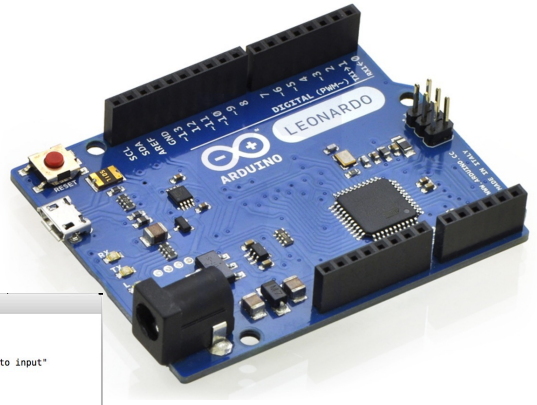
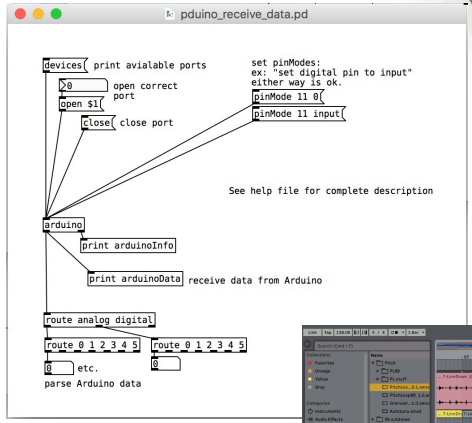
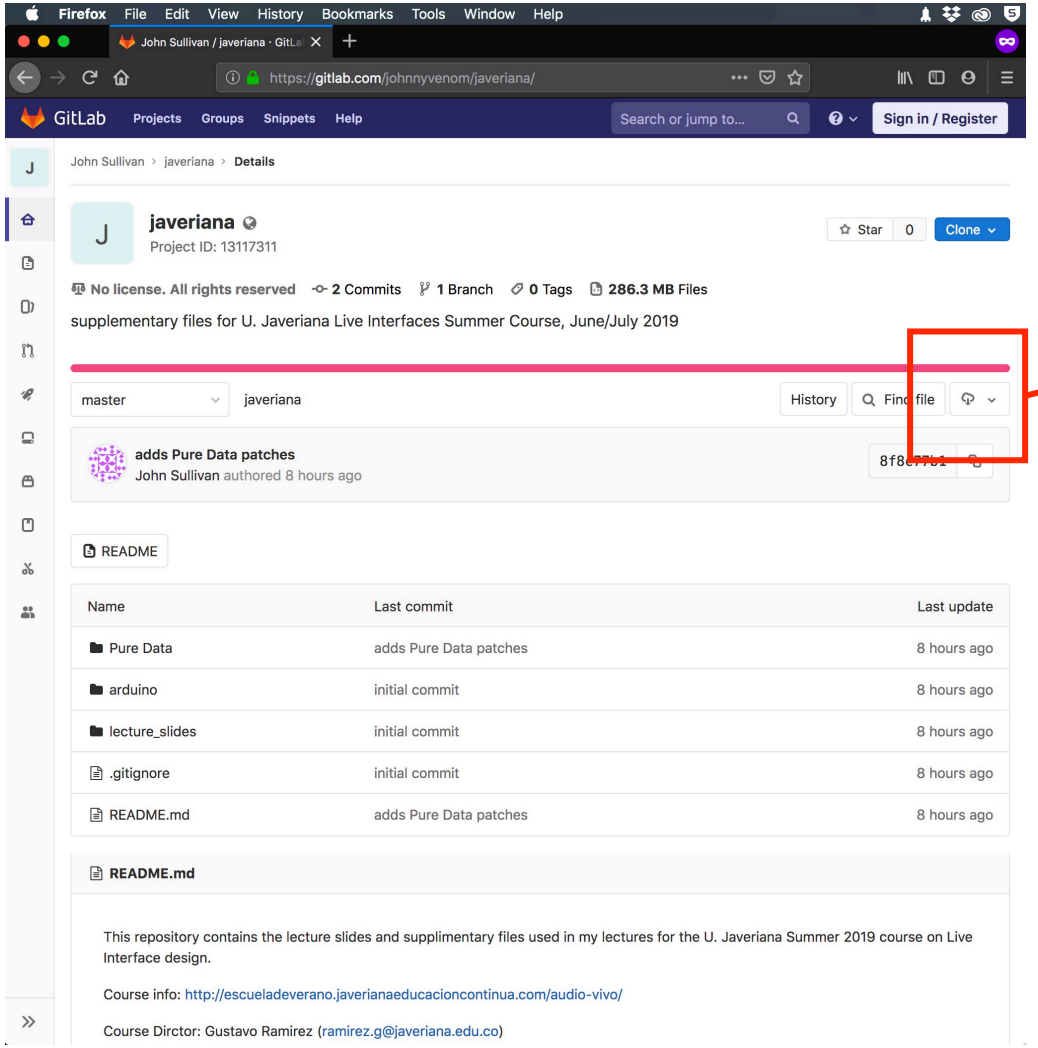
Gesture acquisition & feedback

