1 Meta

- Add Lucas to authors
- add early collaborators like Chris, mackenzie, etc. who gave information and insight.
- add contribution statement
- add citations for bpod and pycontrol papers, i think they have come out since this!!!

1.1 Formatting

- TOC spacing!
- Smart quotes for single quotes
- Margin note positioning throughout (do at end though)

2 Intro

2.1 Foreword to Manuscript v2

- Check the changelog!

2.2 Preamble

- ~~Add: Also experimental software is a missed opportunity and a strong lbockage to our ambitions of "open science" — making an unnecessary rite of passage in struggling through behavioral setups~~
- ~~This is actually a challenging technical and theoretical problem. Tension between totalizing framework and integrating many tools: autopilot tries to be mindful about both — it is a logical framework for the different parts of an experiment, allows easily hooking into each of these components, but also lets them be used separately. Autopilot was designed to *play nice* with other tools, but also give them a place to play nice with each other.~~
- ~~"whats missing is a glue framework"~~
- ~~**Reproducibility para -** Mention of the wiki and plugin systems — using a linux system brings with it all a manner of technical challenges. We have designed autopilot as a system that can integrate fluid technical knowledge along with the code to use it. Also that we have designed it to let work at all stages of "completeness" work together with its plugin framework — example with a stepper motor. thsi one might not be **exactly** what you need but it's better than nothing.~~
- Note that autopilot is a continually developing system, so we don't necessarily do everything here perfectly, many things are in progress, but this document is intended to capture the ideas that animate its development.

2.3 Existing Systems

- "These are the minimum components for a complete behavior system, but there are of course many other things, means of conceptualizing and supporting. Each of these is far from monolithic and has within it a thousand design choices."
- add transforms to "data management"
- Update description for pyControl
- update description for BPod
- link to finite state machine section from "pyControl uses a finite state machine"
- Bpod also has pyBpod and whatever else
- remove "Bpod blocks operation" it's not necessary – refocus this section on just describing the architectural decisions without including shit talking the GUI. In general, the purpose of this section is to illustrate two prior approaches to solving this problem to draw contrast with what we do differently. Can also discuss the mode by which it's used (eg. talk about the IBL needing to make the bigass PDF) and maybe try and find some individual use cases of each that are given as examples. These are designed for individual labs to use as tools, rather than designed as a means for organizing technical knowledge within or across labs (let alone share data).

2.4 Limitations of existing systems

- Documentation is a big problem – don't have low-level docs so people can inspect what's going on. Researchers are expected to interact with it on a top-level only: the tool as is, rather than expand and hack on the library itself. Flexible tools need to have interfaces at multiple levels: yes, just use, but also be able to extend, and modify the existing system. This is an architectural, interface, and documentation question and it's a hard one to solve!
- **New Item: Community Tooling** - No facility for sharing tasks, hardware development, etc. No good means of sharing practical usage advice – basically the community tooling isn't there. Look for examples of Bpod tasks (eg. the IBL task) and how that doesn't make for a cumulative body of knowledge. Use the christof koch quote about organization being the major problem: in order to be a force multiplier, need to be able to have peer-to-peer knowledge sharing instead of treating a piece of software as a fixed entity.
- Data - no schemas, no formal structures, sharing is hard and both take the approach that you need to write a whole additional library to
- Devops and code tooling – Bpod stuff has no means of integrating other code because matlab doesn't have a package manager. No CI, no tests for either
- the bottom line is that these tools weren't designed with integration in mind, ithey might handle some subset of the experiment, but they don't really play well with the other parts of the experiment and ultimately serve as yet another complicated node rather than something that can be compared.

3 Design

- We take a broad lens on reproducibility – not just good data, but a tool that integrates into a broader ecosystem of tools and reduces labor duplication beyond itself. Reproducibility is not just a question of getting everyone to upload their data, but to build the tools that make reproducibility accessible to a broader base of researchers.
- Though only the GPIO is really unique to the raspi – autopilot can be used on any unixlike system.

3.1 Efficiency

- integrated circuit -> microcontroller
- efficiency of interpreted languages is not the only consideration in efficiency: how easy it is to use and the time spent troubleshooting and making do with other systems is itself part of the equation, so we try and balance the ease of use witht he performance of the program
- **Concurrency** - Arguably the most radical concurrency model in autopilot is the use of multiple comptuers. So if a single processor can't do what you want it's trivial to have other computers running other things. This also lets different system components be isolated from one another, something that blends into the notion of agents and behavioral topologies discussed later.
- **Low-Level** - We make use of Python to mature an API that we will then drop down into lower level languages like C and Rust when they are stable.

