Exploratory Notebook Implementation Plan

Overview

This document outlines the implementation plan for an initial exploratory notebook that:

1. Fetches source FITS files from available APIs/integrations
2. Computes differences from previous versions (reference images)
3. Prepares data for anomaly detection in subsequent notebooks
4. Explores options for sourcing ZTF anomalies or creating synthetic anomalies

Notebook Structure

Phase 1: Data Acquisition

Goal: Fetch FITS files from multiple sources and establish a baseline dataset

1.1 Source FITS File Acquisition

- **MAST API Integration** (`src/adapters/external/mast.py`)
    - Use `MASTClient.query_observations_by_position()` to find observations
    - Target sky regions:
        - Known transient regions (e.g., SN2011fe field, M101)
        - Deep field regions (COSMOS, GOODS-N)
        - Avoid galactic plane for cleaner images
    - Fetch observations from:
        - Pan-STARRS1 (PS1) - deep reference images
        - HST/JWST - high-quality deep images
        - TESS - time-series data
    - Download FITS files using `MASTClient.download_files()`
    - Store locally with metadata tracking

- **SkyView Integration** (`src/adapters/external/skyview.py`)
    - Use for on-demand image cutouts
    - Surveys: DSS2, SDSS, GALEX, 2MASS, WISE
    - Standardize image size (e.g., 240x240 pixels) and pixel scale
    - Create reference image library from multiple surveys

- **PS1 Cutout Helper** (`MASTClient.fetch_ps1_cutout()`)
    - Use for quick reference image fetching
    - Already implemented with FITS/JPEG fallback
    - Good for establishing baseline reference images

1.2 Data Organization

- Create directory structure:

```
notebooks/data/exploratory/
├── source_fits/          # Original FITS files
│   ├── science/          # Current observations
│   └── reference/        # Historical reference images
├── processed/            # Processed images
│   ├── aligned/          # WCS-aligned images
│   └── normalized/       # Normalized images
├── differences/          # Difference images
│   ├── zogy/             # ZOGY algorithm results
│   └── classic/          # Simple subtraction
└── metadata/             # JSON metadata files
```

- Track metadata for each observation:
  - Observation ID, RA/Dec, observation time
  - Survey/instrument, filter band
  - File paths, image dimensions
  - Processing status flags

Phase 2: Image Differencing

Goal: Compute differences between current and reference images

2.1 Image Preprocessing

- **FITS I/O** (`src/adapters/imaging/fits_io.py`)
  - Use `FITSProcessor.read_fits()` to load images
  - Extract WCS information for alignment
  - Extract metadata from headers
- **Image Alignment**
  - Use WCS information from `src/domains/preprocessing/processors/wcs_processor.py`
  - Align images to common pixel grid
  - Handle different pixel scales and orientations
  - Use `astropy` coordinate transformations
- **Normalization**
  - Background subtraction
  - Flux normalization
  - Handle different exposure times and zero points
  - Store normalization parameters

2.2 Difference Computation

- **ZOGY Algorithm** (`src/domains/preprocessing/processors/astronomical_image_processing.py`)
    - Use `ImageDifferencingProcessor.zogy_differencing()`
    - Input: science image, reference image
    - Optional: PSF models, noise maps
    - Output: difference image + metrics (significance map, SNR)
- **Classic Differencing** (baseline)
    - Simple subtraction for comparison
    - Use `ImageDifferencingProcessor.classic_differencing()`
- **Difference Image Quality Assessment**
    - Calculate noise properties
    - Identify artifacts (cosmic rays, bad pixels)
    - Compute quality metrics:
        - Noise level (std dev of background)
        - Dynamic range
        - Detection threshold (SNR)

2.3 Difference Image Storage

- Save difference images as FITS files
- Include metadata:
    - Original observation IDs
    - Processing parameters (PSF, noise estimates)
    - Quality metrics
    - Timestamps