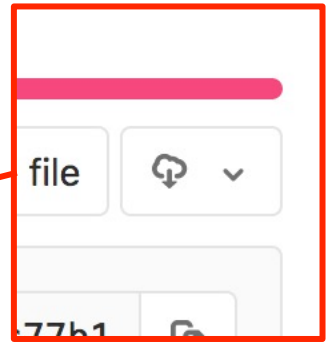


Sending, receiving and parsing serial and MIDI data
- Arduino <-> Pure Data
- Pure Data <-> other software
- Arduino <-> other software



John Sullivan
Centre for Interdisciplinary Research in Music Media and Technology
Input Devices and Music Interaction Laboratory
john.sullivan2@mail.mcgill.ca

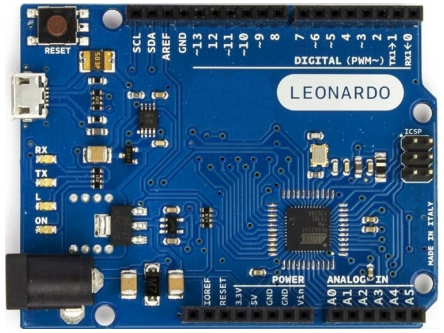


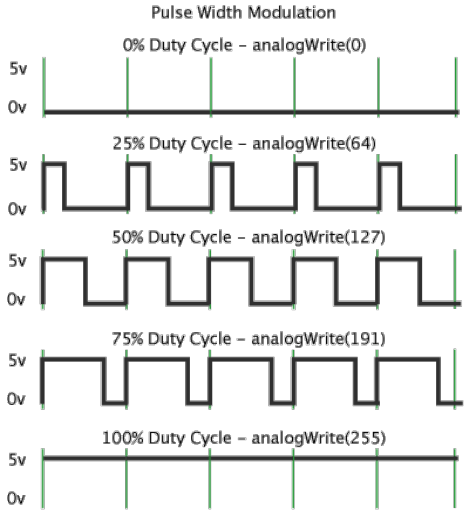


download here

Note: Materials from my lectures (slides, Pd patches, etc.) are online and can be downloaded from:

1. Gesture acquisition with Arduino & Pd





- A microcontroller (Arduino) has 2 types of pins for receiving (and sending):
  - Analog - can read continuous voltages between 0 and 5 volts
    - Arduino analog range is 0 – 1023 (10 bit sampling ADCs)
    - Pins A0 – A5
    - Sensors: potentiometer (knob), temperature sensor, fader, light sensor, etc.
  - Digital - can read or write values 1 or 0 (HIGH or LOW)
    - Sensors: buttons & switches
    - Additionally digital pins 3, 5, 6, 9, 10, 11, 13 (on Leonardo) are equipped with Pulse Width Modulation, allowing them to output analog values from 0 – 255. (Ex. dim an LED)
- Arduinos send and receive data from other devices via **serial** communication (UART – universal asynchronous transmitter-receiver) connected to another device via USB.

Firmata and Pduino

- The next several slides cover the Pduino library for Pure Data. Regrettably, due to an undocumented compatibility issue:
- **Pduino is not compatible with Arduino Leonardo and Windows.**
  - **It does work running Leonardo with MacOS, however it can be buggy.**
- Alternatives:
  - A) Use a different Arduino (Uno, for instance)
  - B) Don't use Pduino and use OSC for communicating between Arduino and Pure Data (See Lecture 3).

Firmata and Pduino

- The fastest and easiest way to get sensor data into Pure Data.
- Firmata: A protocol for communicating with Arduino from a computer via serial port.
  - A standard Firmata sketch is loaded onto the Arduino, and all commands to control it are send directly from software (Pure Data)
  - NO ARDUINO PROGRAMMING NEEDED
- Pduino: An extension for Pure Data to communicate directly with the Arduino
  - The main object is [arduino], which sends and receives data directly from the Arduino.

PROs:

- Easiest way to communicate with Arduino
- No C++ (Arduino) programming knowledge required

CONs:

- Can't handle standard messaging protocols like MIDI or OSC
- Not possible to do any additional computation on the microcontroller (ie., conditional logic, filtering and scaling data)
- Can run slow (on Arduino and Pd)

Installing:

- In PD, install the 'pduino' and 'comport' externals ('Help' menu > Find Externals)
- Add to the PD file search path
- In Arduino IDE, for to File > Examples > Firmata; open and upload the StandardFirmata sketch.

