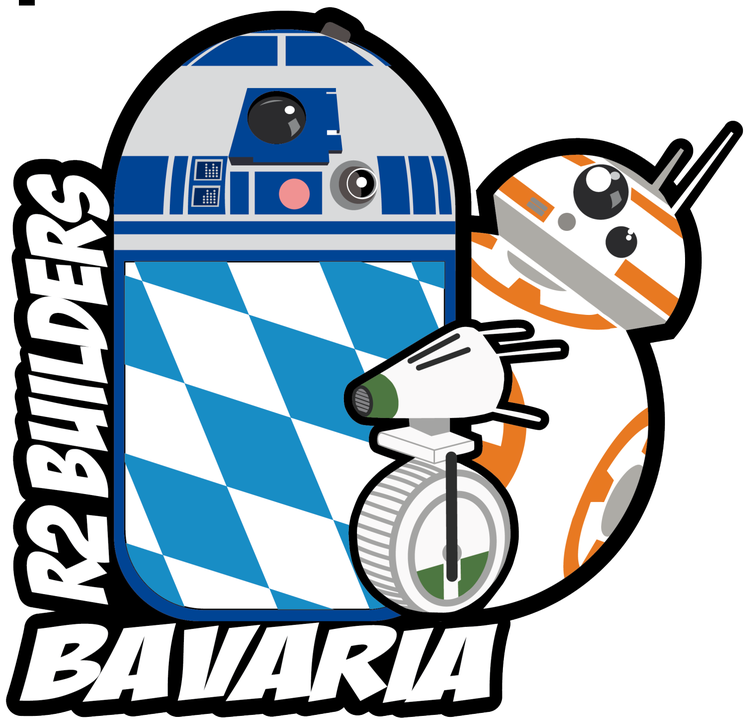
The Monaco Control System

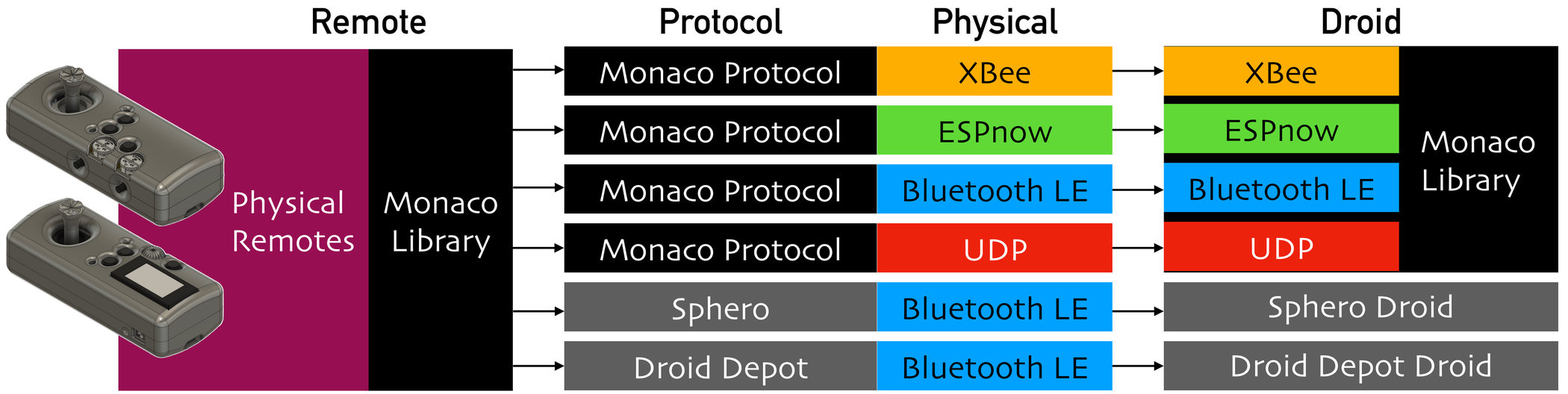
All your droids with one remote



Björn Giesler, December 2025

The Monaco Control System

All Your Droids With One Remote

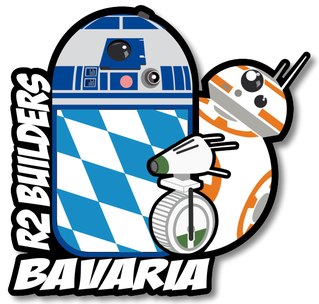


…more protocols can and will be added.

