MOBILTELESCO

The mysteries of the cosmos
now in your pocket

Project Idea:

This project aims to globalize the activity of stargazing through the means of smart phone based astrophotography. Using automated guidance system to point the phone mount in the correct angle based on real time astronomy data, image output and user interface, the process of astrophotography is simplified and hence made more accessible to more people who might not poses the expertise of astrophotography.

The mount works by holding the phone by a vice grip that is swiveling freely on two motors, providing altitude control, next this whole structure is freely movable at the azimuth level by the mechanism of a geared track. Additionally, an equatorial balance control is provide by the help of pistons that can be triggered individually, tilting the whole system from its base at equatorial angles.

Using real time data from Stellarium API, the position of any heavenly body can be tracked from a given position on earth using the phone's accelerometer, gyroscope and latitude + longitude readings. Then, the
image capturing process takes place and the settings on the camera such as ISO, Exposure, WB, interval shoot, focus etc. are changed in real time based on objects of interests that are automatically detected through the camera.

The tracking mechanism is also to be provided using YOLO based image detection model, currently in development and this will be first ever active tracking mechanism not based on any database for an astrophotography device that is not a telescope.

Next, the images are uploaded over server and then processed via DSS cli which I used to stack the images. Next the images are passed through different image processing filters including brightness and contrast control, color based contrast and brightness control, histogram based denoising, kernel based sharpening, and Tone mapping via the customized library created along with this project.

Components:

- Phone mount
- Three axis free moving, gyroscopic holder
- Microcontroller
- Power supply
- Pistons
- Motors

Working:

Module:

The mount works by holding the phone by a vice grip that is swiveling freely on two motors, providing altitude control, next this whole structure is freely movable at the azimuth level by the mechanism of a geared track. Additionally, a equatorial balance control is provide by the help of pistons that can be triggered individually, tilting the whole system from its base at equatorial angles.

Using real time data from Stellarium API, the position of any heavenly body can be tracked from a given position on earth using the phone's accelerometer, gyroscope and latitude + longitude readings. Then, the image capturing process takes place and the settings on the camera such as ISO, Exposure, WB, interval shoot, focus etc. are changed in real time based on objects of interests that are automatically detected through the camera's lens.

The captured images are then uploaded to a cloud server for automated stacking using DeepSkyStacker in a remote mode. The registered and stacked file is then moved on to another programming within the same server for image processing where multiple processes including light, color exposure control, lens aberration removal, noise and grain removal and texture control are performed. Once finished, the image is downloaded.

Imaging Processing:

The processing of astrophotography is generally a diverse process based on the object of desire and the type of image capturing device. However, as this study has used mobile devices and most of them have similar technologies in their image capturing(CMOS based Active Pixel Sensors), we can create a general flow of image processing to bring out the hidden wonders in the captured images.

All astrophotography starts with image stacking, in this study we use the DeepSkyStacker (DSS) software which is a standard for amateur astrophotography and provides reliable output. As there is no API available for the DSS software, we have created a workaround script by running the CLI version of the same on Google Colab. While the stacking could have been done on an exe file on the computer, it wouldn't fit into the ultimate goal of the project that is to make it standalone.

I. Pixel based operations:

1. Exposure and Contrast control:

i.

We use a linear transform function wherein two hyper parameters α and β.
Using linear transformation,

$$\mathbf{I_{i,j,k}} = \sum_{K=0}^{3} \boldsymbol{\alpha} \times \mathbf{I_{i,j,k}} + \boldsymbol{\beta}$$

where α adjusts the contrast and β shifts the brightness, k = r,g,b matrix.

ii.

Exposure Adjustment is based on the formula $new\_img = img \times scale$, where $scale = 2^{stops}$. This simulates adjusting the exposure by a certain number of stops.

$$\mathbf{img_{exp}} = \mathbf{img} \times \mathbf{2^{exposure\_stops}}$$

The scale factor 2^{stops} is derived from the relationship between exposure stops and the intensity of light. Each stop doubles or halves the intensity of the image.
For example:

- Stops = 1 increases the exposure (brightness) by a factor of 2.
- Stops = −1 decreases the exposure (darkens the image) by a factor of 2.

$$\mathbf{img_{cont}(x)} = \frac{\mathbf{1}}{\mathbf{1+e^{-strength\cdot(x-0.5)}}}$$

Here,

- x represents the pixel intensity at position (i,j,k), and midpoint is the reference intensity (0.5, representing the mid-range of intensity values).

