Interactive Networks w/ OSC

Build creative systems that can do almost anything

Yonatan Rozin

https://yonatanrozin.com

Download these:

- ZIG SIM mobile app (iOS app store & Google Play)
- P5-OSC bridge app: https://github.com/yonatanrozin/OSC-Bridge

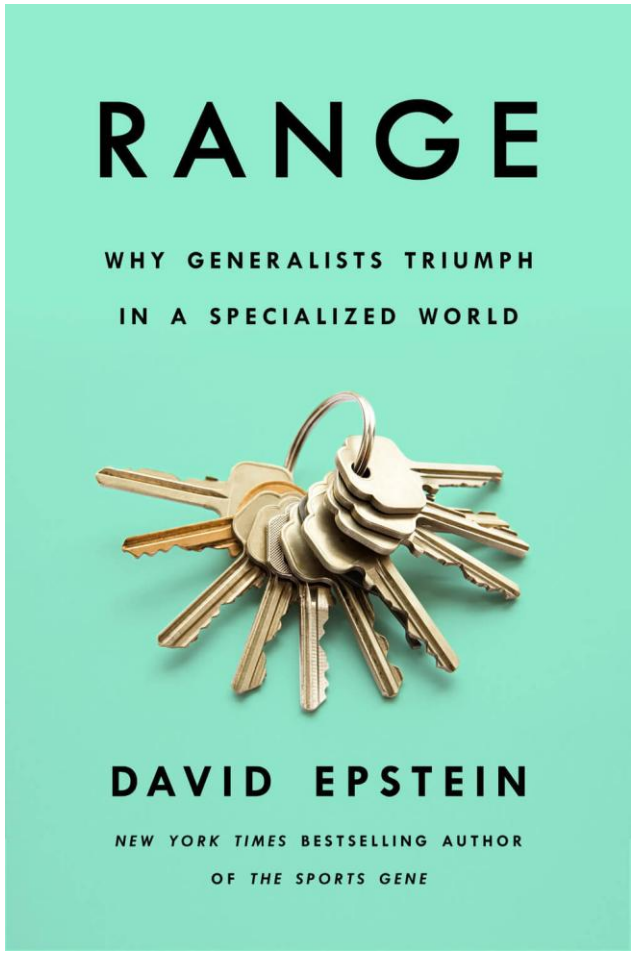
Interactive Networks

Network: any system with 2 or more interconnected parts

Interactive: does cool shit in response to external input

Download these:

- ZIG SIM mobile app (iOS app store & Google Play)
- P5-OSC bridge app: https://github.com/yonatanrozin/OSC-Bridge



Download these:

- ZIG SIM mobile app (iOS app store & Google Play)
- P5-OSC bridge app: https://github.com/yonatanrozin/OSC-Bridge

Work samples

Breathing Alarms (2026) - Ivana Dama

Download these:

- ZIG SIM mobile app (iOS app store & Google Play)
- P5-OSC bridge app: https://github.com/yonatanrozin/OSC-Bridge

Work samples

Kinetic Sculpture & Controller (2025-, in-progress) –
Shakti Shalligram & Yonatan Rozin

Download these:

- ZIG SIM mobile app (iOS app store & Google Play)
- P5-OSC bridge app: https://github.com/yonatanrozin/OSC-Bridge

Work samples

BodyMouth (2022-) - Kat Mustatea

Download these:

- ZIG SIM mobile app (iOS app store & Google Play)
- P5-OSC bridge app: https://github.com/yonatanrozin/OSC-Bridge

Benefits of modular networks

- Not limited to the functionality of a single program/device
  - Bridge together multiple software components, leverage the best features of each, no compromise needed
  - Overcome CPU constraints with multiple devices
- Combine digital and physical
  - Create a physical (handheld, wearable, tabletop, etc.) interface for real-time software
  - Enable flexible audience participation
- Overcome physical constraints
  - Replace USB cables with wireless messaging
  - Distribute multiple devices across a space (or the globe)

OSC – Open Sound Control

(widely supported across ALL mediums, not just sound!)

- Simple messaging syntax supported by a wide range of software programs:
  - TouchDesigner, Unity, Unreal Engine, MadMapper, Processing, P5, OpenFrameworks
  - Max/MSP, Pure Data, Ableton (w/ Max4Live), Supercollider
  - Arduino, ESP32, Raspberry Pi, other (WiFi-enabled) microcontrollers
  - Libraries for Python, JavaScript (node.js), C++, most other programming languages
  - Resolume, QLab, Isadora
  - Most smartphones (via TouchOSC, Zig Sim or other app)
  - Lots of others!

