MicroscoPilot

COMPLETE PROJECT EXPLANATION

What is MicroscoPilot?

Imagine you're a scientist with a super expensive microscope that can see things at the atomic scale (like looking at materials atom by atom). Using this microscope is:

- **Time-consuming:** Scanning the entire sample takes hours
- **Manual:** A human has to decide where to look next
- **Inefficient:** 70% of time is spent looking at boring/empty regions
- **Expensive:** Microscope time costs $500/hour at universities

MicroscoPilot solves this by making the microscope SMART and AUTONOMOUS. It's like giving the microscope a robot brain that can:

1. **Look** at what it's seeing (using AI vision)
2. **Think** about what's interesting (using machine learning)
3. **Decide** where to go next (using intelligent exploration)
4. **Learn** from experience (building memory of what it's seen)

The Big Picture: How Everything Works Together

```
┌─────────────────────────────────────────────────────────┐
│                      HUMAN USER                         │
│             "Find me interesting defects"               │
└─────────────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────────────┐
│                 MAIN.PY (Orchestrator)                  │
│            Coordinates all the components below         │
└─────────────────────────────────────────────────────────┘
        │          │          │          │
        ▼          ▼          ▼          ▼
   ┌────────┐ ┌────────┐ ┌────────┐ ┌──────────┐
   │ AGENT  │ │VISION  │ │MICRO-  │ │VISUALIZER│
   │        │ │        │ │SCOPE   │ │          │
   │ BRAIN  │ │  EYES  │ │ CONTROL│ │  DISPLAY │
   └────────┘ └────────┘ └────────┘ └──────────┘
       │          │          │          │
       │          │          │          │
```

```
        └────────────┴────────────┴────────────┘
                      │
                      ▼
                 ┌────────────────┐
                 │    MEMORY      │
                 │   (What we     │
                 │    learned)    │
                 └────────────────┘
```

Component-by-Component Explanation

1. MICROSCOPE CONTROLLER (microscope.py) - The Arms & Legs

What it does: Interfaces with the microscope hardware (or simulator)

Think of it as: The steering wheel and pedals of a car. You tell it "go to position (X, Y)" and it moves there.

Key Methods:

- `move_to(x, y)` - Move the microscope stage to a location
- `capture_image()` - Take a picture at current location
- `zoom_in/out()` - Adjust magnification
- `get_state()` - Where are we? What are our settings?

Example in real life:

python

```python
microscope = MicroscopeController()
microscope.move_to(100, 200)  # Move to coordinates (100nm, 200nm)
image = microscope.capture_image()  # Take picture
# Now we have an image to analyze!
```

Why it exists: Your agent needs to CONTROL the microscope. This module provides a clean, simple interface to do complex microscope operations.

---

2. VISION ANALYZER (vision.py) - The Eyes & Brain

What it does: Uses Claude's AI to "see" and understand microscope images

Think of it as: A expert microscopist who looks at images and tells you what they see

The Magic - Prompt Engineering:

python

```
prompt = """
You are analyzing an Atomic Force Microscopy (AFM) image.
Look for:
- Grain boundaries (lines where crystal regions meet)
- Defects (imperfections in the material)
- Particles (small objects on surface)
- Smooth vs rough regions

Rate the image:
- Quality (0-10): Is it clear? Good contrast?
- Interestingness (0-10): Are there features worth investigating?

Suggest next action: Should we zoom in? Move to a different area?

Provide your analysis in this format:
{
  "features": ["grain_boundary", "defect"],
  "quality": 8.5,
  "interestingness": 9.0,
  "next_action": "zoom_in",
  "reasoning": "I see a prominent grain boundary with what appears to be a
defect nearby. Zooming in would reveal more detail about the defect
structure."
}
"""
```

Why this is powerful:

- Claude has been trained on millions of images
- It can recognize patterns humans might miss
- It "explains" its reasoning (interpretable AI)
- NO TRAINING DATA NEEDED (zero-shot learning)

