Automated Star Identification System

From Image Input to Labeled Star Map

Mohamad,Ghia,Hamad

Ariel University

June 21, 2025

Presentation Outline

What Does This System Do?

Input: A photo of the night sky

Output: The same photo with stars labeled by name

Think of it as "Shazam for Stars"

- Take any astronomical image
- Automatically identify which stars are visible
- Label them with proper names and information
- No manual star charts needed!

Why Is This Challenging?

Human Challenges:

- Thousands of stars visible
- Stars look very similar
- Need extensive astronomy knowledge
- Time-consuming process

Computer Challenges:

- Distinguish stars from noise
- Determine viewing direction
- Match patterns to database
- Handle image variations

Solution: Break into 6 manageable steps!

The 6-Step Algorithm Pipeline

1. Star Detection — Find bright spots
2. Plate Solving — Where am I looking?
3. Catalog Loading — Get star database
4. Coordinate Transform — Pixel $\rightarrow$ Sky coords
5. Star Matching — Match detections

Step 1: Star Detection

Goal: Find all bright objects that could be stars

Algorithm Steps:

1. Convert photo to grayscale
2. Apply Gaussian blur (reduce noise)
3. Binary thresholding (bright vs dark)
4. Find contours (connected bright regions)
5. Filter by size (remove noise & artifacts)

Real-world analogy: Like using a highlighter to mark all bright spots on a printed photo

Visual Process:

Original Image

↓

Grayscale

↓

Blur

↓

Threshold

↓

Detected Stars

Output: List of (x,y) coordinates where stars are detected

Star Detection: Key Parameters

- **Threshold Value (120):** How bright must a spot be?
  - Lower $\rightarrow$ detect fainter stars (more noise)
  - Higher $\rightarrow$ only brightest stars (miss faint ones)
- **Minimum Area (1 pixel):** Smallest acceptable star
  - Filters out single-pixel noise
- **Maximum Area (800 pixels):** Largest acceptable star
  - Removes planets, satellites, defects
- **Gaussian Blur (3×3):** Noise reduction
  - Smooths image while preserving star shapes

Typical Result: 20-100 detected bright spots per image

Step 2: Plate Solving (Astrometric Calibration)

Goal: Determine which part of the sky we're looking at

Method 1 - Astrometry.net API:

1. Upload image to online service
2. Service compares star patterns
3. Returns sky coordinates & field of view
4. Like fingerprint matching!

Most accurate method

Method 2 - Filename Patterns:

1. Check filename for keywords
2. "orion.jpg" → Orion coordinates
3. "m42_nebula.png" → M42 coordinates
4. Use pre-programmed locations

Fallback when API fails

Real-world analogy: Like using GPS to find your location, but for space photos

Output: Image center coordinates (RA, Dec) and field of view radius

Why Plate Solving Is Crucial

Without knowing WHERE you're looking...

You can't match detected spots to known stars!

What we get:

- **RA (Right Ascension):** Like longitude for space (0-360°)
- **Dec (Declination):** Like latitude for space (-90° to +90°)
- **Field of View:** How much sky the image covers

Example Results:

- Orion constellation: RA = 85°, Dec = -1°, FOV = 15°
- Pleiades cluster: RA = 57°, Dec = 24°, FOV = 5°

Step 3: Star Catalog Loading

Goal: Get list of known stars that should be visible in our image
The Database: Yale Bright Star Catalog v5

- Contains 9,000 brightest stars
- For each star: name, position, brightness, spectral type
- Covers entire sky down to magnitude 6.5

Search Process:

1. "Give me all stars within 15° of (85°, -1°)"
2. Database returns 80-200 stars in region
3. Sort by brightness (magnitude)

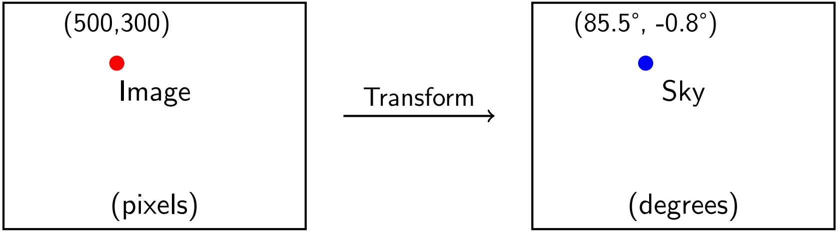
Sample Star Data:

Name	Sirius
HR Number	2491
RA	101.29°
Dec	-16.72°
Magnitude	-1.46
Type	A1V

Real-world analogy: Like opening a phone book for your neighborhood

Step 4: Coordinate Transformation

Goal: Convert detected pixel positions to sky coordinates



Mathematical Process:

1. Calculate pixel scale: $\frac{\text{field of view}}{\text{image size}}$ degrees/pixel
2. Find offset from image center
3. Convert pixel offset to angular offset
4. Apply spherical coordinate correction
5. Add to image center coordinates

Real-world analogy: Like converting "3 inches right on map" to "50 miles east in reality"

Coordinate Transformation: The Math

Key Equations:

$$\text{Pixel Scale} = \frac{\text{Field of View} \times 2}{\min(\text{width}, \text{height})} \text{ deg/pixel} \tag{1}$$

$$\text{Angular Offset}_x = (\text{pixel}_x - \text{center}_x) \times \text{Pixel Scale} \tag{2}$$

$$\text{Angular Offset}_y = -(\text{pixel}_y - \text{center}_y) \times \text{Pixel Scale} \tag{3}$$

$$\text{Star RA} = \text{Image RA} + \frac{\text{Angular Offset}_x}{\cos(\text{Dec})} \tag{4}$$

$$\text{Star Dec} = \text{Image Dec} + \text{Angular Offset}_y \tag{5}$$

Important Notes:

- Y-axis is flipped (image vs sky coordinates)

Coordinate Transformation: Example

Given: Orion image, 1000×800 pixels, FOV = 15°, center at RA=85°, Dec=-1°

Step 1: Calculate pixel scale

$$\text{Pixel Scale} = \frac{15 \times 2}{\min(1000, 800)} = \frac{30}{800} = 0.0375 \text{ deg/pixel} \tag{6}$$

Step 2: Find a detected star at pixel (650, 300)

$$\text{Angular Offset}_x = (650 - 500) \times 0.0375 = 150 \times 0.0375 = 5.625 \tag{7}$$

$$\text{Angular Offset}_y = -(300 - 400) \times 0.0375 = -(-100) \times 0.0375 = 3.75 \tag{8}$$

Step 3: Convert to sky coordinates

$$\text{Star RA} = 85 + \frac{5.625}{\cos(-1)} = 85 + 5.628 = 90.628 \tag{9}$$

$$\text{Star Dec} = -1 + 3.75 = 2.75 \tag{10}$$

Result: Pixel (650,300) → Sky coordinates (90.63°, 2.75°)

This could be Betelgeuse at RA=88.8°, Dec=7.4°!

Algorithm Evolution: Part 1 vs Part 2

Part 1: Geometric Pattern Matching

- Used triangle and quadrilateral patterns
- Matched star shapes between images
- Calculated normalized side lengths
- Robust to rotation and scaling
- Complex voting system

Advantages:

- Works without sky coordinates
- Handles image transformations
- More sophisticated matching

Part 2: Coordinate-Based Matching

- Uses sky coordinates (RA/Dec)
- Matches by angular distance
- Simple nearest-neighbor algorithm
- Relies on plate solving
- Direct star-to-catalog matching

Advantages:

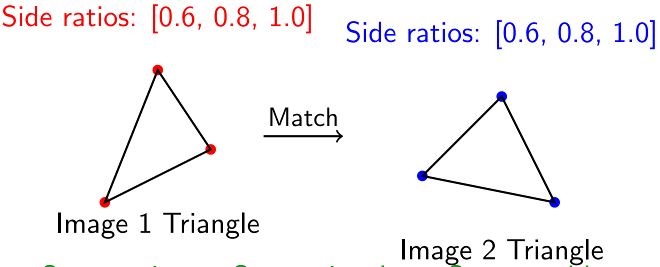
- Much faster processing
- Simpler to understand
- Works with star catalogs
- Real astronomical coordinates

Evolution: From image-to-image matching → image-to-catalog matching

Triangle Algorithm (Part 1) - How It Worked

Geometric Pattern Matching Process:

1. **Find Neighbors:** For each star, find k=5 nearest neighbors
2. **Form Triangles:** Create triangles from star + 2 neighbors
3. **Calculate Distances:** Measure all 3 side lengths
4. **Normalize:** Divide by longest side → pattern ratios
5. **Match Patterns:** Compare triangle ratios between images
6. **Vote:** Triangles vote for star correspondences



Same ratios → Same triangle → Star match!