- Strength controls how much contrast adjustment is applied. A higher value increases the contrast, and a lower value reduces it.

- For positive values of strength, the contrast is enhanced, and for negative values, the image's contrast is decreased.

$$\mathbf{adjusted\_img} = \int_{\mathbf{x=0}}^{\mathbf{n}} \mathbf{img_{exp}(x) + img_{cont}(x)}$$

2. Highlights and Shadow control:

Then, we use the exposure control, again a linear transform(but in Greyscale only)

$$\mathbf{I_{new}(x,y) = \alpha \cdot I(x,y) + \beta}$$

Where:

- I(x,y) is the pixel intensity of the original image at position (x,y).

- Inew(x,y) is the pixel intensity of the transformed image at the same position.

- α is the scaling factor (contrast). It stretches or compresses the intensity range:
  - If α>1, the contrast increases, and the image becomes more "dynamic" with more distinct highlights and shadows.
  - If α<1, the contrast decreases, making the image appear flatter.

- β is the offset (brightness). It shifts the pixel intensity:
  - If β>0, the image becomes brighter.
  - If β<0, the image becomes darker.

3. White and Blacks control:

Here we control the presence of white and black saturation levels by use of Drago's tone mapping

i. Luminance calculation:

Luminance is calculated using a weighted sum of the RGB channels based on the standard luminosity formula for converting RGB to grayscale:

$$L(x,y) = 0.2126 \bullet R(x,y) + 0.7152 \bullet \mathbf{G}(\mathbf{x},\mathbf{y}) + 0.0722 \bullet \mathbf{B}(\mathbf{x},\mathbf{y})$$

where L(x,y) represents the luminance at pixel location (x,y), and R(x,y),G(x,y),B(x,y) represent the red, green, and blue channel values at the same location.

ii. Logarithmic compression:

The luminance is compressed using the natural logarithm to bring out details in darker regions and reduce the influence of bright areas:

$$\mathbf{L}(\mathbf{x}',\mathbf{y}') = \mathbf{log}(\mathbf{L}(\mathbf{x},\mathbf{y}) + \mathbf{1})$$

Log p(x) is used to compute log(x+1to avoid logarithmic singularities for very small values. The result is then normalized by dividing by the maximum luminance to scale it to the range [0,1][0, 1][0,1].

iii. Gaussian Blur for local Contrast:

A Gaussian blur is applied to the logarithmically compressed luminance to compute the local average luminance:

$$\mathbf{I_G}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{1}}{\mathbf{2\pi\sigma^2}} \int_{-\infty}^{\infty} \mathbf{exp}\left(-\frac{(\mathbf{x}-\mathbf{x^t})^2}{\mathbf{2\sigma^2}}\right) \bullet \mathbf{L}(\mathbf{x}',\mathbf{y}')\mathbf{dx}'$$

The Gaussian kernel smoothens the luminance values, enhancing local contrast by reducing the influence of local variations.

iv. Gamma Correction:

Gamma correction is applied to the enhanced luminance to adjust the image's brightness in a perceptually uniform manner:

$$\mathbf{img_{tonemapped}} = \mathbf{image} \times \mathbf{I_G(x,y)^{\gamma}} \times \mathbf{s}$$

The parameter γ controls the brightness and contrast, with values less than 1 darkening the image and values greater than 1 brightening the image.