The Monaco Control System

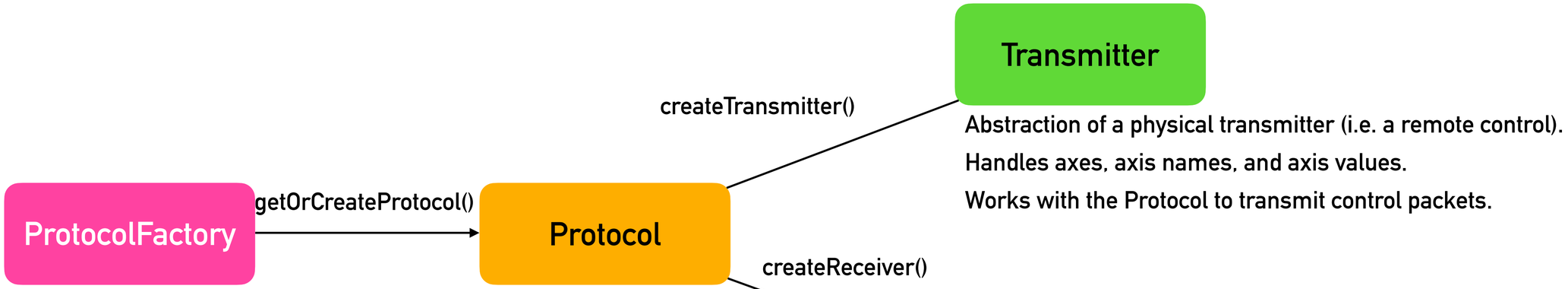
All Your Droids With One Remote

- Monaco is…
  - a software abstraction API to help add safe, fully-featured RC to your droid without a lot of work.
  - a protocol definition for real-time RC control, telemetry, configuration and network management.
  - an open-source library that implements API and protocol, over a number of different hardware mechanisms.
  - a reference implementation in the form of dual concealable fully-featured remotes, the software to drive them, and the software to drive a number of droids (D-0, MSE-6, R2-D2, BB-8).
- Monaco is not…
  - a commercial product (although the physical remotes and some droid hardware are).



Core Concepts

Protocols, Receivers, Transmitters



Can be used to create Protocol objects.

Handles load/store of protocol configs.

Understands the on-the-wire protocol.

Subclasses for specific protocols.

Handles

- pairing and network management
- packet distribution
- node configuration
- safety features.

Abstraction of a physical transmitter (i.e. a remote control).

Handles axes, axis names, and axis values.

Works with the Protocol to transmit control packets.

Abstraction of a physical receiver (i.e. a droid).

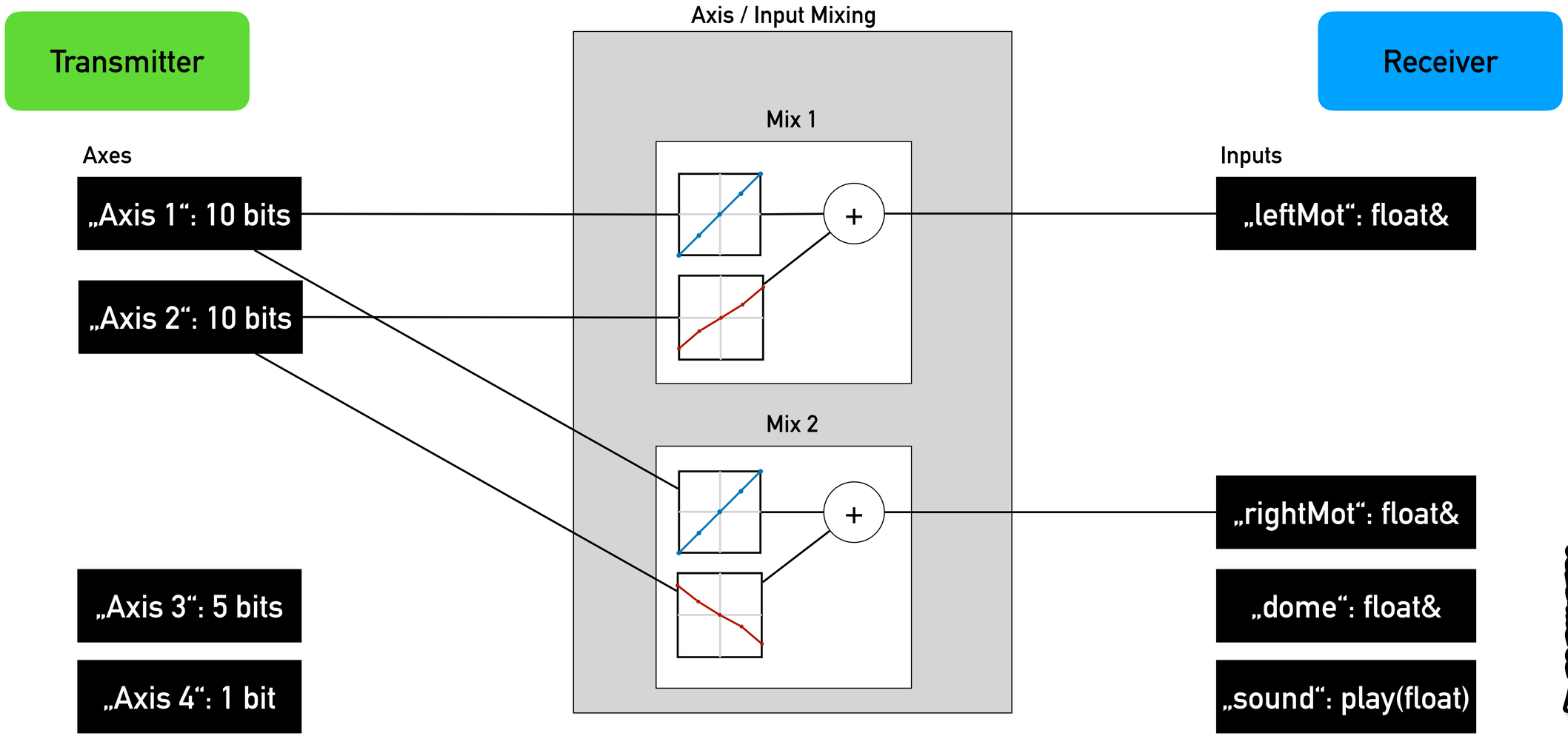
Handles inputs, input names, and axis/input mappings.

