🔋 uMyo Python Tools ⚡

Professional EMG sensor toolkit for real-time biomedical applications

Transform muscle signals into powerful applications with cutting-edge EMG sensor technology

🚀 Quick Start • 📖 Documentation • ❓ FAQ • 🤝 Contributing

✨ What is uMyo?

uMyo is a revolutionary wireless EMG sensor ecosystem that bridges the gap between human biology and digital technology. Each compact sensor combines:

- 🧠 **EMG Electrodes** → Capture electrical signals from muscle contractions
- 📡 **9-DOF IMU** → Track 3D orientation with precision accelerometer, gyroscope, and magnetometer
- ⚡ **Wireless Communication** → High-speed data transmission to USB base station
- 🔋 **Extended Battery Life** → Optimized power management for hours of operation
- 🌐 **Multi-sensor Networks** → Connect up to 64 sensors for complex gesture recognition

Perfect for researchers, developers, and engineers creating next-generation applications in prosthetics, VR/AR, robotics, and assistive technology.

🌟 Key Features

📡 Real-time Data

- **High-Speed Serial**: 921.6k baud USB interface

🧠 Advanced Processing

- **EMG Analysis**: Real-time muscle activity processing

🎮 Rich Applications

- **Gesture Control**: EMG-based mouse interface

- **Multi-Device**: Up to 64 sensors simultaneously
- **Low Latency**: <5ms end-to-end processing
- **Robust Protocol**: Automatic error correction

- **Frequency Spectrum**: 16-band spectral decomposition
- **3D Orientation**: Professional quaternion mathematics
- **Sensor Fusion**: IMU data integration

- **3D Visualization**: Real-time signal plotting
- **CAD Integration**: Direct FreeCAD control
- **Assistive Tech**: Accessibility applications

🚀 Quick Start

Installation

```
# Create virtual environment (recommended)
python -m venv umyo_env
source umyo_env/bin/activate  # On Windows: umyo_env\Scripts\activate

# Install dependencies
pip install pyserial pygame matplotlib pyautogui numpy bleak
```

Connection Options

🔌 USB Dongle (Traditional)

```
# Connect via USB dongle (921.6k baud)
import serial
from serial.tools import list_ports

port = list(list_ports.comports())[-1].device
ser = serial.Serial(port, 921600, timeout=0)
```

Test Connection

```
# test_umyo.py - Verify your setup in 30 seconds
import umyo_parser
import serial
from serial.tools import list_ports
import time

# Connect to uMyo sensor
ports = list(list_ports.comports())
ser = serial.Serial(ports[-1].device, 921600, timeout=0)

print("🔍 Searching for uMyo sensors...")
for _ in range(100):  # 10 seconds at ~10Hz
    if ser.in_waiting > 0:
        data = ser.read(ser.in_waiting)
        umyo_parser.umyo_parse_preprocessor(data)

        devices = umyo_parser.umyo_get_list()
        if devices:
            device = devices[0]
            print(f"✅ Found sensor {device.unit_id:08X}")
            print(f"🔋 Battery: {device.batt}mV | 📶 RSSI: {device.rssi}")
            print(f"💪 EMG: {device.data_array[-1]} | 🌐 Orientation: {device.Qsg}")
            break
    time.sleep(0.1)
```

Try the Applications

```
# 🖱️ Gesture-based mouse control with calibration (USB)
python umyo_mouse.py

# 📊 Real-time 3-channel EMG visualization
python umyo_3ch_detector.py

# 🔧 Multi-sensor development interface
python serial_test.py
```

📁 Project Structure

```
uMyo_python_tools/
├── 🏠 Core System
│   ├── umyo_class.py          # Device data model & sensor representation
│   ├── umyo_parser.py         # Protocol parsing & device management
│   └── quat_math.py           # 3D orientation mathematics
│
├── 🎮 Applications
│   ├── umyo_mouse.py          # 🖱️ Gesture-based mouse control (USB)
│   ├── umyo_bluetooth_mouse.py # 🖱️ Gesture-based mouse control (Bluetooth)
│   ├── umyo_3ch_detector.py   # 📊 Multi-channel EMG analysis
│   ├── umyo_freecad.py        # 🔧 CAD software integration
│   └── umyo_testing.py        # 🧪 Development & validation
│
├── 📊 Visualization
│   ├── display_stuff.py       # Multi-sensor data visualization
│   ├── display_mouse.py       # Mouse control UI & calibration
│   ├── display_3ch.py         # Three-channel signal display
│   └── parse_plot_data.py     # Matplotlib real-time plotting
│
├── 🔧 Utilities
│   ├── bootloader_usb.py      # Firmware upload & programming
│   └── serial_test.py         # Serial communication testing
│
└── 📚 Documentation
    ├── docs/api-reference.md     # Complete API documentation
    ├── docs/umyo_class.md        # uMyo class specifications
    ├── docs/optimization_analysis.md # Performance optimization
    ├── docs/FAQ.md               # Frequently asked questions
    └── docs/contributing.md      # Development guidelines
```

🎯 Application Showcase

🖱️ Gesture Mouse Control

```
# Transform arm movements into cursor control
python umyo_mouse.py
```

- **Multi-modal input**: EMG + 3D orientation
- **Adaptive calibration**: Personalized thresholds
- **Real-time feedback**: Visual control indicators

📊 Multi-Channel Analysis

```
# Monitor 3 muscle groups simultaneously
python umyo_3ch_detector.py
```

- **Spectral analysis**: Mid-frequency band focus
- **Signal processing**: Exponential smoothing

