,0)
p0.set_xlim(-250,250)
p0.set_ylim(-250,250)
p0.set_xlabel("x (m)")
p0.set_ylabel("y (m)")
p0.set_aspect('equal')

pp0 = plt.subplot(2,4,5)
```

```
pp0.set_title("ALMA from source (perspective)",fontsize = 15)
pp0.text(4.5,-0.3,r"Powered by: Friendly VRI C.R. Purcell R. Truelove",h

hrangle_rad = np.radians(hourangle*15)
dec_rad = np.radians(DEC)
xx_earth_cen = -yy_antpos*np.sin(np.radians(-23.023))
yy_earth_cen = xx_antpos
zz_earth_cen = yy_antpos*np.cos(np.radians(-23.023))

xx_antpos_proj = -(xx_earth_cen*np.sin(hrangle_rad)+yy_earth_cen*np
yy_antpos_proj = -xx_earth_cen*np.sin(dec_rad)*np.cos(hrangle_rad) +

if hourangle > 0:
    pp0.scatter(yy_antpos_proj,xx_antpos_proj)
elif hourangle < 0:
    pp0.scatter(-yy_antpos_proj,-xx_antpos_proj)
elif hourangle == 0:
    pp0.scatter(xx_antpos_proj,-yy_antpos_proj)

pp0.set_xlim(-250,250)
pp0.set_ylim(-250,250)
pp0.set_aspect('equal')
pp0.set_xlabel("x (m)")
pp0.set_ylabel("y (m)")

#plot original image
p1 = plt.subplot(2,4,2)
if imagefile == "/home/kiosk/alma_sid_07_05_2025/Alma_main/models/
    qmarkimg=mpimg.imread('/home/kiosk/alma_sid_07_05_2025/Alma_
    p1.imshow(qmarkimg,cmap = colormap)
else:
    p1.imshow(np.real(obsMan.modelImgArr),origin = 'lower',cmap = col
p1.axes.get_xaxis().set_visible(False)
p1.axes.get_yaxis().set_visible(False)
p1.set_title("Picture of the source", fontsize = 15)
```

```
# print("SINGLEDISH")
p2 = plt.subplot(2,4,3)
sigma = 50
x = np.linspace(-250, 250, 100)
y = np.linspace(-250, 250, 100)
x_grid, y_grid = np.meshgrid(x, y)
single_beam = np.exp(-((x_grid)**2 + (y_grid)**2)/(2*sigma**2))
p2.imshow(single_beam,origin = 'lower',cmap = colormap)
p2.text(-0.1,0.5,r"$\otimes$",ha='center', va='center',fontsize = 25, trar
p2.axes.get_xaxis().set_visible(False)
p2.axes.get_yaxis().set_visible(False)
p2.set_title("Beam of single dish",fontsize = 15)


#plot different final image, no beam
# with open(imagefile.split(".")[0]+"_SDOUT.pickle",'rb') as fin:
#     simage_sd = pickle.load(fin)
sgdish_image = gaussian_filter(obsMan.modelImgArr,sigma,mode = 'cc
p3 = plt.subplot(2,4,4)
p3.imshow(sgdish_image,origin = 'lower',cmap = colormap)
p3.text(-0.1,0.5,r"$=$",ha='center', va='center',fontsize = 25, transform
p3.axes.get_xaxis().set_visible(False)
p3.axes.get_yaxis().set_visible(False)
p3.set_title("Single Dish View",fontsize = 15)



#fft of original image

p4 = plt.subplot(2,4,6)
# mm,ll = np.shape(obsMan.modelFFTarr)
#p4.imshow(np.log10(abs(obsMan.modelFFTarr)),cmap = 'gist_heat',int
# p4.imshow(np.log10(abs(obsMan.modelFFTarr)),cmap = 'gist_heat',in
p4.imshow(obsMan.modelImgArr,cmap = colormap,origin = 'lower')
# p4.set_xlim(ll/2-ll/ZOOM,ll/2+ll/ZOOM)
```