- Flexible structure: send (basically) any quantity of custom data

OSC – Open Sound Control

(widely supported across ALL mediums, not just sound!)

- OSC message syntax: address + arguments
  - i.e. `/gyroscope -75.23 50 “hello”`

OSC – Open Sound Control

(widely supported across ALL mediums, not just sound!)

- OSC message syntax: address + arguments
  - i.e. **`/gyroscope`** `-75.23 50 “hello”`

- `/gyroscope` – OSC address
  - Message “label” – identifies the importance of the message/data
  - Must begin with a “/” and consist of letters, numbers or underscores
  - Can have multiple levels, i.e. `/sensor1/gyroscope/x`

OSC – Open Sound Control

(widely supported across ALL mediums, not just sound!)

- OSC message syntax: address + arguments
  - i.e. `/gyroscope` **`-75.23 50 “hello”`**

- `-75.23 50 “hello”` – data arguments
  - (optional) one or more arguments to send in the message
  - Main datatypes: strings, integers, floats, Booleans, blobs (raw binary)

Message transport protocols – TCP + UDP

Transport protocols – concerned with **transmission** of data, regardless of the data itself

- TCP – Transmission Control Protocol – **reliable but slow**
  - Guarantees delivery by re-requesting lost packets (chunks of data)
  - Guarantees data accuracy with checksums and confirmations
  - Introduces latency + jitter (inconsistent timing, sensitive to network speed)

- UDP – User Datagram Protocol – **fast fast fast** – OSC convention
  - Sends the data, moves on
  - No guarantee of receipt or accurate information (data loss is VERY uncommon)
  - Low-latency – perfect for real-time interactions + high-volume streams

Transmitting data over TCP/UDP

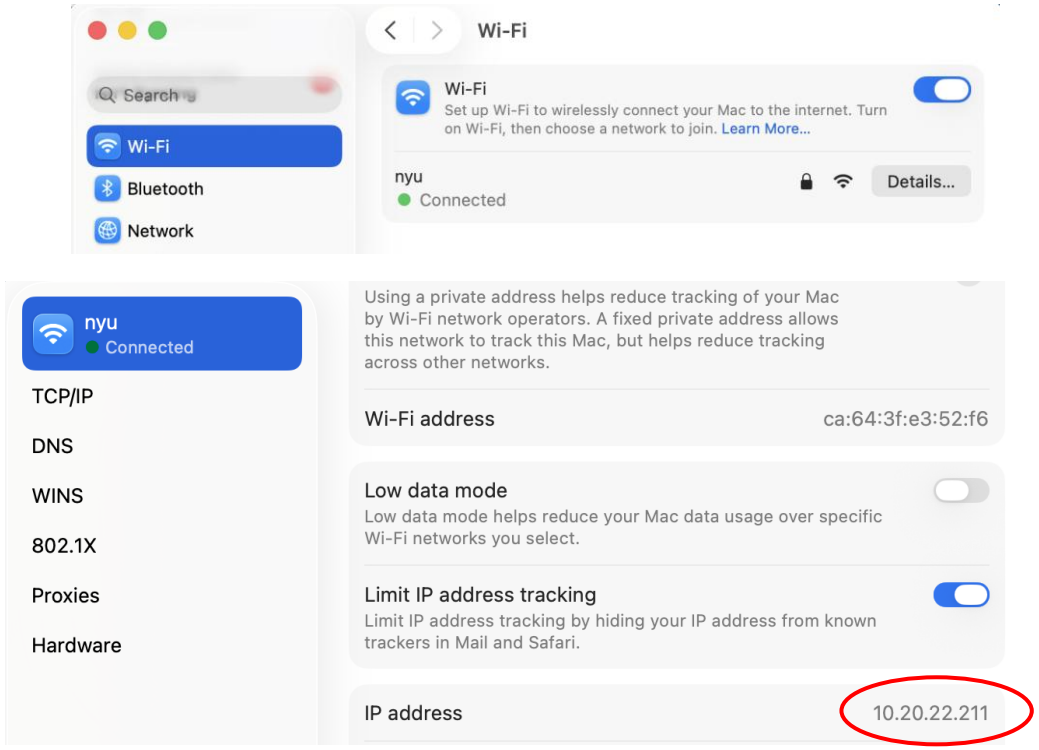
- IP Address (i.e. `192.168.1.53`)
  - Unique address used to send wireless messages between devices
  - Special loopback address: `127.0.0.1` or `localhost`
    - Used for sending data between multiple programs on a single device
    - Message is routed internally – bypasses the network entirely
- Port #
  - Included in outgoing messages, used to differentiate between multiple incoming streams of information on a single device
  - Receiver listens for incoming information with a specific port
  - Gates at an airport, apartments in a building, etc.