II. Kernel based operations:

1. Texture Enhancement (Sharpening)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & \gamma & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

γ -> Level of sharpness

Convolution Operator of I on K

$$\mathbf{I = I + (I * k) \bullet \beta}$$

Sharpened Image = Orignal Image + Sharpened Details • Strength

2. Clarity Enhancement/Edge contrast

$$\boldsymbol{Edges} = \nabla^2 \mathrm{f(x,y)}$$

∇ -> Laplacian Operator

Edge Detection:

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

Edge Amplification:

$$\varepsilon_{amp} = Edges \bullet Edge - strength$$

Blending:

$$I_{clarity} = (I \bullet (I - blend\_factor)) + (E\_amp \bullet blend + factor)$$

III. Histogram based operations:

1. Noise removal:

Convert to YCrCb:

Y - > Luminance: Brightness,
CrCb -> Chrominance: Color Information

$$Y = 0.2126 \cdot B + 0.7152 \cdot G + 0.0722 \cdot R$$

2DFFT on Y:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-2\pi i}\left(\frac{ux}{M} + \frac{vy}{N}\right)$$

Low pass filter Mask:

$$dist(u,v) = \sqrt{(u-u_0)^2 + (v-v_0)^2}$$

(u_0, v_0) centre of frequency domain

$$F_{lowpass}(u, v) = F_{shifted}(u, v) \bullet mask(u, v)$$

Blending mask:

$$mask(u, v) = \left(\frac{distance(u, v)}{max_{dist}}\right)^{m}$$

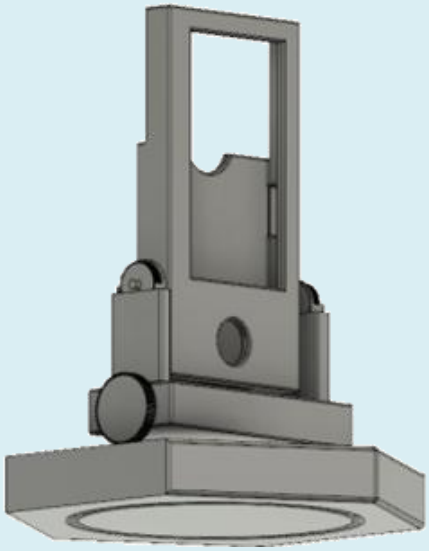
Blended DFT:

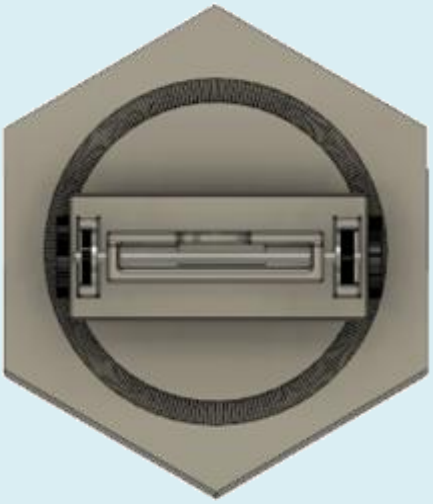
$$F_{blended}(u, v) = F_{lowpass}(u, v) \bullet \big(1 - mask(u, v)\big) + F_{shift}(u, v) \bullet mask(u, v)$$

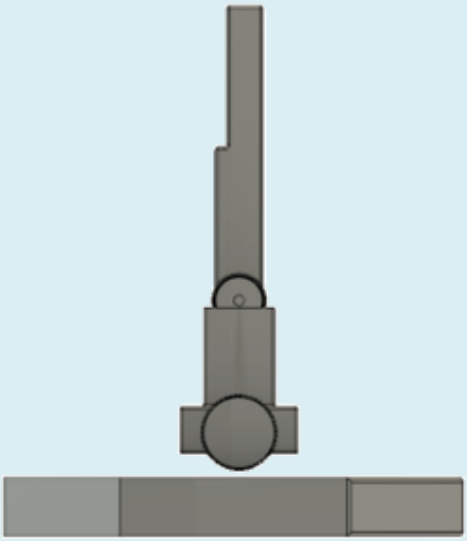
Inverse FFT:

