Home Cage Software Design

Ryan Cameron
October 28^{th}, 2020
Donaldson Lab, University of Colorado at Boulder

I. Introduction

The home cage system being designed in the Donaldson Lab at the University of Colorado at Boulder is meant to be a long-term behavioral experiment where prairie voles are trained to associate rewards with a partner vole. The apparatus is a three chamber system, consisting of one main chamber and two smaller, individual chambers. The SolidWorks design for the system is shown below in an isometric view.

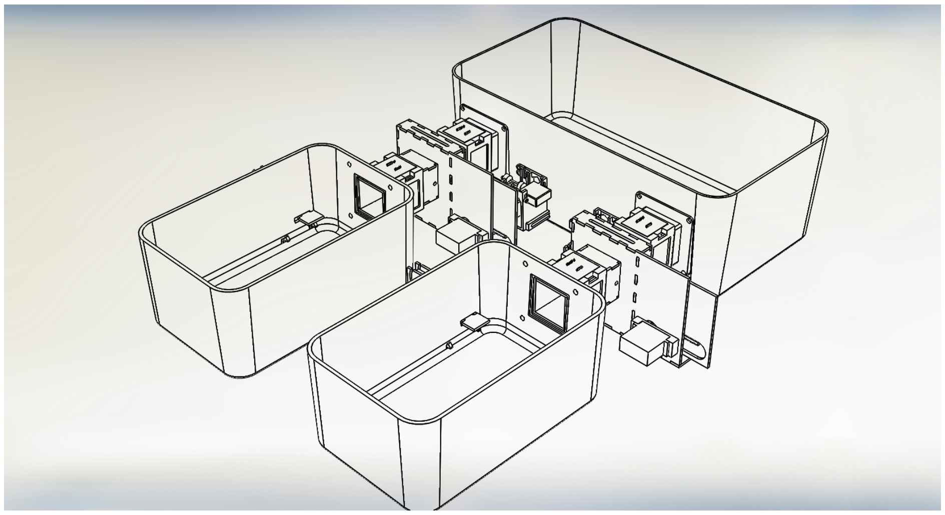


Figure 1. Isometric View of Home Cage Assembly

Between each small chamber and the large chamber, is a lever operated door, where a lever can be activated from either side by the vole. The idea here is that the voles will learn that they can press the lever to access their partner vole in the other chamber, but this setup is meant to be fully autonomous for long term behavior experiments.

The requirement to be fully autonomous leads to an interesting software challenge, which is the subject of this document. This document will outline the software package written to control and monitor the performance of the home cage setup.

II. Software Design

A. Overview

To begin there needs to be a discussion on some standards that are used throughout the code. First, as a base, the code tracks where each animal is based on a numbering system between the cages. A chart describing this numbering system is shown below.

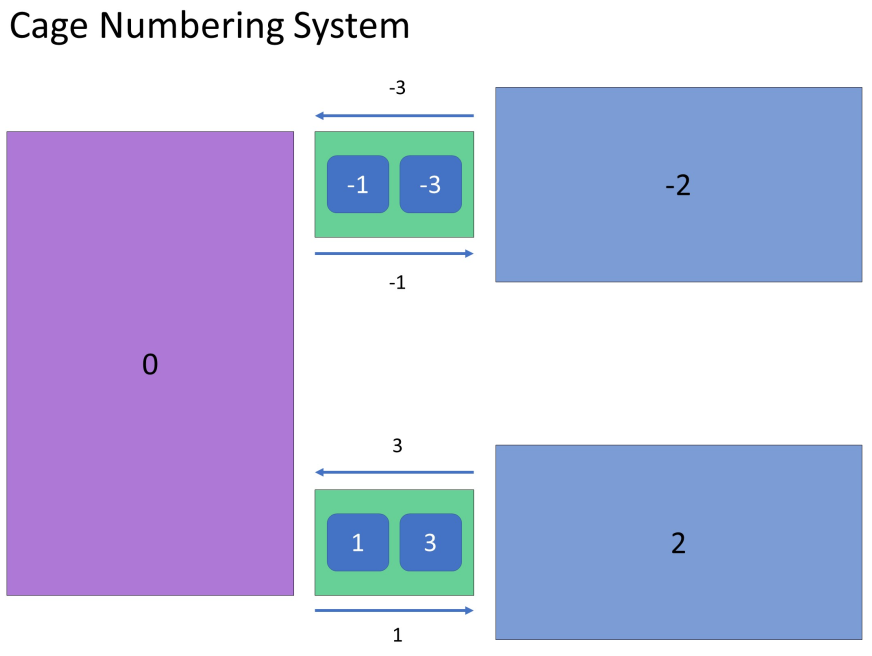


Figure 2. Numbering System Chart

In this diagram, each cage has a number: 0, 2, and -2. Then, between each cage, there is a directional value assigned: 1, 3, -1, and -3. This lets the software know not only the location of the animal but also the direction the animal is moving in. This allows the software to deduce an effective vector for the animal whenever it passes through a doorway.