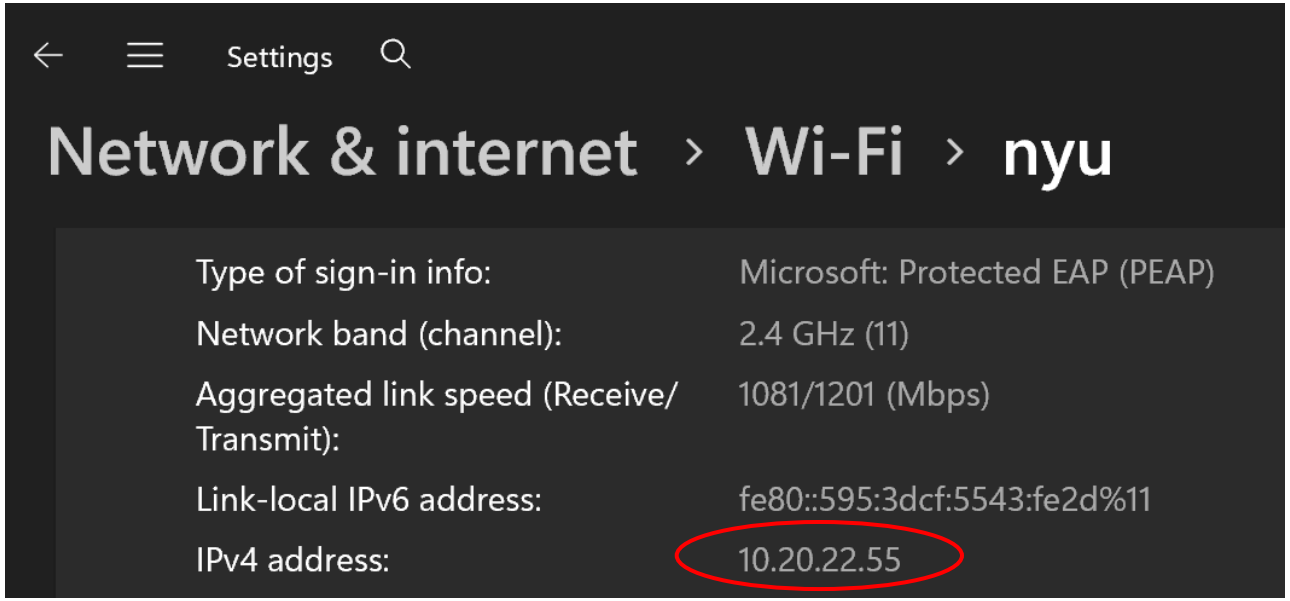
Sending OSC messages (from anywhere)

- Sender and receiver device must both be connected to the same network – wireless (WiFi) or wired (ethernet)
  - (Unless device is sending to itself using loopback address `127.0.0.1`)
- Receiver listens for incoming data with a specific port #
- Sender device sends messages to the receiver's IP address with the specific port #
- Receiver processes the message data (or not) according to the OSC address

Get your IP address! (IPv4)

- Remember, your IP address will change whenever you connect to a WiFi network – check your IP address frequently!





Smartphone as custom remote w/ OSC

- Reduce e-waste! Don’t trade-in for $5, repurpose an old (or new) smartphone into a feature-packed remote control instead
- No SIM card or active phone plan required – use virtually any smartphone from the past decade
- No custom coding, wiring or additional hardware/modules required

Smartphone Hardware (vs. individual module value)

- GPS ($20+) – create site-specific projects
- Accelerometer / Gyroscope / Compass ($20) – gestural controls!
- Multi-touch touchscreen ($40 for 7” module wtf)
- WiFi module ($25+ for a WiFi-enabled Arduino)
- HD Camera ($10-15)
- Microphone ($5-10)
- Long-lasting rechargeable battery ($10)
- Compact form-factor (priceless)
  - No prototyping, soldering, Arduino bricking or crying required