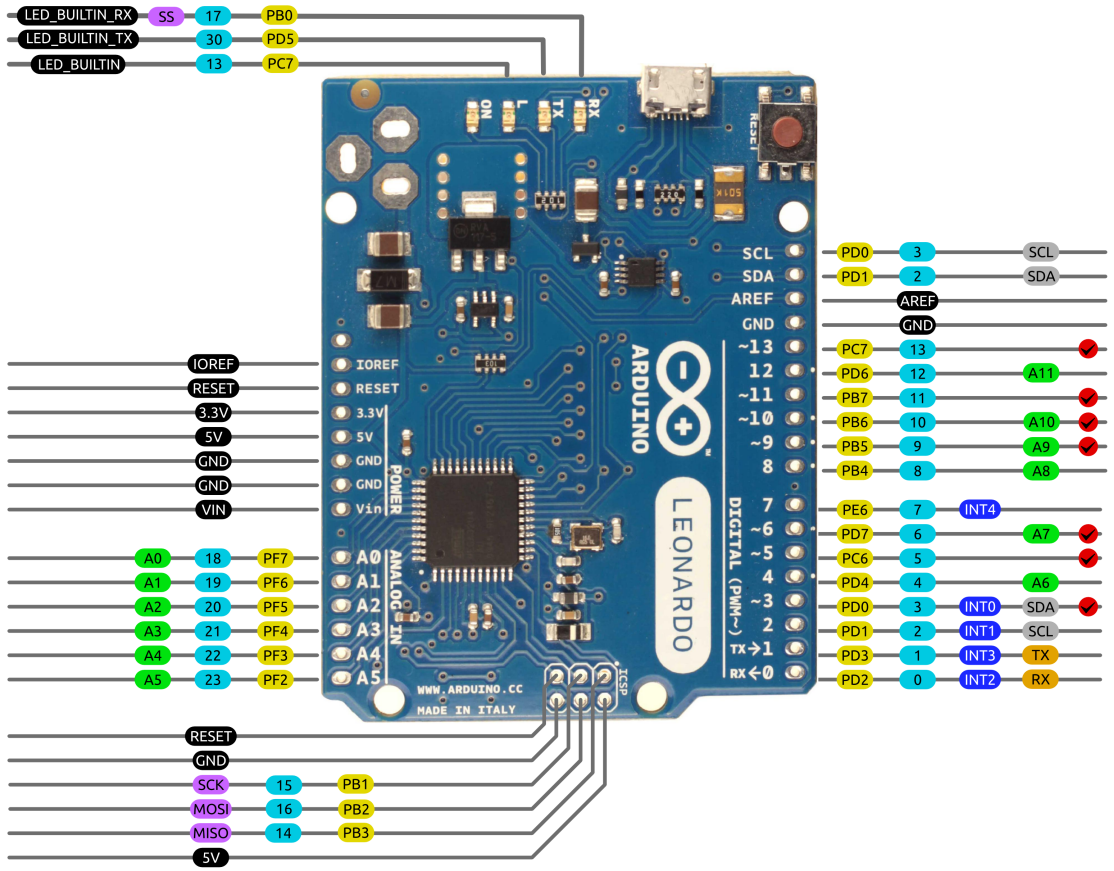
pinModes for Arduino Leonardo:

- Digital pins are 0 – 13
- Analog pins are A0 – A5 (also function as digital pins 18 – 22)
  - also A6 – A11 on digital pins 4, 6, 8, 9, 10, 11 12
- Available pinModes are:
  - 0 INPUT (digital)
  - 1 OUTPUT (digital)
  - 2 ANALOG (analog, but assign to their digital pin #s)
  - 3 PWM (digital, 3, 5, 6, 9, 10, 11, 13)
    - look for ~ sign on board
  - 4 SERVO (digital, 2 – 13)