```
# p4.set_ylim(mm/2-mm/ZOOM,mm/2+mm/ZOOM)
p4.axes.get_xaxis().set_visible(False)
p4.axes.get_yaxis().set_visible(False)
p4.set_title("Picture of the source",fontsize = 15)


if not singledish:
    computetimestart = time.time()
    #plot uvcoverage
    p5 = plt.subplot(2,4,7)
    p5.scatter(obsMan.arrsSelected[0]['uArr_lam'],obsMan.arrsSelected
    p5.scatter(-1.*obsMan.arrsSelected[0]['uArr_lam'],-1*obsMan.arrsSe
    thistime = time.time(); print(("TIMING %f  %f uv coverage" %(thistim
    # p5.set_xlim(obsMan.pixScaleFFTX_lam*(-ll/ZOOM),(obsMan.pixSca
    # p5.set_ylim(obsMan.pixScaleFFTY_lam*(-mm/ZOOM),(obsMan.pixS
    p5.set_aspect(p0.get_aspect())
    p5.text(-0.1,0.5,r"$\otimes$",ha='center', va='center',fontsize = 20, tr
    p5.axes.get_xaxis().set_visible(False)
    p5.axes.get_yaxis().set_visible(False)
    p5.set_title("Beam of the ALMA interferometer",fontsize = 15)


    #fft of final image
    p6 = plt.subplot(2,4,8)
    # p6.imshow(np.log10(abs(obsMan.obsFFTarr)+1e3)-3,cmap = color
    p6.imshow(np.real(obsMan.obsImgArr),origin='lower',cmap=colorma
    # p6.set_xlim(ll/2-ll/ZOOM,ll/2+ll/ZOOM)
    # p6.set_ylim(mm/2-mm/ZOOM,mm/2+mm/ZOOM)
    p6.text(-0.1,0.5,r"$=$",ha='center', va='center',fontsize = 20, transfo
    p6.set_aspect('equal')
    p6.axes.get_xaxis().set_visible(False)
    p6.axes.get_yaxis().set_visible(False)
    p6.set_title("ALMA view",fontsize = 15)
    computetimestart=time.time()
else:
    p5 = plt.subplot(2,4,7)
```

```
            sigma = 80
            x = np.linspace(-250, 250, 100)
            y = np.linspace(-250, 250, 100)
            x_grid, y_grid = np.meshgrid(x, y)
            single_beam = np.exp(-((x_grid)**2 + (y_grid)**2)/(2*sigma**2))
            p5.imshow(single_beam,origin = 'lower',cmap = colormap)
            p5.text(-0.1,0.5,r"$\otimes$",ha='center', va='center',fontsize = 25, tr
            p5.axes.get_xaxis().set_visible(False)
            p5.axes.get_yaxis().set_visible(False)
            p5.set_title("Beam of single ALMA antenna",fontsize = 15)


            #plot different final image, no beam
            # with open(imagefile.split(".")[0]+"_SDOUT.pickle",'rb') as fin:
            #     simage_sd = pickle.load(fin)
            sgdish_image = gaussian_filter(obsMan.modelImgArr,sigma,mode =
            p6 = plt.subplot(2,4,8)
            p6.imshow(sgdish_image,origin = 'lower',cmap = colormap)
            p6.text(-0.1,0.5,r"$=$",ha='center', va='center',fontsize = 25, transfo
            p6.axes.get_xaxis().set_visible(False)
            p6.axes.get_yaxis().set_visible(False)
            p6.set_title("Single ALMA View",fontsize = 15)


    except Exception as e:
        print(e)
        pass

else:
    print("Not plotting")
    pass

if verbose:
    print(ant_pos)
    print(("Time for one loop: %s" % str((time.time()-starttime))))
print("Pausing for   " ,LOOP_TIME)
```

```
        plt.pause(LOOP_TIME)

        #break
        # r=input()

    except KeyboardInterrupt:
        plt.clf()
        pl = plt.subplot(111)
        pl.imshow(np.real(obsMan.modelImgArr),origin='lower',cmap='gist_heat')
        pl.axes.get_xaxis().set_visible(False)
        pl.axes.get_yaxis().set_visible(False)
        if False:
            # This is for getting screenshots for debugging. However in production
            # we do not save any images for reasons of privacy in case the images
            # were done with the webcam.
            plt.savefig(imagefile.split(".")[0]+"_almaview_saved"+"."+ imagefile.split("

        break
# cam.release()
plt.close()
```

Here are some important line in the code that do the calculations, rest of it is just plotting.

```
obsMan.invert_model() # calculates the fft invert of our image
obsMan.grid_uvcoverage() # gets the grid UV coverage
obsMan.calc_beam() # calucaltes the beam
obsMan.invert_observation() # applies the UV coverage to create the observed i
```

Rest of the code that just plots rest of the images.