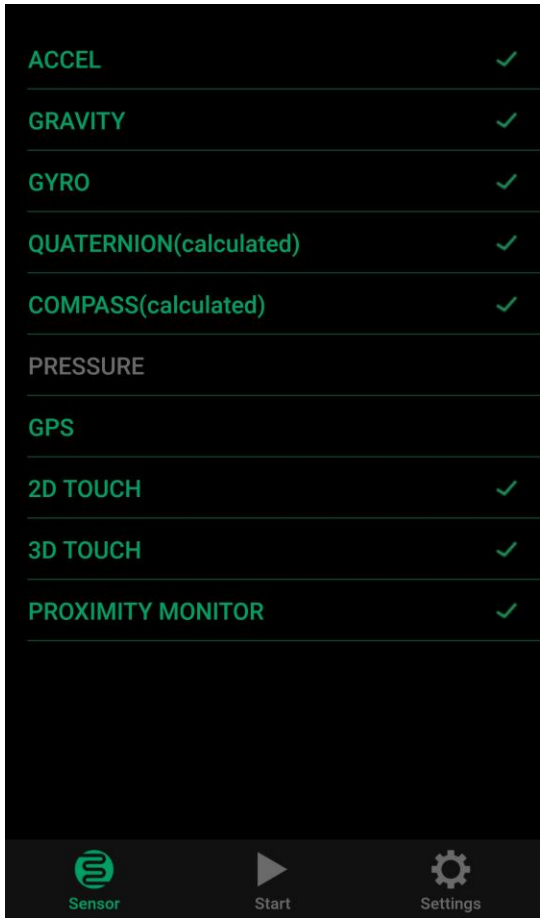
Sending OSC from smartphone

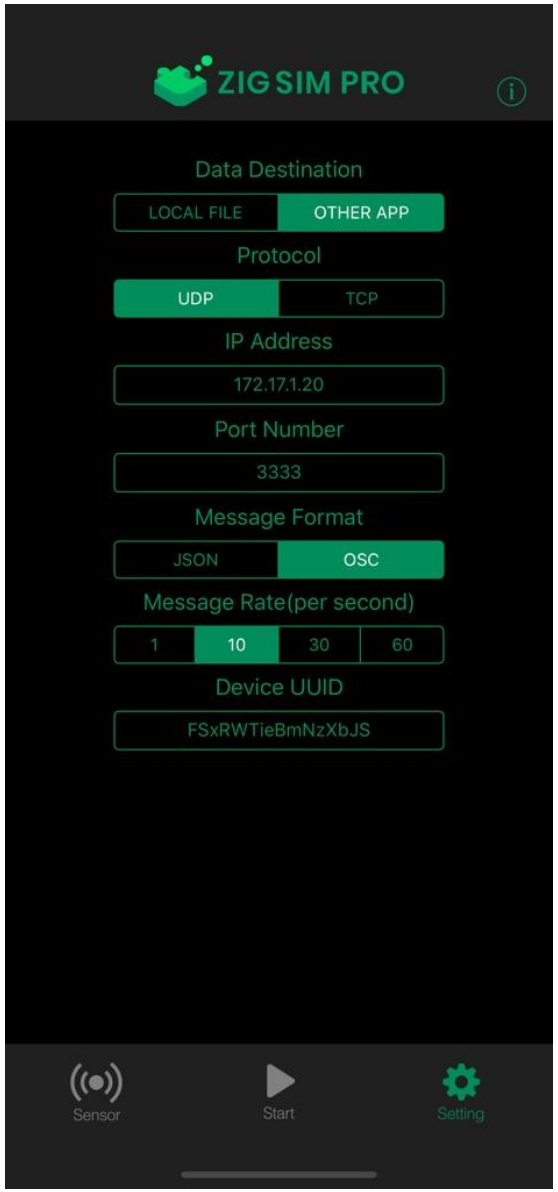
- TouchOSC
  - Highly customizable w/ knobs, buttons, sliders, touchpads, etc.
  - Free desktop app, EXPENSIVE ($15) mobile app

- Zig Sim – recommended for prototyping
  - Less customizable, supports 2D touch location, accelerometer, gyroscope, compass, GPS, mic volume
  - **Free** on all mobile devices (iOS and Android) – PC not supported
  - Pro version ($4, absolutely worth) adds NFC reader, face + body tracking (w/ ARKit), object detection, raw camera + microphone streams over NDI