Why We Changed Approaches

Challenges with Triangle Matching:

- Computationally expensive: $O(n^3)$ complexity for triangles
- Image-to-image only: Couldn't match to star catalogs
- Complex voting: Multiple algorithms, harder to debug
- No real star names: Just matched pixel patterns

Benefits of Coordinate Matching:

- Real astronomy: Uses actual sky coordinates
- Star identification: Get real star names and data
- Faster: $O(m \times n)$ where m,n are much smaller
- Simpler: Easier to understand and modify
- Catalog integration: Works with professional databases

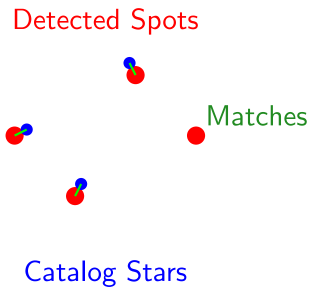
Trade-off: Sacrificed geometric robustness for astronomical accuracy

Step 5: Star Matching Algorithm (Current Approach)

Goal: Match detected bright spots with known catalog stars

Algorithm: Greedy Nearest-Neighbor

1. Take brightest catalog star
2. Find closest detected spot within 0.8°
3. If match found, pair them up
4. Remove both from available pools
5. Repeat with next brightest catalog star



Star Matching: Key Considerations

Why 0.8° threshold?

- Accounts for measurement errors
- Coordinate conversion uncertainties
- Star catalog precision limits
- Atmospheric effects

Why brightest stars first?

- More likely to be correctly detected
- Higher confidence matches
- Reduces false positives

One-to-one matching:

- Each detected spot matched to at most one catalog star
- Prevents duplicate assignments
- Ensures unique identifications

Typical Result: 2-10 successful matches per image (3-5% success rate)

Step 6: Image Annotation

Goal: Create beautiful labeled star map

Visual Elements:

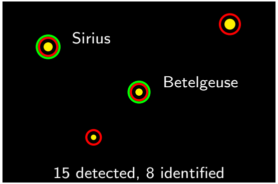
- **Red circles:** All detected spots
- **Green circles:** Identified stars
- **Yellow labels:** Star names & info
- **Arrows:** Connect labels to stars
- **Title:** Summary statistics

Information Displayed:

- Star name (e.g., "Sirius")
- Harvard Revised number (HR2491)
- Visual magnitude (-1.46)
- Total detection/identification counts

Real-world analogy: Like adding captions to a photo

Sample Output:



System Performance & Results

Typical Performance:

- **Input:** 50-200 visible stars
- **Detected:** 20-80 bright spots
- **Identified:** 3-10 named stars
- **Success Rate:** 3-5%
- **Processing Time:** 40-90 seconds

Best Results With:

- Clear, focused star images
- 5-20° field of view
- Bright stars (magnitude ¡ 4)
- Good contrast photos

Limitations:

- Only 9,000 brightest stars in catalog
- Requires internet for best plate solving
- Struggles with very wide/narrow fields
- Can't identify faint deep-sky objects

Common Issues:

- Overexposed images
- Out-of-focus stars
- Light pollution
- Clouds or atmospheric haze

Overall: Excellent tool for amateur astronomy and education!

Real-World Applications

Educational:

- Astronomy classes
- Planetarium shows
- Student projects
- Public outreach

Amateur Astronomy:

- Astrophotography labeling
- Observation planning
- Star party activities
- Equipment testing

Professional:

- Telescope automation
- Camera calibration
- Archive processing
- Research applications

Fun Applications:

- Social media posts
- Travel photography
- Citizen science

Making astronomy accessible to everyone!

Technical Architecture

System Components:

- **star_algorithm.py:** Core algorithmic logic
  - All 6 algorithm steps implemented here
  - Pure computational functions
  - No web interface dependencies
- **flask_star_identifier.py:** Web server wrapper
  - HTTP endpoints for image upload
  - File handling and validation
  - JSON result formatting
  - User interface integration

Key Technologies:

- **OpenCV:** Computer vision and image processing
- **NumPy/Pandas:** Numerical computations and data handling
- **SQLite:** Star catalog database
- **Flask:** Web framework for user interface
- **Astrometry.net:** Professional plate solving service

Key Takeaways

What We Accomplished:

- Automated star identification from photos
- Robust 6-step algorithmic pipeline
- Web-based user interface
- Real-world practical applications

Algorithm Strengths:

- Combines multiple approaches (CV + databases + web APIs)
- Handles real-world challenges (noise, errors, failures)
- Provides useful results (3-5% identification rate)
- Easy to use (just upload a photo!)

Future Improvements:

- Larger star catalogs (Gaia, Hipparcos)
- Better coordinate transformations
- Machine learning enhancement

Questions & Discussion

Thank You!

Questions?

Code available at:
Your GitHub repository