Works with the Protocol to receive control packets, compute input values, and let the droid know that inputs have changed.

Core Concepts

Axes and Inputs

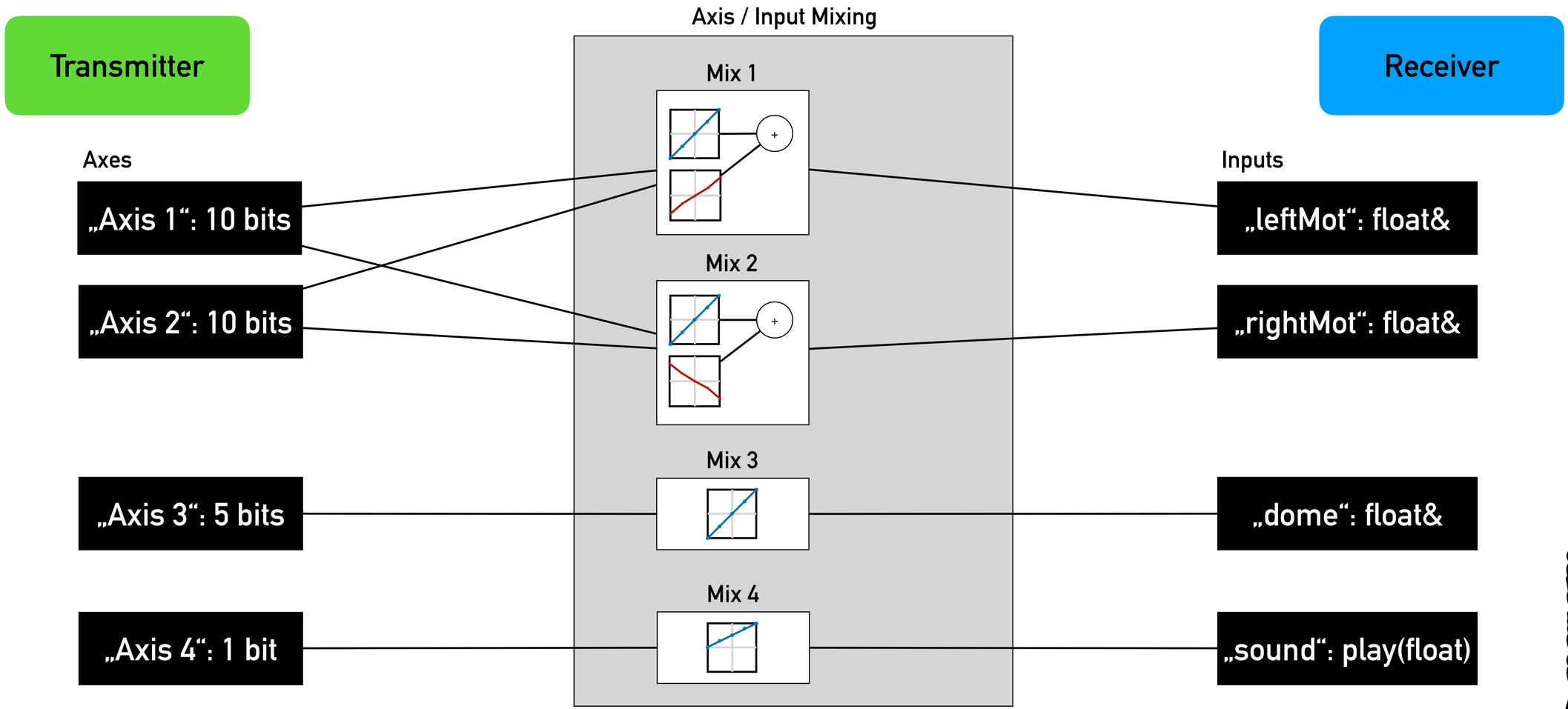
The library supports mixing of up to two axes into each input, with 5-point mixing curves for each axis.