Sending OSC w/ Zig Sim

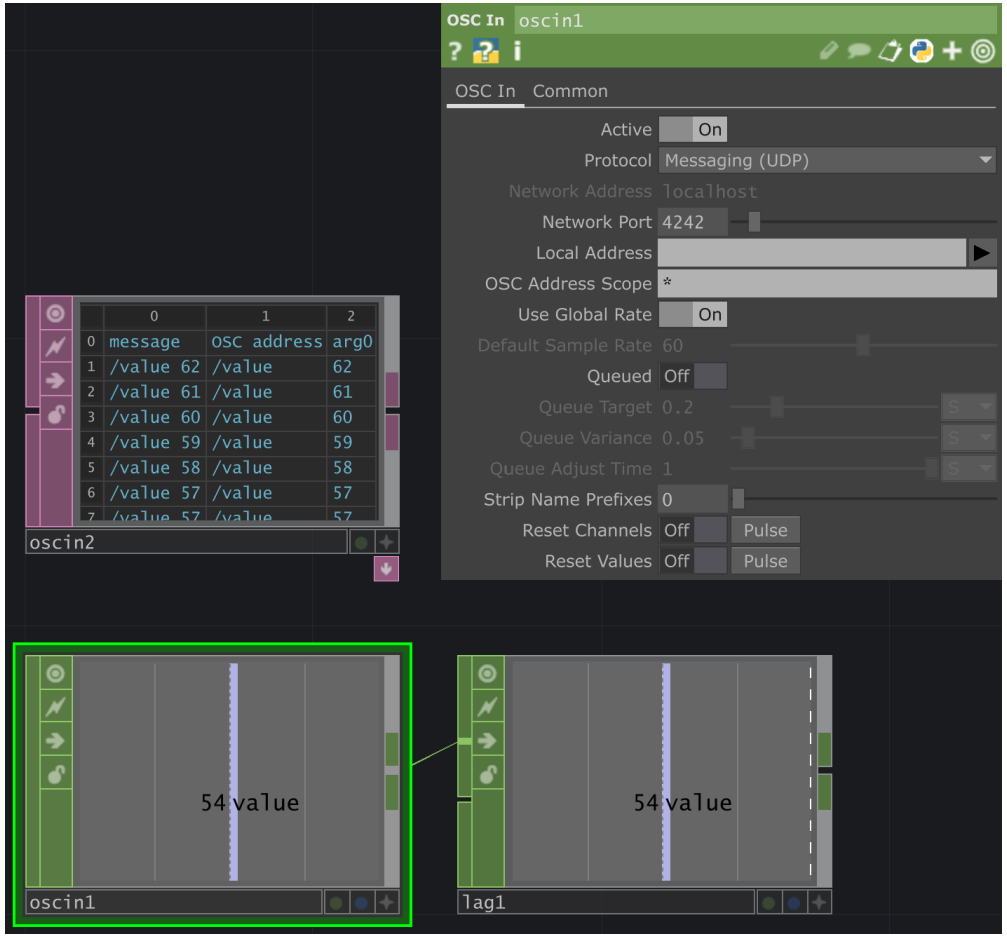
- Sensor tab
  - Choose desired sensor(s) to use
- Settings tab
  - Choose protocol (likely UDP)
  - Enter receiver IP address + port
  - Choose OSC format + FPS
- Enter start tab to begin stream!
- (recommended) turn off low-battery mode & disable phone sleep mode