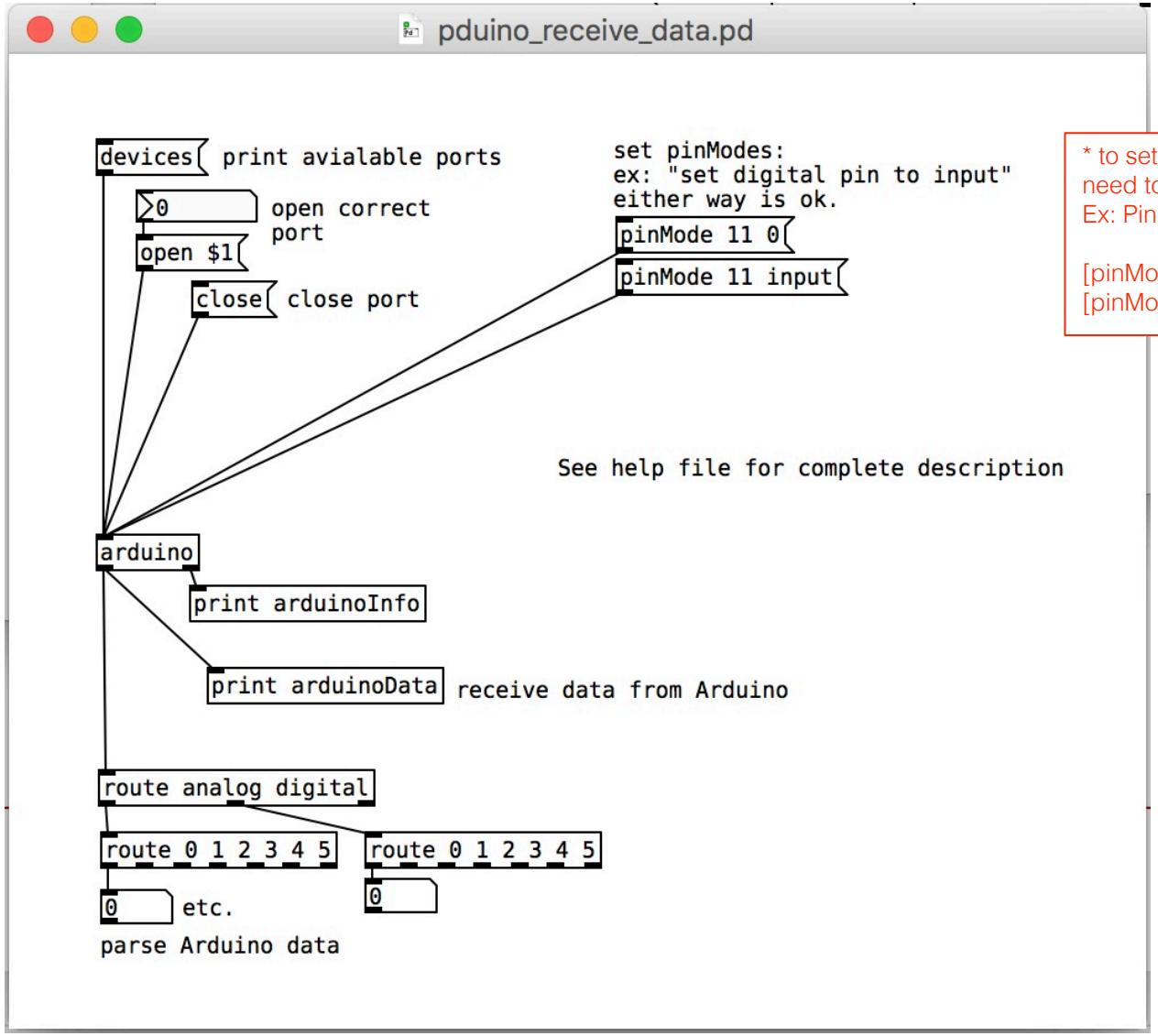
```
LIST OF AVAILABLE PIN MODES

pinMode 11 input    ==    pinMode 11 0      digital input
pinMode 11 output   ==    pinMode 11 1      digital output
pinMode 16 analog   ==    pinMode 16 2      analog input
pinMode 11 pwm      ==    pinMode 11 3      pulse width modulation (output)
pinMode 11 servo    ==    pinMode 11 4      servo (output)
```