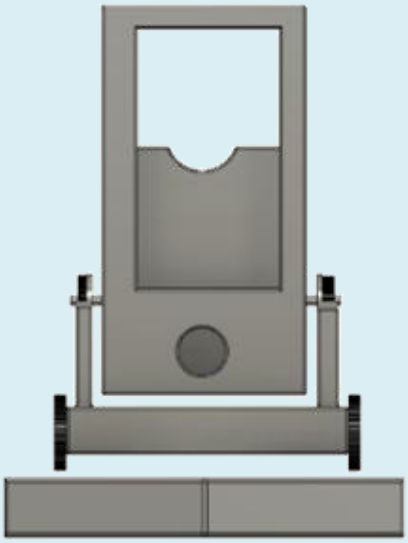
$$\int_{denoised}(x, y) = \frac{1}{M \bullet N} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f_{blended}(u, v) \bullet e^{2\pi i\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

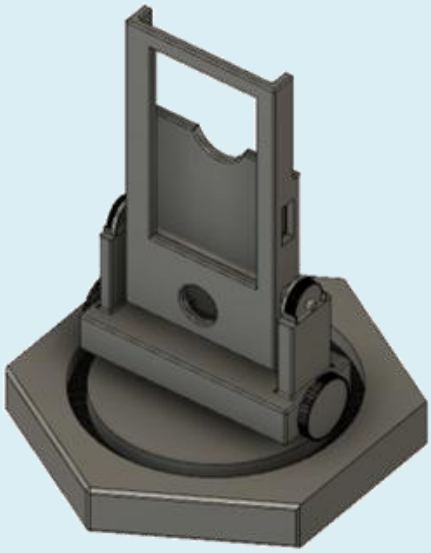
Visualization:





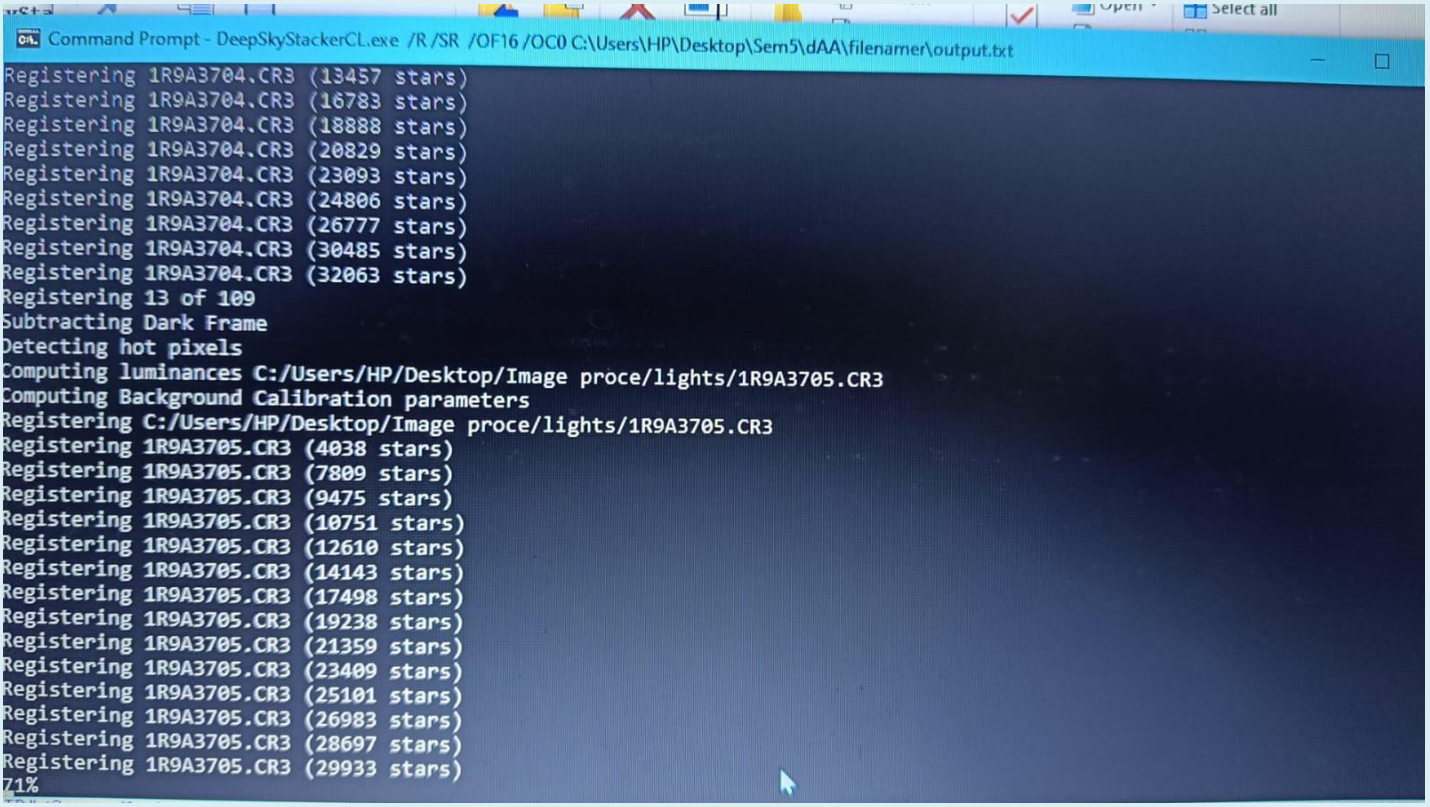




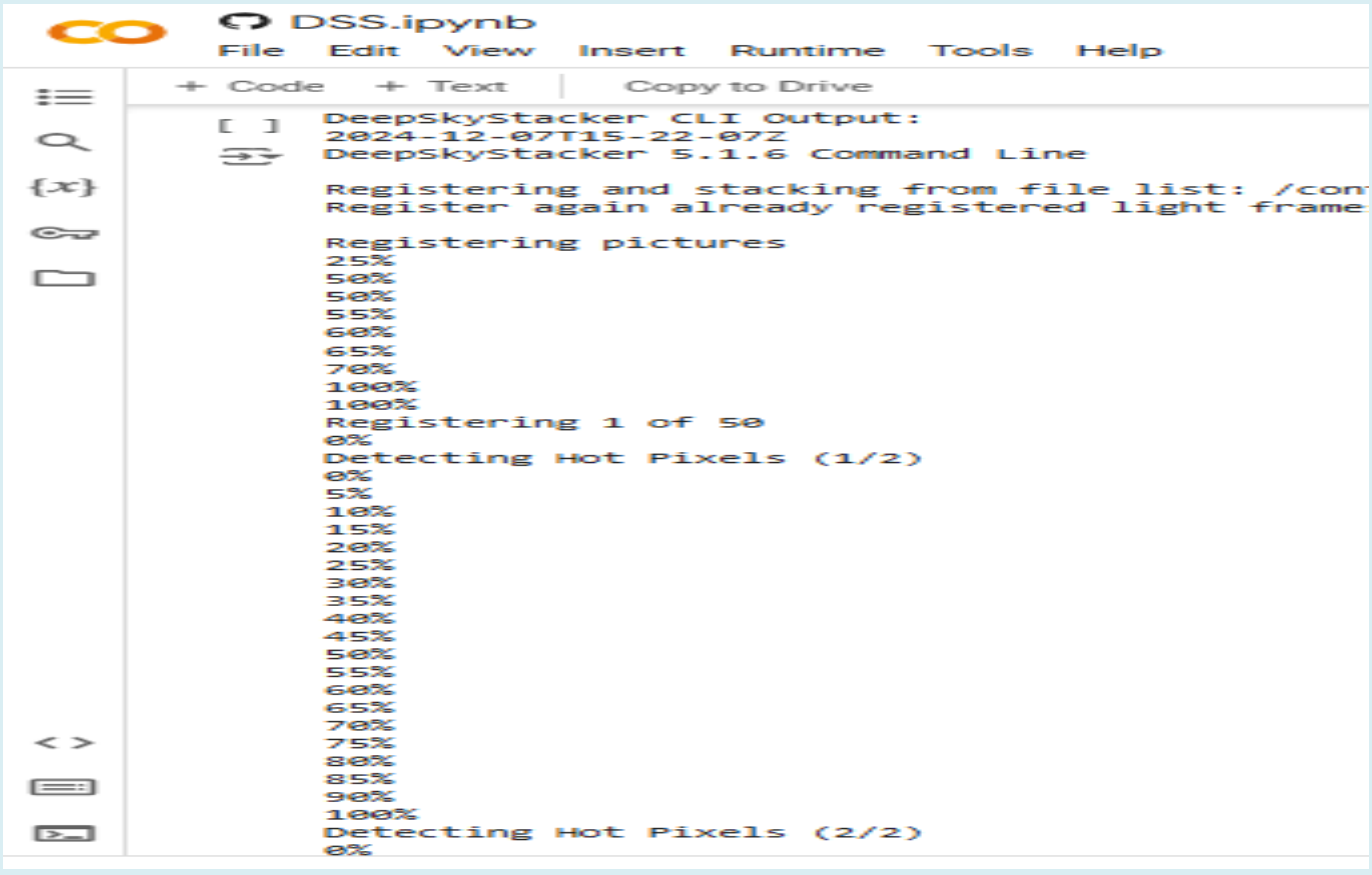


Programming:

Deep Sky Stacker CLI:



Server based CLI: https://colab.research.google.com/github/Shantanu909/MobilTelesco/blob/main/DSS.ipynb



Android Capture App:

```
package com.example.cameracaptureapp

import android.Manifest
import android.content.Context
import android.content.pm.PackageManager
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import android.location.Location
import android.location.LocationListener
import android.location.LocationManager
import android.os.Bundle
import android.os.Environment
import android.os.Handler
import android.util.Log
import android.widget.Button
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.camera.core.CameraSelector
import androidx.camera.core.ImageCapture
import androidx.camera.core.ImageCaptureException
import androidx.camera.core.Preview
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.core.content.ContextCompat
import androidx.lifecycle.LifecycleOwner
import java.io.File
import java.io.FileWriter
import java.io.IOException
import java.util.Locale
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors

class MainActivity : AppCompatActivity() {

    private var imageCapture: ImageCapture? = null
    private lateinit var cameraExecutor: ExecutorService
    private lateinit var locationManager: LocationManager
    private var gpsLocation: Location? = null
    private lateinit var sensorManager: SensorManager
    private var accelerometerData = FloatArray(3)
    private val handler = Handler()
    private lateinit var captureRunnable: Runnable

     override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Initialize Camera Executor
        cameraExecutor = Executors.newSingleThreadExecutor()

        // Initialize location manager and sensor manager
        locationManager =
getSystemService(Context.LOCATION_SERVICE) as
LocationManager
        sensorManager =
getSystemService(Context.SENSOR_SERVICE) as SensorManager

        // Register accelerometer listener
        val accelerometer =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER
)
        sensorManager.registerListener(object : SensorEventListener {
            fun onSensorChanged(event: SensorEvent) {
                accelerometerData = event.values.clone()
            }

             fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}
        }, accelerometer, SensorManager.SENSOR_DELAY_NORMAL)

        // Capture button listener
        val captureButton = findViewById<Button>(R.id.captureButton)
        captureButton.setOnClickListener {
            if (checkPermissions()) {
                startCamera()
                startLocationUpdates()