Receiving OSC in TouchDesigner

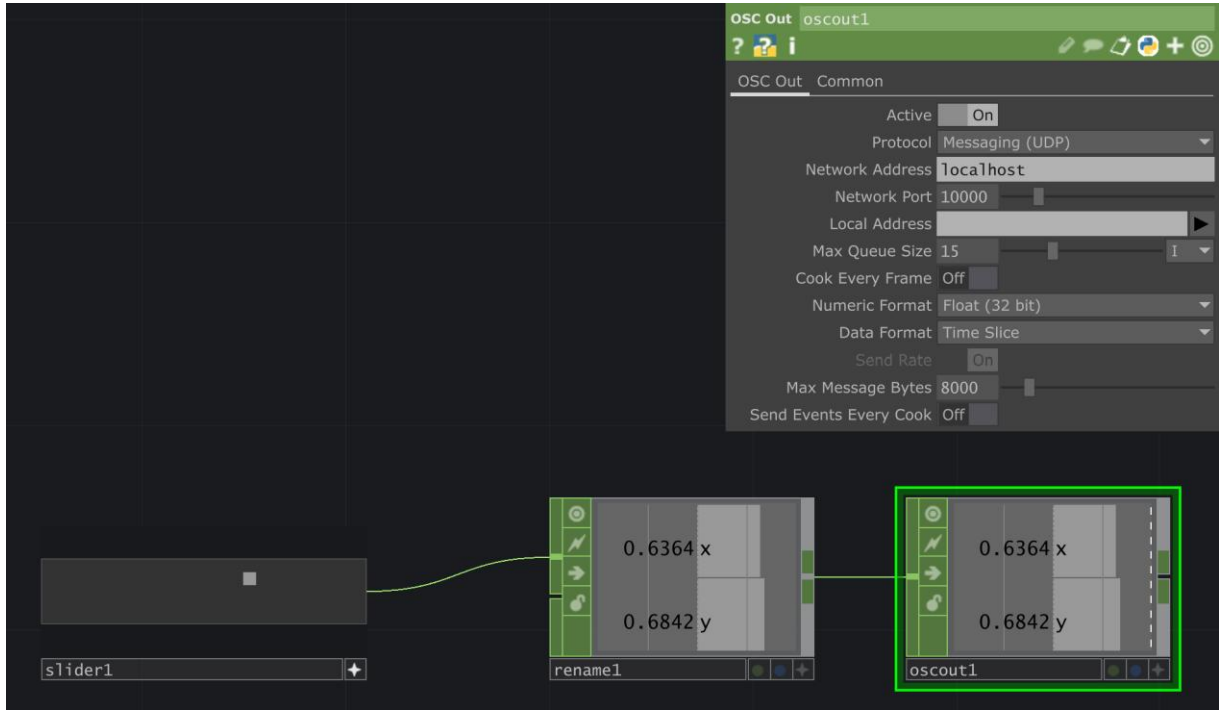
- OSC In CHOP (receives ints/floats) or OSC In DAT (receives strings)
  - Specify port # to receive messages on
- Optional – add specific addresses to receive in OSC address scope
  - Use wildcard * in OSC address scope to match ANY string in that address level (*/temp will match abc/temp, efg/temp, etc.)
- Recommended for OSC In CHOP: add Lag CHOP to smooth incoming data
- Recommended for OSC In DAT: enable “split message into columns” & “split bundle into messages”



Sending OSC from TouchDesigner

- Assemble outgoing channels (from panel components, etc.)

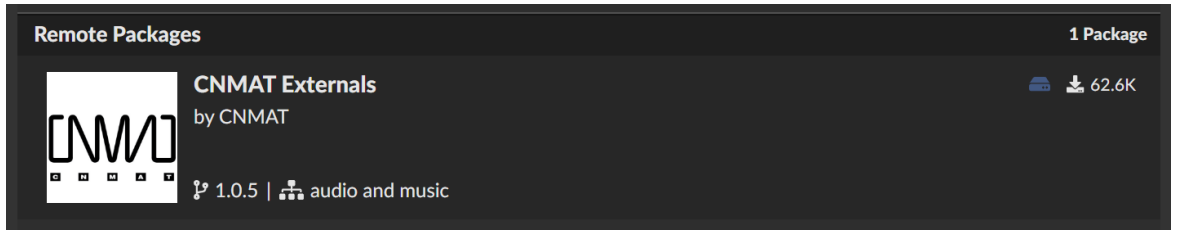
- Add Rename CHOP
  - rename channels to corresponding OSC addresses (leading slashes NOT needed)



- OSC Out CHOP – sends channel values as OSC messages
  - Specify protocol, destination IP address and port in OP parameters
    - Be sure receiver device is expecting data on the port!
  - Recommended – disable “cook every frame”