- **Safety features**: Emergency stop mechanisms

Perfect for: Accessibility applications, hands-free computing, assistive technology

- **Visual feedback**: Real-time bar visualization
- **Research ready**: Clinical data collection

Perfect for: Prosthetic control, rehabilitation, sports science

🔧 CAD Integration

```
# Control 3D models with hand gestures
python umyo_freecad.py
```

- **FIFO communication**: Real-time orientation streaming
- **3D manipulation**: Intuitive gesture-based control
- **Professional workflow**: Direct FreeCAD integration
- **Educational tool**: Tangible 3D learning

Perfect for: Design workflows, education, accessibility

🧪 Development Platform

```
# Advanced multi-sensor visualization
python serial_test.py
```

- **64-sensor support**: Comprehensive device monitoring
- **Performance metrics**: Real-time diagnostics
- **Flexible visualization**: Multiple display modes
- **Debug tools**: Development & troubleshooting

Perfect for: Research, algorithm development, testing

📚 Documentation & Resources

Complete Documentation

- **API Reference** - Complete documentation of all classes, functions, and protocols
- 📚 **uMyo Class Documentation** - Detailed class specifications and usage examples
- ⚡ **Optimization Analysis** - Performance optimization techniques and analysis
- ❓ **FAQ** - Frequently asked questions and troubleshooting guide
- 🤝 **Contributing Guide** - Development guidelines, testing, and community contributions

🎯 Quick Links

- **Hardware Design**: uMyo Hardware Repository
- **FAQ & Troubleshooting**: FAQ
- **Community**: GitHub Discussions
- **Support**: Issue Tracker

⬆ Back to Top • 📖 Documentation • 🚀 Get Started

Empowering the future of human-computer interaction through biomedical sensing

`quat_math.py` - 3D Mathematics Foundation

Purpose: Professional-grade quaternion mathematics library for precise 3D orientation calculations and spatial transformations.

Mathematical Foundations:

- **Quaternion Algebra**: Complete implementation of quaternion arithmetic
- **Rotation Mathematics**: Efficient 3D vector rotation using quaternion operations
- **Numerical Stability**: Normalized operations to prevent accumulation errors
- **Geometric Conversions**: Seamless conversion between rotation representations

Core Data Types:

```
sV = namedtuple("sV", "x y z")        # 3D Vector (x, y, z)
sQ = namedtuple("sQ", "w x y z")      # Quaternion (w + xi + yj + zk)
```

Essential Functions:

Quaternion Operations:

- `q_norm(q)` : Computes quaternion magnitude ($|q| = \sqrt{(w^2 + x^2 + y^2 + z^2)}$)
- `q_renorm(q)` : Normalizes quaternion to unit length for valid rotations
- `q_make_conj(q)` : Computes quaternion conjugate (w, -x, -y, -z)
- `q_mult(q1, q2)` : Quaternion multiplication for rotation composition

Vector Operations:

- `v_norm(v)` : Vector magnitude calculation
- `v_renorm(v)` : Vector normalization to unit length
- `v_mult(v1, v2)` : Cross product for perpendicular vector calculation
- `v_dot(v1, v2)` : Dot product for angle and projection calculations

Spatial Transformations:

- `rotate_v(q, v)` : Rotates 3D vector by quaternion ($v' = q * v * q^*$)
- `q_from_vectors(u, v)` : Computes quaternion rotation between two vectors

Applications in uMyo System:

- **Sensor Fusion**: Combines accelerometer and magnetometer readings

- **Orientation Tracking**: Maintains absolute orientation reference
- **Gesture Recognition**: Calculates relative rotations for gesture detection
- **Calibration**: Establishes reference frames for user-specific setups

🎮 End-User Applications

`umyo_mouse.py` - Advanced Gesture-Based Mouse Control

Purpose: A complete human-computer interface that translates muscle contractions and arm movements into precise computer mouse control.

System Architecture:

- **Multi-modal Input**: Combines EMG signals with 3D orientation for rich interaction
- **Adaptive Calibration**: User-specific threshold learning for personalized control
- **Real-time Processing**: Sub-10ms latency for responsive user experience
- **Safety Features**: Fail-safe mechanisms to prevent unintended actions

Control Modalities:

Movement Control:

- **Orientation Mapping**: Arm rotation directly controls cursor position
- **Sensitivity Scaling**: Adjustable gain for fine vs. coarse control
- **Dead Zone**: Configurable center region to prevent drift
- **Smoothing**: Low-pass filtering for steady cursor movement

Click Detection:

- **EMG Thresholding**: Muscle contraction intensity triggers mouse clicks
- **Temporal Logic**: Prevents accidental double-clicks with timing constraints
- **Multi-muscle Support**: Different muscles can trigger different click types

Scroll Control:

- **Wrist Rotation**: Z-axis rotation maps to scroll wheel movement
- **Proportional Control**: Rotation speed determines scroll rate
- **Bidirectional**: Clockwise/counterclockwise for up/down scrolling

Calibration Workflow:

1. **Center Position**: Establishes neutral orientation reference
2. **X-Axis Mapping**: User moves right to define horizontal control axis
3. **Y-Axis Mapping**: User moves up to define vertical control axis
4. **Z-Axis Mapping**: User rotates wrist to define scroll axis

5. **Muscle Calibration**: Records relaxed and active EMG thresholds
6. **Validation**: Real-time testing of all control modalities

🌟 Acknowledgments

Made with ❤️ by Ultimate Robotics