Core Concepts

Axes and Inputs

On receiver side, inputs can be float references that are silently updated, or callback methods to trigger actions.

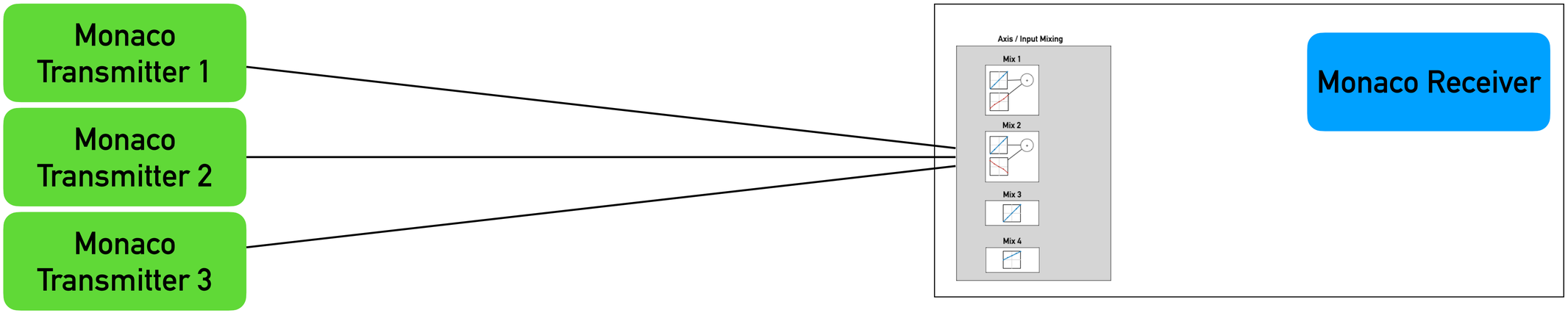


Core Concepts

Axes and Inputs

Mixing is handled on receiver side to allow for multiple transmitters, unless we can't do that because the receiver is proprietary; then we do it on transmitter side.

Receiver-Side Axis / Input Mixing



Transmitter-Side Axis / Input Mixing

Code Sample: Receiver Side (Droid!)

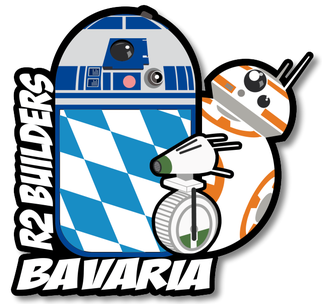
```
Protocol *protocol;
Receiver *rx;
float remoteSpeed, remoteTurn;

void setup(void) {
    // create protocol and receiver
    protocol = ProtocolFactory::getOrCreateProtocol(MONACO_ESP);
    rx = protocol->createReceiver();

    // define inputs
    InputID speedInput = rx->addInput(„speed“, remoteSpeed);
    InputID turnInput = rx->addInput(„turn“, remoteTurn);

    // optional - add default mappings for axes 0 and 1
    rx->setMix(speedInput, AxisMix(0, INTERP_LIN_CENTERED, 1, INTERP_LIN_CENTERED));
    rx->setMix(speedInput, AxisMix(0, INTERP_LIN_CENTERED, 1, INTERP_LIN_CENTERED_INV));
}

void loop(void) {
    proto->step();
    // do something with remoteSpeed and remoteTurn
    delay(10);
}
```



Code Sample: Transmitter Side (Remote!)

```
Protocol *protocol;
Transmitter *tx;
const uint8_t JOY_V_PIN = 5, JOY_H_PIN = 6;

void setup(void) {
    // create protocol and transmitter
    proto = ProtocolFactory::getOrCreateProtocol(MONACO_ESP);
    tx = proto->createTransmitter();
}

void loop(void) {
    tx->setAxis(0, analogRead(JOY_V_PIN)/float(1<<analogReadResolution()), UNIT_UNITY);
    tx->setAxis(1, analogRead(JOY_H_PIN)/float(1<<analogReadResolution()), UNIT_UNITY);
    proto->step();
    delay(10);
}
```