Arduino Leonardo Pinout



Pduino: Receive data from Arduino

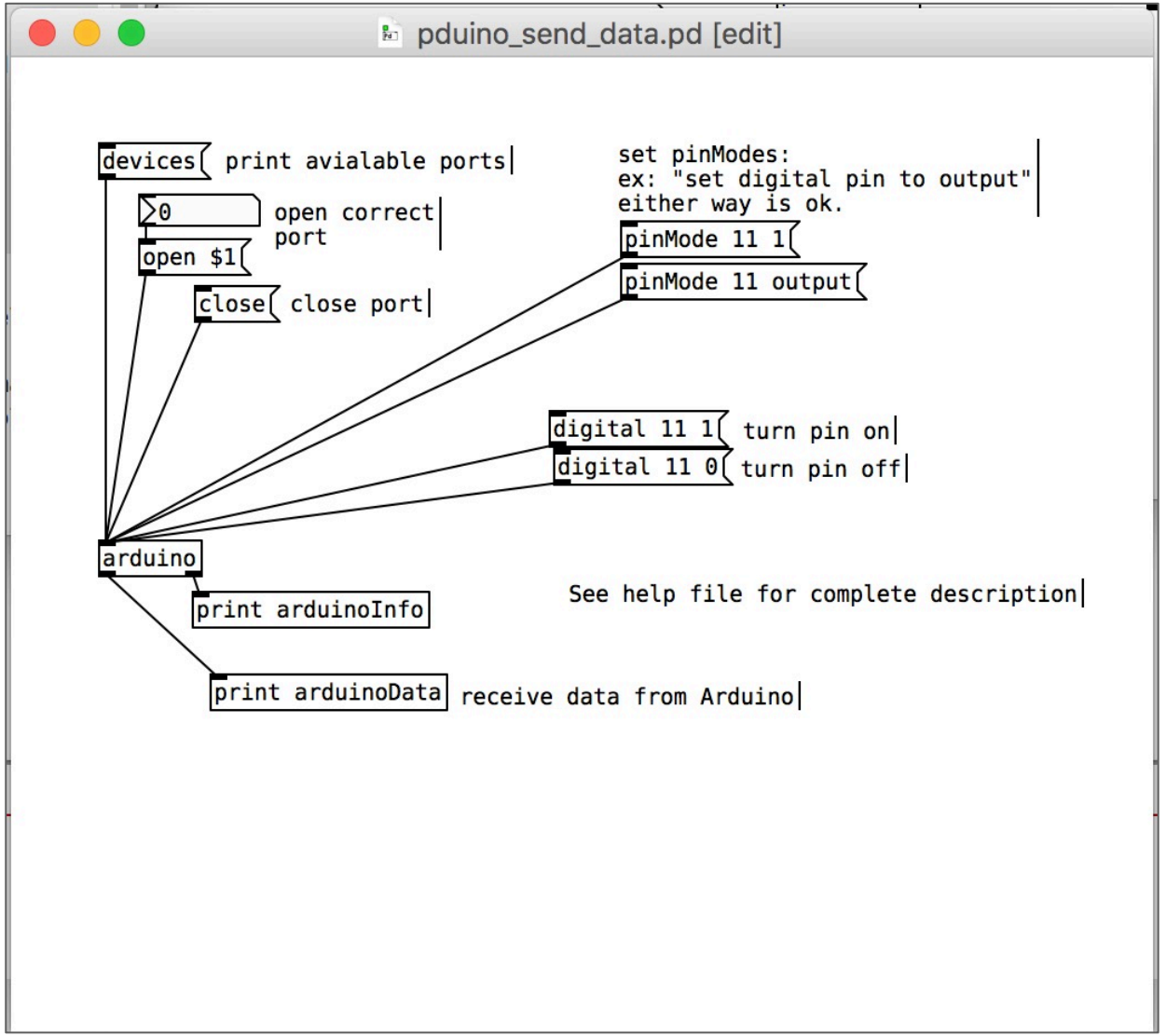


* to set analog pins you need to use their digital pin. Ex: Pin A0 would be set:

[pinMode 18 analog] or [pinMode 18 2]