                startImageCapture()
            }
        }

        // Request necessary permissions
        requestPermissions()
    }

    private fun checkPermissions(): Boolean {
        val cameraPermission =
ContextCompat.checkSelfPermission(this,
Manifest.permission.CAMERA)
        val locationPermission =
ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
        return cameraPermission ==
PackageManager.PERMISSION_GRANTED && locationPermission
== PackageManager.PERMISSION_GRANTED
    }

    private fun requestPermissions() {
        val requestMultiplePermissions =

registerForActivityResult(ActivityResultContracts.RequestMultiplePer
missions()) { permissions ->
            permissions.entries.forEach {
                if (!it.value) {
                    Toast.makeText(this, "Permission ${it.key} denied",
Toast.LENGTH_SHORT).show()
                }
            }
        }


requestMultiplePermissions.launch(arrayOf(Manifest.permission.CAM
ERA, Manifest.permission.ACCESS_FINE_LOCATION))
    }

    private fun startCamera() {
        val cameraProviderFuture =
ProcessCameraProvider.getInstance(this)
        cameraProviderFuture.addListener({
            val cameraProvider: ProcessCameraProvider =
cameraProviderFuture.get()

            val preview = Preview.Builder().build()
                .also {

it.setSurfaceProvider(findViewById<androidx.camera.view.PreviewVie
w>(R.id.viewFinder).surfaceProvider)
                }

            imageCapture = ImageCapture.Builder().build()
```

```
            val cameraSelector =
CameraSelector.DEFAULT_BACK_CAMERA

            try {
                cameraProvider.unbindAll()
                cameraProvider.bindToLifecycle(this as LifecycleOwner,
cameraSelector, preview, imageCapture)
            } catch (exc: Exception) {
                Log.e("MainActivity", "Use case binding failed", exc)
            }

        }, ContextCompat.getMainExecutor(this))
    }

    private fun startLocationUpdates() {
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
            return
        }


locationManager.requestLocationUpdates(LocationManager.GPS_PR
OVIDER, 1000, 1f, object : LocationListener {
            fun onLocationChanged(location: Location) {
                gpsLocation = location
            }

            fun onStatusChanged(provider: String?, status: Int, extras:
Bundle?) {}
            fun onProviderEnabled(provider: String) {}
            fun onProviderDisabled(provider: String) {}
        })
    }

    private fun startImageCapture() {
        captureRunnable = object : Runnable {
            fun run() {
                captureImage()
                handler.postDelayed(this, 1000) // Capture every 1 second
            }
