

# WebSocket Auto-Detection Feature

---

## Overview

WebSocket consumer sekarang memiliki fitur **auto-detection** yang otomatis melakukan prediksi ketika ikan terdeteksi dalam **3 frame berturut-turut**, tanpa perlu action **analyze** atau **capture** manual.

## How It Works

### 1. Quick Detection (Lightweight)

- Setiap frame yang masuk akan di-check dengan detection model saja
- Tidak melakukan classification (lebih cepat)
- Hanya check: "Ada ikan atau tidak?"

### 2. Detection Buffer

- Sistem menyimpan hasil deteksi dalam buffer (max 5 frame)
- Tracking apakah ada ikan di setiap frame

### 3. Consecutive Detection Threshold

- **Default: 3 frame berturut-turut**
- Ketika ikan terdeteksi di 3 frame berturut-turut → Auto-trigger full recognition
- Full recognition = Detection + Classification + LLM (lengkap)

### 4. Auto-Clear Buffer

- Setelah recognition berhasil, buffer di-clear
- Menghindari recognition berulang untuk ikan yang sama

## Message Flow

Input (Client → Server)

```
{
  "type": "camera_frame",
  "data": {
    "frame_data": "base64_encoded_image",
    "frame_id": 123,
    "include_faces": true,
    "include_segmentation": true,
    "include_visualization": true
  }
}
```

## Output: Detection Status (Server → Client)

Setiap frame akan mengirim status deteksi:

```
{
  "type": "detection_status",
  "data": {
    "has_fish": true,
    "consecutive_count": 2,
    "threshold": 3,
    "buffer": [true, true, false]
  },
  "timestamp": "2025-12-08T14:45:00.123Z"
}
```

## Output: Auto Recognition Result (Server → Client)

Ketika threshold tercapai (3 frame dengan ikan):

```
{
  "type": "recognition_result",
  "data": {
    "frame_id": 125,
    "processing_time": 5.23,
    "timestamp": "2025-12-08T14:45:05.456Z",
    "source": "auto_detection",
    "trigger": "fish_detected_3_frames",
    "results": {
      "fish_detections": [...],
      "classification": [
        {
          "name": "Ikan Bandeng",
          "scientific_name": "Chanos chanos",
          "accuracy": 0.95,
          "source": "llm",
          "species_id": -1
        },
        ...
      ],
      "llm_verification": {...},
      "visualization_image": "base64_image"
    }
  }
}
```

## Configuration

### Client Settings

```
{
  "type": "settings_update",
  "data": {
    "auto_detection_enabled": true, // Enable/disable auto-detection
    "include_visualization": true, // Include annotated image
    "include_faces": true,
    "include_segmentation": true,
    "processing_mode": "accuracy" // "accuracy" or "speed"
  }
}
```

## Server-Side Settings (in consumer)

```
self.consecutive_fish_threshold = 3 # Consecutive frames needed
self.max_buffer_size = 5           # Max detection buffer
self.min_processing_interval = 0.5 # Throttle between recognitions
```

## Behavior Comparison

### ✗ OLD (Manual Mode)

1. Client streams frames → Server stores frames
2. Client sends "analyze" action → Server checks for fish
3. Client sends "capture" action → Server does full recognition
4. **Problem:** Requires manual actions, delayed response

### ✓ NEW (Auto-Detection Mode)

1. Client streams frames → Server does quick detection
2. Fish detected in 3 consecutive frames → **Auto-trigger recognition**
3. **Benefit:** Automatic, real-time, no manual actions needed

## Performance

### Quick Detection (per frame)

- **Speed:** ~50-100ms (detection only, no classification)
- **CPU:** Low (only YOLO detection)
- **Memory:** Minimal

### Full Recognition (when triggered)

- **Speed:** ~5-6 seconds (detection + classification + LLM)
- **CPU:** High (all models + LLM inference)
- **Memory:** Moderate

## Throttling

- Minimum 0.5 seconds between full recognitions
- Prevents overwhelming the system
- Ensures quality over quantity