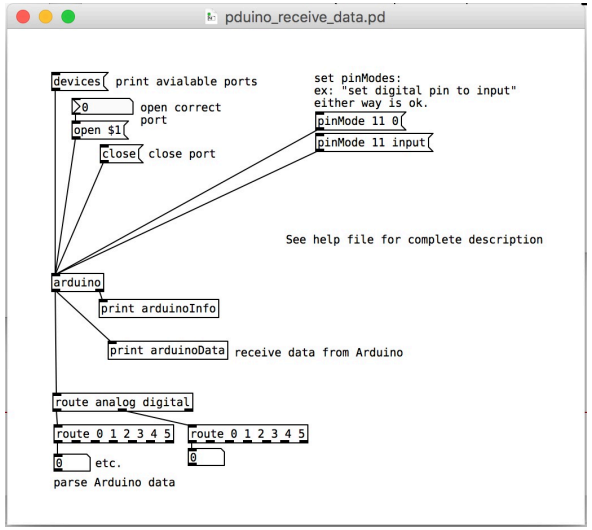
Ex: lecture-2/pduino_receive_data.pd

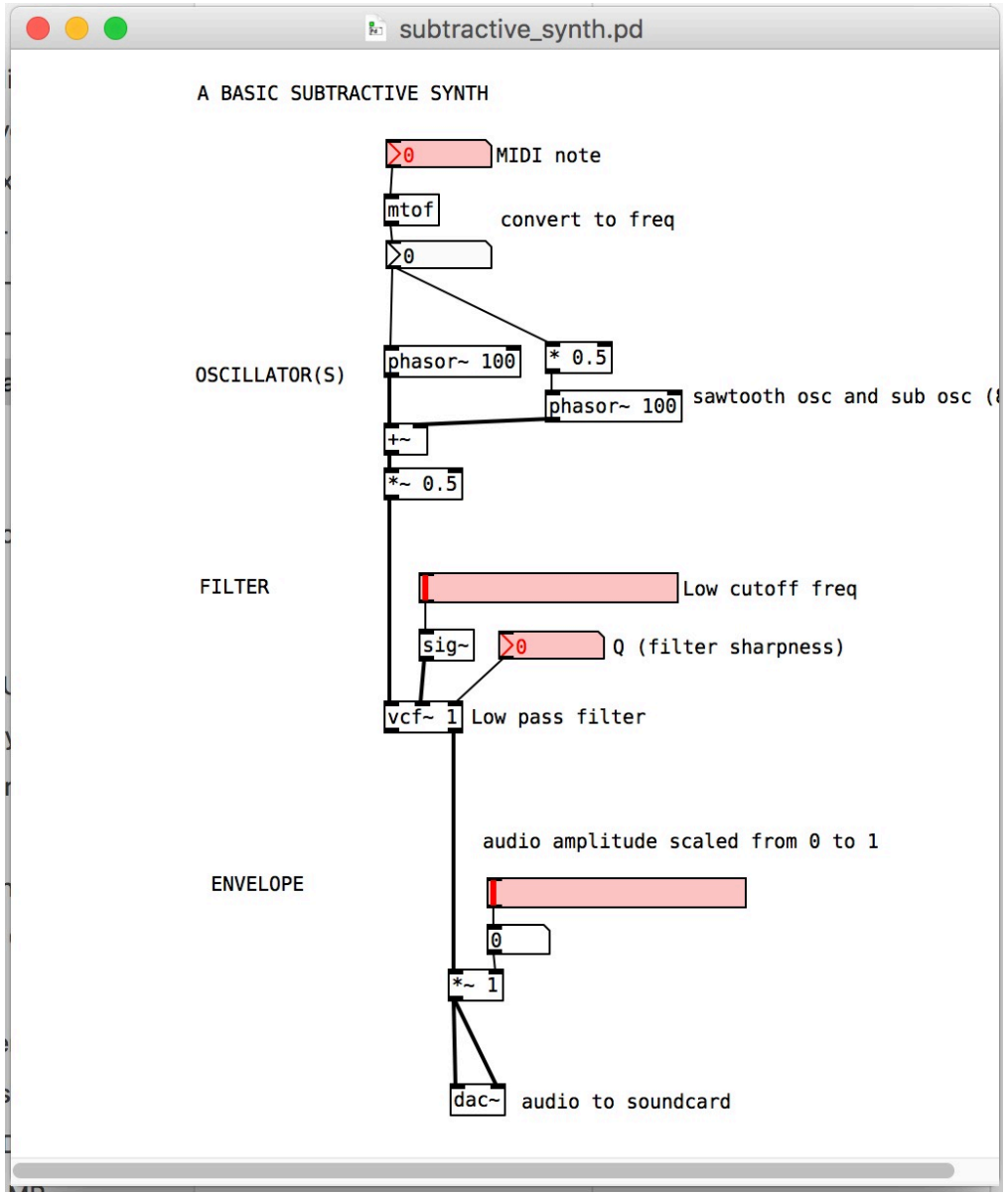
Pduino: Send data to Arduino



Ex: lecture-2/pduino_send_data.pd

Pduino: Control a simple synth in PD



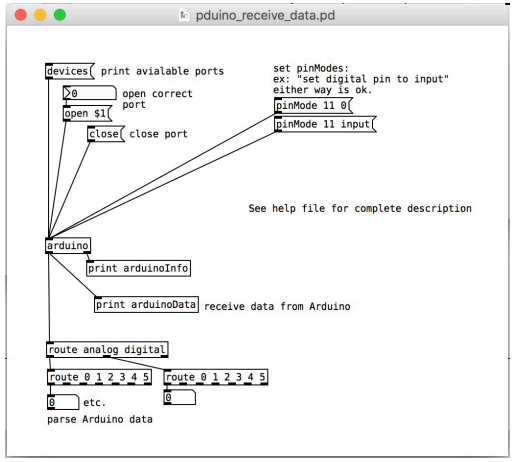


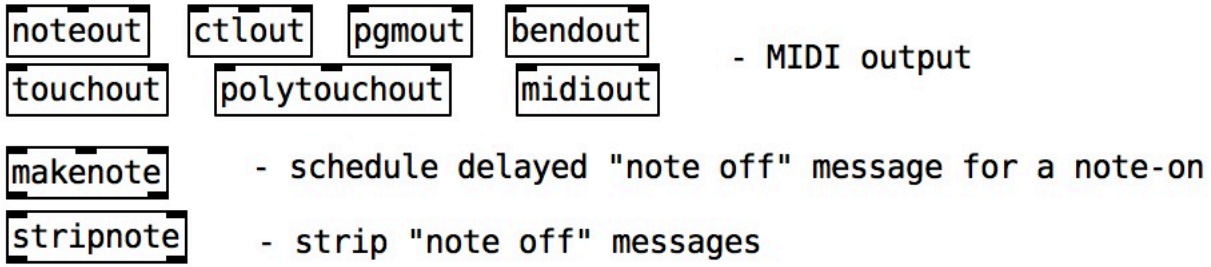
Ex: lecture-2/subtractive_synth.pd

- connect data from Arduino to synth parameters
  - Need to scale appropriately with math
  - use operators +, -, *, or /; or use [expr]

Pduino: Translate data to MIDI mesages

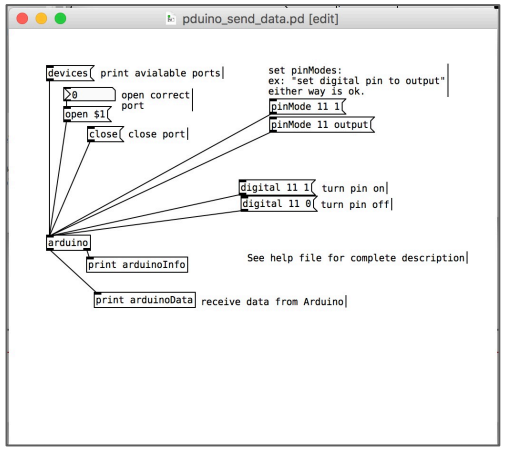
- to control other commercial softwares or instruments (like Traktor or Ableton Live), you will probably want to send MIDI messages.
  - Need to scale appropriately with math (use operators +, -, *, or /; or use [expr]
  - convert to MIDI using the Pure Data midi objects (see help files for objects below)
  - Need a virtual MIDI device to route the messages:
    - MacOS: built-in IAC Driver
    - Windows: install loopMIDI
    - Set this as the MIDI Output Device in Pure Data