Example:

python

```
vision = VisionAnalyzer(api_key="your_key")
analysis = vision.analyze_image("data/sample.png")
```

```
# Returns: {"features": [...], "quality": 8.5, ...}

if analysis['interestingness'] > 7:
    print("🎯 Found something interesting!")
```

3. AGENT (agent.py) - The Decision-Making Brain

What it does: The autonomous "pilot" that makes decisions

Think of it as: A self-driving car's AI, but for microscopes

The Core Loop:

python

```
while not done:
    # 1. SENSE: Where am I? What do I see?
    current_position = microscope.get_current_position()
    image = microscope.capture_image()

    # 2. ANALYZE: What's in this image?
    analysis = vision.analyze_image(image)

    # 3. REMEMBER: Store what we learned
    memory.store(current_position, analysis)

    # 4. THINK: What should I do next?
    next_action = decide_next_action(analysis, current_position)

    # 5. ACT: Execute the decision
    if next_action['type'] == 'move':
        microscope.move_to(next_action['x'], next_action['y'])
    elif next_action['type'] == 'zoom':
        microscope.zoom_in()

    # 6. LEARN: Update our strategy
    update_exploration_strategy()
```

Decision Logic (Simple Version):

python

```python
def decide_next_action(analysis, position):
    # If we found something REALLY interesting → Zoom in for detail
    if analysis['interestingness'] > 8:
        return {'type': 'zoom'}

    # If we've been here before → Skip it, go somewhere new
    if memory.has_visited(position):
        return {'type': 'move', 'x': random_new_location()}

    # If image quality is poor → Try adjusting settings
    if analysis['quality'] < 5:
        return {'type': 'adjust_settings'}

    # If nothing interesting lately → Jump to random area
    if steps_since_discovery > 20:
        return {'type': 'move', 'x': random_location()}

    # Otherwise → Move to nearby area
    return {'type': 'move', 'x': position[0] + 50, 'y': position[1]}
```

Advanced Decision Logic (ML Version - Day 2):

- **Exploitation vs Exploration:** 80% time go to "promising" areas, 20% explore new regions
- **Pattern Recognition:** "When I find grain boundaries, defects are usually nearby"
- **Active Learning:** Focus on areas where AI is uncertain

4. MEMORY (memory.py) - The Learning System

What it does: Stores what the agent has learned

Think of it as: The agent's notebook where it writes down everything it discovers

Three Types of Memory:

A) Spatial Memory (Map):

python

```python
# Divides sample into grid
# Each cell remembers:
spatial_memory = {
    (100, 100): {
        'visits': 3,
        'best_quality': 8.5,
        'features_found': ['grain_boundary'],
        'avg_interestingness': 7.2
    },
    (200, 200): {
        'visits': 1,
        'best_quality': 6.0,
        'features_found': [],
        'avg_interestingness': 3.0
    }
}
```

B) Episodic Memory (Timeline):

python

```python
# List of important events:
episodes = [
    {
        'time': '10:30:15',
        'position': (150, 200),
        'event': 'found_defect',
        'confidence': 0.9,
        'image_path': 'outputs/img_023.png'
    },
    {
        'time': '10:31:02',
        'position': (155, 205),
        'event': 'found_grain_boundary',
        'confidence': 0.85
    }
]
```

C) Semantic Memory (Facts):

python

```python
# High-level knowledge:
learned_facts = [
    "Top-right quadrant (x>700, y>700) has high defect density",
    "Smooth regions are in bottom-left (x<300, y<300)",
    "Grain boundaries run diagonally across sample",
    "Optimal zoom level for defects: 1000x"
]
```

Why Memory Matters:

python

```python
# WITHOUT MEMORY:
agent.explore()
# Visits (100, 100) → "Cool! A defect!"
# Visits (200, 200) → "Boring, nothing here"
# Visits (100, 100) again → "Cool! A defect!" ← WASTE OF TIME!

# WITH MEMORY:
agent.explore()
# Visits (100, 100) → "Cool! A defect!" → SAVES TO MEMORY
# Visits (200, 200) → "Boring" → SAVES TO MEMORY
# Checks (100, 100) → "Already been here" → SKIPS ← EFFICIENT!
```