Phase 3: Anomaly Preparation

Goal: Prepare difference images for anomaly detection

3.1 Source Extraction

- Use `SEP` or `photutils` for source detection on difference images
- Extract candidate sources:
    - Position (x, y and RA/Dec)
    - Flux and significance (SNR)
    - Shape parameters (ellipticity, size)
    - Quality flags
- Filter candidates:
    - Minimum SNR threshold (e.g., 5σ)
    - Exclude known artifacts (bad pixels, diffraction spikes)
    - Size constraints (point sources vs extended)

3.2 Image Cutouts

- Extract cutouts around each candidate:
    - Science image cutout
    - Reference image cutout
    - Difference image cutout
    - Standard size (e.g., 64x64 or 128x128 pixels)
    - Include padding for context
- Store cutouts:
    - As individual FITS files or
    - In a multi-extension FITS file
    - Include metadata (position, SNR, timestamp)

3.3 Data Format for Next Notebook

- Create structured dataset:
    - NumPy arrays or PyTorch tensors
    - Image triplets: (science, reference, difference)
    - Labels: (anomaly flag, source type, confidence)
    - Metadata: (coordinates, timestamps, quality metrics)
- Export formats:
    - HDF5 file for large datasets
    - Pickle/numpy format for smaller sets
    - CSV metadata file
    - JSON configuration file

Phase 4: Anomaly Data Sources

Goal: Explore options for real and synthetic anomalies

4.1 ZTF Database Options

Option A: ZTF Public Data Release

- **ZTF Public Data Portal** (IRSA)
    - Access: https://irsa.ipac.caltech.edu/Missions/ztf.html
    - Public data releases with 6-month delay
    - Difference images and alert packets
    - Light curves for confirmed transients
- **ZTF Alert Stream** (via brokers)
    - Kowalski database (MongoDB) - query interface
    - ZTF alert schema with image cutouts

- Requires: API access, database connection
- Filter by: object type, magnitude, location

- **ZTF Data Products**

  - Difference images: science - reference
  - Science image stamps
  - Reference image stamps
  - Metadata: RA/Dec, mag, filter, MJD

Option B: ZTF via API

- Query ZTF public catalogs:
  - Use `astroquery` or direct API calls
  - Filter confirmed transients/supernovae
  - Download associated image stamps
  - Cross-match with known object catalogs

Option C: ZTF Simulation/Reconstruction

- Use ZTF-like parameters:
  - Instrument characteristics (PSF, pixel scale)
  - Survey strategy (cadence, depth)
  - Generate synthetic ZTF-like images

4.2 Synthetic Anomaly Generation

Leverage existing code (`standalone_training.py`)

- **SyntheticAnomalicalDataset** (lines 359-508)

  - Already generates synthetic astronomical images
  - Anomaly types: transient, variable, supernova, asteroid
  - Configurable anomaly ratio and noise levels

- **Enhancement for Difference Images**:

  - Generate reference image (steady state)
  - Generate science image with injected anomaly
  - Compute difference image
  - Vary anomaly parameters:
    - Brightness (magnitude range)
    - Size (point source vs extended)
    - Position (random vs known locations)
    - Type (SN Ia, SN II, nova, etc.)

- **Realistic Anomaly Injection**:

  - Add real transient light curves
  - Use PSF models from real surveys
  - Include realistic noise characteristics

    - Match ZTF-like image properties

4.3 Hybrid Approach

- Start with synthetic anomalies for validation
- Gradually incorporate real ZTF data as available
- Use synthetic data for:
    - Training anomaly detection models
    - Testing pipeline robustness
    - Handling edge cases
- Use real ZTF data for:
    - Validation and benchmarking
    - Understanding real-world challenges
    - Model generalization testing

Phase 5: Data Validation and Quality Control

Goal: Ensure data quality before anomaly detection

5.1 Image Quality Checks

- Verify FITS file integrity
- Check image dimensions and data types
- Validate WCS solutions
- Check for NaN/inf values

5.2 Difference Image Validation

- Verify alignment quality (residuals)
- Check noise characteristics (should be Gaussian)
- Identify and flag artifacts
- Validate source extraction results