    - Set as MIDI Input Device in receiving software





Receive MIDI mesages

- You can also receive MIDI messages from other MIDI-capable software or hardware
- Parse the MIDI data using the Pd objects below (see help files for usage)
- Scale appropriately and send to Arduino to provide active feedback for an interface
- Examples:
  - A metronome click lighting an LED
  - Control a servo
  - piezo speaker

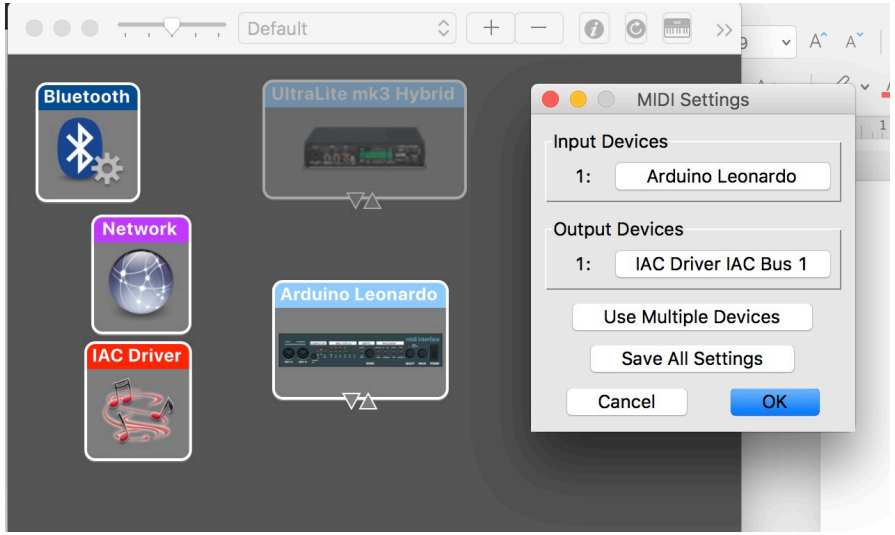


notein ctlin pgmin bendin touchin midiin
polytouchin sysexin midirealtimein

- MIDI input

MIDIUSB with Arduino Leonardo

- Certain Arduino models, including the Arduino Leonardo, can be used directly as USB MIDI devices, that will natively bind to a MIDI port and can be used in host software.
- To do this, you need to install the MIDIUSB library in your Arduino IDE
  - ‘Sketch’ menu > Include Library > Manage Libraries
- Include the library in your sketch, and your Leonardo will now show up as a MIDI device.
  - The library includes functions to send and receive standard MIDI messages. See examples (MIDIUSB_write.ino) for usage.