3.2 Flexibility

- **Modularity** - There is a tension between flexibiltiy and providing a complete behavioral package, to balance that we have to be very capable of keeping our code modular. We do that by implementing a sensible inheritance hierarchy with predictable and shared APIs for different objects. That way we can keep a predictable set of methods exposed while being able to modify the other
- **New Section: Open Ecosystem** - split apart modularity section into a discussion about being able to pick and choose different parts of the library with a new section about the choice to use the raspberry pi and a generalize dhardware approach rather than an ecosystem of specific hardware objects. Describe the wiki here?
- **New Section: Plugins** - replace structured expansino/code transparency section with plugin sectino. By separating local configuration files and etc. we are able to have a very flexible plugin system: users are not expected to modify files in the repository to use it (which makes it difficult to contribute later) or implement everything in a single script. instead we provide a system for expansion that scaffolds their use of the software and reuse in tasks. Expand the discussion of the task library to a general purpose pick and choose your own tool library with plugins that can be maintained by you but be part of a larger project.
- CI is also really relevant here – to make sure that it's possible to change the system freely, we have CI and as we have been developing have been

3.3 Reproducibility

- New to data management section: schema stuff! We want to provide a well annotated data structure that can be exported to many different formats. To do that we have built a bunch of data modeling tools
- Also working on a related ingest project to be able to automatically bring in data from other sources made during use.
- Change standardized task decriptions to being about schemas. Tasks and data use a shared representation (along with the subject class) that lets you export descrptions as JSON, but also into NWB and etc.
- Self-documenting data -> be more explicit that we record *everything*
- **New Section: Knowledge Organization** - There is a ton of additional informatino that's needed to perform any experiment that doesn't fit neatly in code or docs, hence the wiki! To make things both *technically* reproducible AND *practically* reproducible, the plugin system is built into a semantic wiki that can organize structured data (wiki stuff here for sure, not in the other place i said)
- Update prices!

4 Structure

- Start by describing the hierarchical object strucutre – Autopilot is designed around a few fundamental types with abstract methods that are then elaborated. This is in progress!
- **New Section: Transforms** - also include DLC stuff.
- Stuff that all autopilot objects have – sick logging (give example). Autopilot is about making best practices baked into the process.

4.1 Tasks

- Update task header to use Pydantic models.
- end FSM section being like "rather than trying to standardize on a particular model for tasks, we standardize on *use*" – We provide a set of minimal tools that can be used to write tasks, but then purposefully leave the space open for people to write them how they want, and then allow standardization to happen naturally rather than presupposing a structure and asking people to conform to it. Cite aaron swartz description of standards from his book.

4.2 Hardware

- Update link to refer to the wiki rather than the hardware page on autopilot website. also mention build guides and whatever. Also describe vision of being ableo to declare specific part numbers for hardware objects to be able to document the parameters of their use.
- System adaptability section seems redundant with separating task params. consolidate this.

4.3 Stimuli

- You can also use a jetson or some other SBC with a video card if performance is more important. as we did in (cite DLC paper)
- remove stimulus and reward manager section, we don't really have them.

4.4 Agents

- Remove child agent.
- talk about managing dependencies for particular types of agents here, more relevant than terminal vs. pilot etc.
- Talk about scripts and system configuration here.

4.5 Networking

- Rewrite to say that the backbone "station" structure is being phased out in favor of a point to point communiocatino system. For now you can use point to point communication if timing is important, but otherwise you can use the hierarchical networking structure for now.
- remove treelike

4.6 GUI

- Change framing so we're talking about moving the GUI towards being a visual representation of the rest of the data modeling systems, so that different components of the system automatically get interfaces. Decoupling all the logic of the GUI into general system methods that the GUI can just visually present.
- We're going to change the plotting stuff to be built into the data description, but for now it's independent.

5 Tests

- Functionality testing is done in our CI, so these are just rough performance characterizations that can always change. So as much as it doesn't make sense to publish performance characterization in a static paper, we'll do some of that here
- Add version numbers to each of the test in margin note.
- Update language on tests. We have been working on autopilot continuously, # commits have happened, etc.
- Remove Bpod comparison tests. too messy. Then also remove language at the top that we improve efficiency an order of magnitude
- **CODE:** replace JSON with msgpack.
- **Re-run all tests**

  - Sound latency
  - Network latency
  - Network throughput
- Make sure to actually NTP sync!
- **NEW Tests**
  - GPIO on/off latency
  - networked GPIO on/off latency
  - Camera FPS for picam/FLIR cam?
- Just say we haven't done any work on visual stimuli so this test was run with the release version of autopilot.