Receiving OSC in Max/MSP

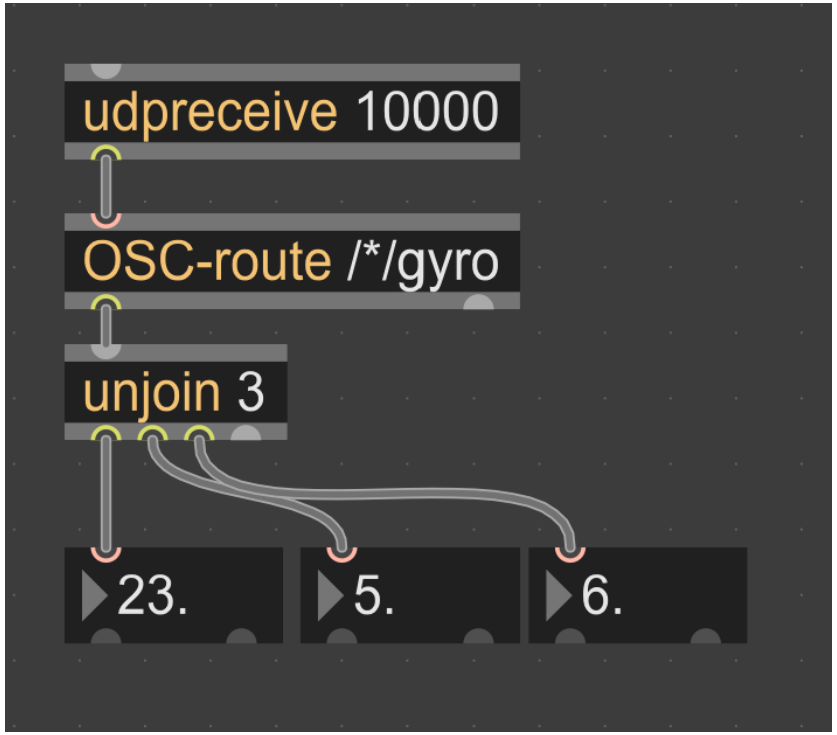
- Install OSC package
  - File > Package Manager
  - Search + install **CNMAT Externals**



- Use `[udpreceive <port>]` to receive messages on specified port

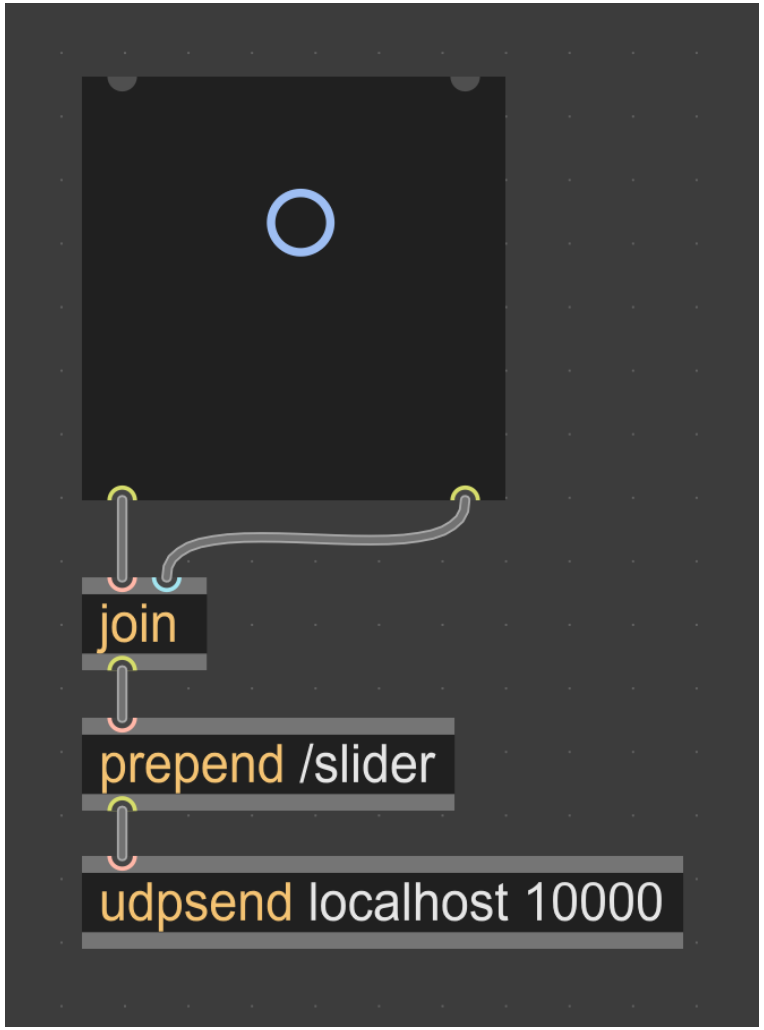
- Use `[OSC-route <route>]` to route messages by address
  - `*` wildcard: will match ANY address value in that level (`/sensor1/gyro`, `/sensor2/gyro`, …)

- Use `[unjoin <#-of-args>]` to split multi-argument messages



Sending OSC from Max/MSP

- Use `[udpsend <IP_ADDR> <port>]`
  - Specify receiver IP address + port
- Send messages to udpsend: `(<addr> <args>)`
  - (optional) use `[join]` to combine multiple arguments into a single message
  - Use `[prepend <addr>]` to prepend an OSC address to an existing Max message (number, string, list, etc.)