Ex: Control other software with Leonardo

Now we will create a basic interface with the provided sensors sending MIDI messages to software (Ableton Live for this example)

- Return to last week's Arduino sketches that read sensor data
- Combine this with the MIDIUSB functions to set up some basic controls for your software – most common will be:
  - Control Change
  - Noteon and Noteoff messages
  - Experiment with Pitch Bend, Aftertouch, or Program Change.

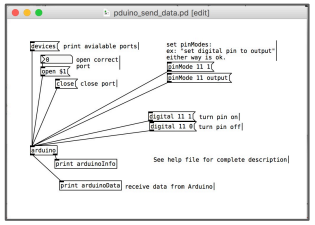
Example: lecture-2/sensor_to_midi/sendor_to_midi.ino

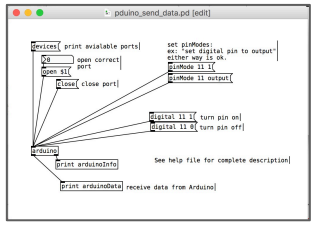
...and control Leonardo with other software

Of course, we can also send MIDI the other way too, so our other software can send MIDI messages directly to our interface (the Leonardo)

- in your other music software (DAW, soft synth, DJ software, etc.), choose relevant parameters that can be sent out via MIDI to provide feedback to the performer.
- See the Arduino 'MIDIUSB_read.ino' example, and modify it to control some things on the Leonardo (ie., blink LED in time, etc.).

Review of techniques:



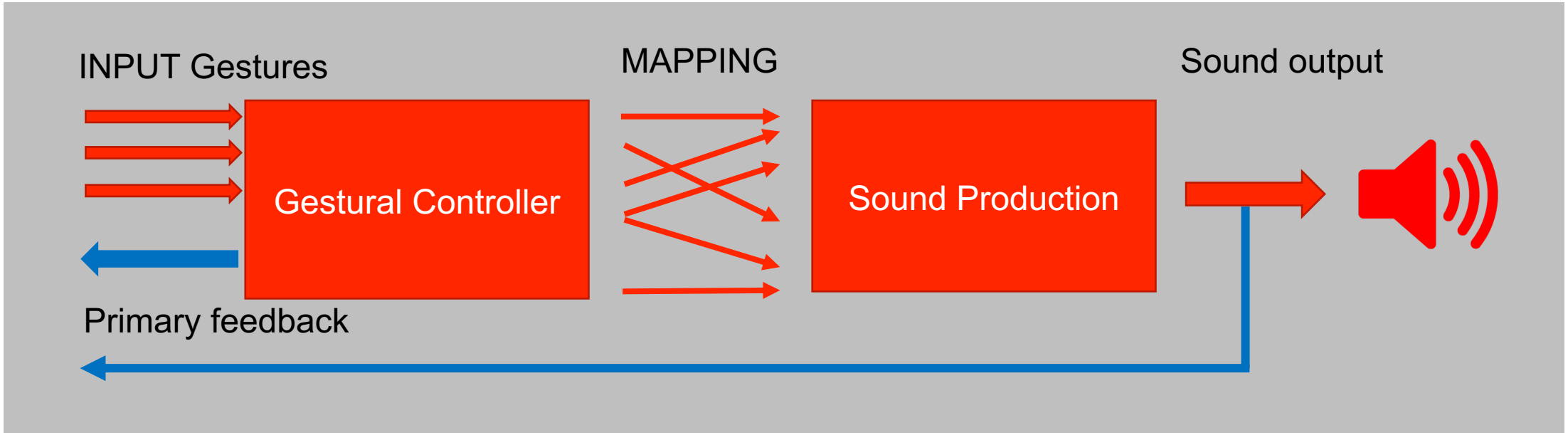


- Pduino: send and receive data between Arduino and Pd via serial.
  - Good: quick and easy. No C++ needed.
  - Bad: overly simple, can only send serial data (numbers), can't handle OSC or MIDI
- Pd MIDI formatting: send and receive MIDI between Pd and other music software.
  - Good: create custom digital interfaces in Pd; translate other input data into standard MIDI format
  - Bad: If connecting with an interface, need a middle Pd layer to do conversion
- MIDIUSB: Communicate directly between certain Arduinos (incl. Leonardo) and other music software.
  - Good: no need for Pd layer, can write more sophisticated routines to be processed on the Arduino
  - Bad: Can only handle MIDI messages
  - Good/bad? Requires C++ (Arduino) programming

Resources: https://gitlab.com/johnnyvenom/javeriana

Workshop (if there is still time):

- Review the model of a Digital Musical Instrument:



- Create your own with all of the elements: gestural controller (Arduino and sensors), mappings, sound production (can be other software), and feedback.
- You can use whatever techniques work best for your use case, and use more than one technique if it suits.