5.3 Metadata Completeness

- Ensure all required metadata present
- Check coordinate consistency
- Validate timestamps
- Verify cross-references (science ↔ reference)

Phase 6: Visualization and Exploration

Goal: Visualize results and understand data characteristics

6.1 Image Visualization

- Display science, reference, and difference images
- Show source extraction results (overlay)
- Create comparison plots (ZOGY vs classic)

- Generate quality assessment plots

6.2 Statistical Analysis

- Noise distribution analysis
- Source detection statistics
- Anomaly candidate distribution
- Quality metrics distribution

6.3 Data Summary

- Generate summary statistics
- Create data quality report
- Document known issues/limitations
- Prepare data catalog for next notebook

Implementation Details

Notebook Sections

1. **Setup and Configuration**
    - Import libraries
    - Set up paths and directories
    - Configure API clients
    - Set random seeds for reproducibility
2. **Data Acquisition**
    - Define target sky regions
    - Fetch observations from MAST/SkyView
    - Download FITS files
    - Organize and catalog files
3. **Image Processing**
    - Load FITS files
    - Align images using WCS
    - Normalize images
    - Quality assessment
4. **Image Differencing**
    - Compute ZOGY differences
    - Compute classic differences (baseline)
    - Calculate quality metrics
    - Save difference images
5. **Source Extraction**
    - Detect sources in difference images

  - Filter candidates
  - Extract cutouts
  - Create candidate catalog

6. **Anomaly Data Preparation**

   - Option 1: Query ZTF database (if available)
   - Option 2: Generate synthetic anomalies
   - Option 3: Hybrid approach
   - Prepare training/validation dataset

7. **Data Export**

   - Save processed images
   - Export metadata
   - Create dataset for next notebook
   - Generate summary report

8. **Visualization**

   - Display example images
   - Show difference images
   - Plot source detection results
   - Create quality assessment plots

Technical Requirements

Dependencies

- Existing AstrID modules:
  - `src/adapters/external/mast.py` - MAST API client
  - `src/adapters/external/skyview.py` - SkyView client
  - `src/adapters/imaging/fits_io.py` - FITS I/O
  - `src/domains/preprocessing/processors/astronomical_image_processing.py` - ZOGY differencing
  - `src/domains/preprocessing/processors/fits_processing.py` - FITS processing
- External packages:
  - `astropy` - FITS, WCS, coordinates
  - `astroquery` - MAST queries
  - `numpy`, `matplotlib` - Data handling, visualization
  - `sep` or `photutils` - Source extraction
  - `scipy` - Image processing

Configuration Options

- Target sky regions (RA/Dec, radius)
- Survey selection (MAST missions, SkyView surveys)
- Image size and pixel scale
- Differencing algorithm parameters
- Source detection thresholds

- Anomaly generation parameters (if synthetic)

Deliverables

1. **Jupyter Notebook** (`notebooks/exploratory_data_preparation.ipynb`)

    - Complete implementation of all phases
    - Well-documented with markdown cells
    - Clear section organization

2. **Data Directory Structure**

    - Organized FITS files
    - Processed images
    - Difference images
    - Metadata files

3. **Dataset for Next Notebook**

    - Processed image triplets (science, reference, difference)
    - Candidate source catalog
    - Anomaly labels (if available)
    - Metadata and configuration files

4. **Documentation**

    - README for the notebook
    - Data format specification
    - Known issues and limitations
    - Next steps documentation

Next Steps (Subsequent Notebook)

The output from this exploratory notebook will feed into:

- **Anomaly Detection Notebook**
    - Load prepared dataset
    - Apply U-Net model for anomaly detection
    - Evaluate detection performance
    - Visualize results
    - Compare with ground truth (if available)

Notes

- Start small: Begin with 10-20 observations to validate pipeline
- Use existing code: Leverage `standalone_training.py` for synthetic data
- API rate limits: Be mindful of MAST/SkyView rate limits
- Data storage: Consider disk space for FITS files
- Reproducibility: Save all random seeds and configuration
- Error handling: Robust error handling for API failures

- Progress tracking: Log progress for long-running operations