        }
        handler.post(captureRunnable)
    }

    private fun captureImage() {
        val imageCapture = imageCapture ?: return

        val photoFile =
File(getExternalFilesDir(Environment.DIRECTORY_PICTURES),
"${System.currentTimeMillis()}.jpg")

        val outputOptions =
ImageCapture.OutputFileOptions.Builder(photoFile).build()

        imageCapture.takePicture(outputOptions, cameraExecutor,
object : ImageCapture.OnImageSavedCallback {
            fun onImageSaved(outputFileResults:
ImageCapture.OutputFileResults) {
                Log.d("MainActivity", "Image saved:
${photoFile.absolutePath}")
                saveDataToCSV(photoFile.name)
            }

            fun onError(exception: ImageCaptureException) {
                Log.e("MainActivity", "Image capture failed:
${exception.message}", exception)
            }
        })
    }

    private fun saveDataToCSV(imageName: String) {
        val csvFile = File(getExternalFilesDir(null), "data.csv")

        val latitude = gpsLocation?.latitude ?: "Unknown"
        val longitude = gpsLocation?.longitude ?: "Unknown"

        val csvData = String.format(
            Locale.US, "%s,%s,%s,%f,%f,%f\n",
            imageName, latitude.toString(), longitude.toString(),
            accelerometerData[0], accelerometerData[1],
accelerometerData[2]
        )

        try {
            val writer = FileWriter(csvFile, true)
            writer.append(csvData)
            writer.close()
        } catch (e: IOException) {
            Log.e("MainActivity", "Error writing to CSV file", e)
        }
    }

        fun onDestroy() {
        super.onDestroy()
        cameraExecutor.shutdown()
        handler.removeCallbacks(captureRunnable)
    }
}
```

Image Processing:

https://github.com/Shantanu909/MobilTelesco/tree/main/src

Technology Survey:

Patent numbers:

US7982951B1 (Digital tracking platform for telescopes),

US10424045B2 (Machine learning model for automatic image registration quality assessment and correction)

US20230209169A1 (Method and Astrophotographic Apparatus for Acquiring Images of Targets in Sky Area)

US9191587B2 (Method and apparatus for image stacking)

Research Papers:

Ashley, J., 2015. Portable Observatories. In Astrophotography on the Go (pp. 239-261). Springer, Cham.

Baird, R., Gomez, M., Liddell, T. and Saunders, H., Astronomy Made Easy: An Overview of the Design and Fabrication of an All-In-One Stargazing Solution.

Falkner, D.E., 2020. Astrophotography Using a Compact Digital Camera or Smartphone Camera. In The Mythology of the Night Sky (pp. 269-294). Springer, Cham.

Biersdorfer, J.D., 2020. Get the Most Out of Your Fancy Smartphone Camera. International New York Times, pp.NA-NA.