## Use Cases

### Real-Time Camera Stream

```
const ws = new WebSocket('ws://localhost:8000/ws/recognition/');

ws.onopen = () => {
  // Enable auto-detection
  ws.send(JSON.stringify({
    type: 'settings_update',
    data: {
      auto_detection_enabled: true,
      include_visualization: true
    }
  }));

  // Start streaming frames
  setInterval(() => {
    const frame = captureFrame(); // Your camera capture logic
    ws.send(JSON.stringify({
      type: 'camera_frame',
      data: {
        frame_data: frame,
        frame_id: Date.now()
      }
    }));
  }, 100); // Stream at ~10 FPS
};

ws.onmessage = (event) => {
  const msg = JSON.parse(event.data);

  if (msg.type === 'detection_status') {
    // Show detection indicator
    console.log(`Fish detected: ${msg.data.has_fish}`);
    console.log(`Consecutive:
    ${msg.data.consecutive_count}/${msg.data.threshold}`);
  }

  if (msg.type === 'recognition_result') {
    // Auto-triggered recognition!
    console.log('🐟 Fish recognized:', msg.data.results);
    displayResults(msg.data.results);
  }
};
```

## Disable Auto-Detection (Fallback to Manual)

```
ws.send(JSON.stringify({
  type: 'settings_update',
  data: {
    auto_detection_enabled: false // Disable auto-detection
  }
}));

// Now you need to send classification_frame manually
ws.send(JSON.stringify({
  type: 'classification_frame',
  data: {
    frame_data: capturedFrame
  }
}));
```

## Advantages

### ✓ User Experience

- **No manual actions needed:** Fully automatic
- **Real-time feedback:** Immediate detection status
- **Fast response:** Triggers as soon as fish is stable in frame

### ✓ Performance

- **Resource efficient:** Quick detection is lightweight
- **Smart throttling:** Only does full recognition when needed
- **Buffer management:** Prevents duplicate recognitions

### ✓ Accuracy

- **3-frame confirmation:** Reduces false positives
- **LLM as primary output:** Indonesian name first
- **Stable detection:** Only triggers when fish is consistently present

## Edge Cases

### Scenario 1: Fish appears and disappears

```
Frame 1: [true] - Fish detected
Frame 2: [true, true] - Still there (2 consecutive)
Frame 3: [true, true, false] - Fish gone (reset count to 0)
Frame 4: [true, false, true] - Fish back (count = 1)
→ No recognition triggered (never reached 3 consecutive)
```

## Scenario 2: Multiple fish in sequence

```
Frame 1-3: [true, true, true] - First fish → Recognition triggered  
Frame 4: Buffer cleared → []  
Frame 5-7: [true, true, true] - Second fish → Recognition triggered  
again  
→ Each fish gets recognized separately
```

## Scenario 3: Processing throttle

```
Frame 1-3: Fish detected → Recognition started (takes 5s)  
Frame 4-10: More fish detected → Skipped (processing in progress)  
Frame 11: Recognition done, can trigger again  
→ Prevents system overload
```

## Testing

See `test_websocket_auto_detection.py` for WebSocket client simulation.

## Troubleshooting

### Detection not triggering?

- Check `detection_status` messages
- Ensure `consecutive_count` reaches `threshold` (3)
- Verify `auto_detection_enabled` is `true`

### Too many recognitions?

- Increase `consecutive_fish_threshold` (e.g., 5 frames)
- Increase `min_processing_interval` (e.g., 1.0 seconds)

### Detection too slow?

- Decrease `consecutive_fish_threshold` (e.g., 2 frames)
- Change `processing_mode` to `"speed"`

## Future Enhancements

- ☐ Configurable threshold via WebSocket settings
- ☐ Detection confidence threshold
- ☐ Multiple fish tracking (separate buffers per bbox)
- ☐ Face detection integration (skip if human detected)
- ☐ Quality-based threshold adjustment