5. VISUALIZER (visualizer.py) - The Dashboard

What it does: Shows what the agent is doing in real-time

Think of it as: Mission Control dashboard for the agent

4-Panel Display:

```
┌──────────────────────┬──────────────────────┐
│  CURRENT IMAGE       │  EXPLORATION MAP     │
│                      │                      │
│  [Microscope View]   │  [Heatmap showing    │
│   • Current image    │   where agent has    │
│   • Features marked  │   been]              │
│   • Position shown   │   • Blue = new       │
│                      │   • Red = visited    │
│                      │     many times       │
├──────────────────────┼──────────────────────┤
```

```
│  DISCOVERIES          │  STATISTICS          │
│                       │                      │
│  [Timeline Graph]     │  Steps: 47           │
│   Showing when        │  Coverage: 68%       │
│   interesting         │  Features: 12        │
│   features were       │  Efficiency: 0.26    │
│   found over time     │  Strategy: Smart     │
│                       │  Time: 5:23          │
└───────────────────────┴──────────────────────┘
```

Why it's CRITICAL:

- **For Development:** You can SEE if your agent is working
- **For Debugging:** Spot problems visually (stuck in loops? missing areas?)
- **For Demos:** Judges NEED to see your agent in action
- **For WOW Factor:** Live animations are impressive!

Example Usage:

python

```python
visualizer = Visualizer()
visualizer.update(
    current_image=image,
    position=(150, 200),
    discoveries=[...],
    stats={...}
)
visualizer.show()  # Opens window with live dashboard
```

The Machine Learning Concepts (Simplified)

1. Vision AI (What you're using)

Normal ML: Train a model on 10,000 labeled images of defects → Model learns to detect defects **Your approach:** Use Claude (already trained on millions of images) → Just give it good instructions via prompts

Analogy: Instead of teaching a dog new tricks, you hired a trained dog and just give it commands

Advantage: NO TRAINING DATA NEEDED! Works immediately!

2. Reinforcement Learning (Optional for Day 2)

What it is: Learning from trial and error

How it works:

python

```python
# Agent tries action
agent.move_to(random_location())

# Gets reward/penalty
if found_interesting_feature:
    reward = +10
else:
    reward = -1

# Learns: "Moving to X was good, moving to Y was bad"
# Next time: More likely to move to places like X
```

Analogy: Teaching a dog - give treats for good behavior, dog learns what gets treats

Your use: Agent learns which exploration strategies find more features

3. Bayesian Optimization (Smart Exploration)

What it is: Balancing "exploitation" (use what you know) vs "exploration" (try new things)

Example:

- You're at a buffet
- **Exploitation:** Keep eating the pizza you know is good
- **Exploration:** Try the mystery curry - might be amazing or terrible
- **Bayesian:** 80% pizza, 20% try new dishes

Your use:

python

```python
if random.random() < 0.8:
    # 80% time: Go to regions NEAR previous discoveries
```

```
        next_pos = near_known_good_region()
    else:
        # 20% time: Try completely new area
        next_pos = random_unexplored_region()
```

4. Pattern Recognition

What it is: Agent learns correlations

Example:

python

```
# After 100 observations, agent notices:
# "When I find grain_boundary, there's a 75% chance
#  a defect is within 50nm"

# So when it finds grain_boundary:
search_nearby_for_defects()
```

Analogy: You learn "when it's cloudy, it often rains" → You bring umbrella when cloudy

5. Active Learning

What it is: Focus effort where you're most uncertain

Example:

python

```
# Image quality = 9, confidence = 0.95 → "I'm sure this is smooth"
# → Skip it, don't need more scans

# Image quality = 6, confidence = 0.55 → "Unclear what this is"
# → Come back with different settings, try to clarify
```

Analogy: In an exam, you skip easy questions, spend time on hard ones