OSC in P5 sketch or other JS web-app

(software undergoing development + testing: functionality **NOT** guaranteed)

Introducing **OSC Bridge** – a desktop app that turns your P5 sketch or other static webpage into a standard desktop application & includes functions for sending/receiving OSC directly in your sketch

- Download + install from https://github.com/yonatanrozin/OSC-Bridge (`.dmg` file for Mac, `.exe` file for Windows)
  - App may raise security warnings on Mac, bypass these in "Privacy & Security" settings (scroll down to find message – click "open anyway")

- Open application, enter ctrl-E to open sketch files folder – place your P5 sketch files in here!

OSC in P5 sketch or other JS web-app

(software undergoing development + testing: functionality **NOT** guaranteed)

To handle messages: `OSC.route("<addr>", <callback>)`

- `<addr>` – OSC address, accepts * wildcards
- `<callback>` - a function that will do something with the message arguments
  - If callback involves P5 functions, place the route inside `setup()`

```
OSC.route("/touch0", (vals) => {
  circle(
    map(vals[0], -1, 1, 0, width),
    map(vals[1], -1, 1, 0, height),
    radius
  );
});
OSC.route("/touchradius0", (val) => {radius = val;});
OSC.route("/accel", (vals) => {accel = vals;});
```

OSC in P5 sketch or other JS web-app

(software undergoing development + testing: functionality **NOT** guaranteed)

To send messages:

`OSC.send("<address>", <args>, <IP_addr>, <port>)`

- `<address>` - OSC address (leading slash required)
- `<args>` - either a single argument (number or string) or an array of arguments
- `<IP_addr>, <port>` (optional) - destination IP address + port
  - Leave out to use default: localhost port 4243

Receive OSC in Arduino

See complete example OSC receiver at https://github.com/yonatanrozin/OSC-Bridge/blob/main/receivers

- Install OSC library by Adrian Freed, `#include <OSCMessage.h>`
- Enter WiFi network name + password, messaging port + OSC address
- Connect to WiFi in `setup()` (recommended, print IP address)
- Use `getFloat(i)`, `getInt(i)`, `getString(i)`, `getBoolean(i)` to extract message arguments in `loop()` (`i`: argument #, counting from 0)

```
22  const char WIFI_SSID[] = ""; //WiFi network name
23  const char WIFI_PASS[] = ""; //WiFi network password
24
25  const int port = 4243; //choose a port # (2000+ recommended)
26
27  const char OSC_address[] = "/slider"; //choose an OSC address
```

```
45    while (WiFi.status() != WL_CONNECTED) {
46      WiFi.begin(WIFI_SSID, WIFI_PASS);
47      Serial.print(".");
48      delay(500);
49    }
50    Serial.print("\nIP: ");
51    Serial.println(WiFi.localIP());
```

```
68          if (msg.fullMatch(OSC_address)) {
69            val = msg.getFloat(0) * 180;
70          }
```

Send OSC from Arduino

- Install & include OSC library, connect to WiFi network (see previous slide example)
- Create destination IP address + port variables
- Create new message w/ OSC address, add argument(s), send!

```
IPAddress outIp(128, 32, 122, 125);
const unsigned int outPort = 9999;
```

```
OSCMessage msg("/analog/0");
msg.add((int32_t)analogRead(0));
Udp.beginPacket(outIp, outPort);
msg.send(Udp);
Udp.endPacket();
msg.empty();
```

Highly recommended - use your own router!

- Public networks are unreliable & annoying
  - Require re-programming network name + password in each new location
  - Device IP address may change with each connection
  - Possible inconsistent speed + bandwidth
  - Possible cybersecurity limitations – may block wireless messaging entirely

- Portable travel router ($35) - 1 minute setup, no modem needed
  - Flip mode switch to “AP/rng Ext/Client”
  - Power with USB cable – supports wall power or computer USB port!
  - Connect devices using the same SSID (network name), password and IP address every time

Thank you!!

https://yonatanrozin.com/

https://www.instagram.com/yonatanrozin