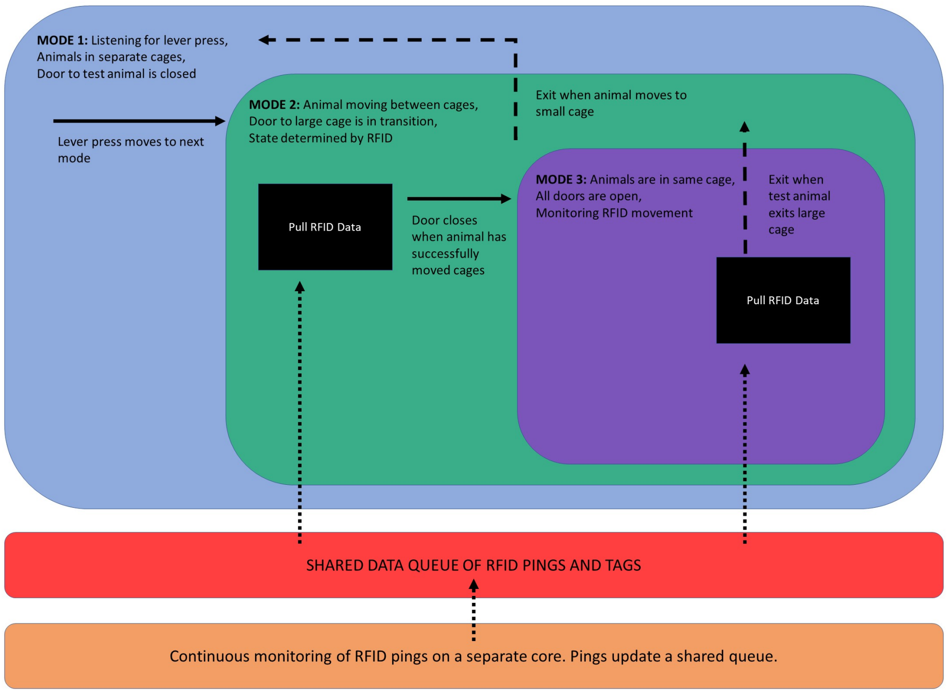
Next, the software uses multiple different python packages to control and monitor the experimental setup. First, to monitor the RFID status of each animal, therefore obtaining a location and direction of travel of each vole, it needs to monitor this status at all times no matter what is happening in the setup where doors may be opening and closing. This means that the python packages *multiprocessing* and *threading* must be used. The details of how these are implemented will be discussed later in the document.

Finally, the last section of the code that needs to be discussed is how the code will actually be controlling the setup. Python was chosen as the best programming language for the job because of the ease of use on the develpment end, also since this will be implemented through a Raspberry Pi (RPi) microcontroller system. The RPi easily reads python code because it is a Linux based OS, and this reduces the risk of something going wrong on the RPi itself.

Now to give an overview of the software structure, it is broken up by the types of data that needs to be monitored. The primary data that controls the flow of when doors open and close is the RFID tags which need to be monitored continuously so as not to miss either of the animals moving into or through a region of interest. This leads to the choice of having the RFID pings monitored on one process by the CPU, and the door logic monitored on a separate process by the CPU.

B. Control Logic

This section goes over the logic that controls the opening and closing of the doors, which is broken up into 3 modes of operation described by the diagram below.



MODE 1: Listening for lever press, Animals in separate cages, Door to test animal is closed
Lever press moves to next mode
MODE 2: Animal moving between cages, Door to large cage is in transition, State determined by RFID
Exit when animal moves to small cage
Pull RFID Data
Door closes when animal has successfully moved cages
MODE 3: Animals are in same cage, All doors are open, Monitoring RFID movement
Exit when test animal exits large cage
Pull RFID Data
SHARED DATA QUEUE OF RFID PINGS AND TAGS
Continuous monitoring of RFID pings on a separate core. Pings update a shared queue.

Figure 3. Mode Description

As seen in the diagram, the modes are actually nested inside of each other so that the software can only move between one mode at a time. This serves as a safety measure, making sure that the necessary checks are always being made before a door opens or closes.

Mode 1 is the main, idle, mode of operation for the software system. Here, RFID pings are not being pulled and the animals are not interacting at all. This means that at least one of the animals is in a side chamber. So, to enter into the next mode, the animal in the small chamber must press the lever to indicate that the door should be opened. This triggers the system to enter Mode 2, which is a transitional mode. Here, the software begins pulling and reading the continuous RFID data. A move between modes from here is determined from the pings received by the RFID data. If the animal moves back into the small cage, the door closes and the system moves back to Mode 1. If the animal moves into the large cage, the door remains open and the system moves into Mode 3. Once in Mode 3, the RFID data is being monitored and when the animal moves back into the small cage, the door closes and the system moves into Mode 2.

The design of having a transitional mode between when the animal is in the small cage (Mode 1) and when the animal is in the large cage interacting (Mode 3), ensures a level of redundancy in detecting where the animal is. This is because the cage design has an RFID pickup on either side of the door to the small cage, so to successfully move between Mode 1 and Mode 3, the system needs two affirmative RFID pings indicating the direction of travel of the vole. The software decision tree for how the software transitions between modes is shown below.

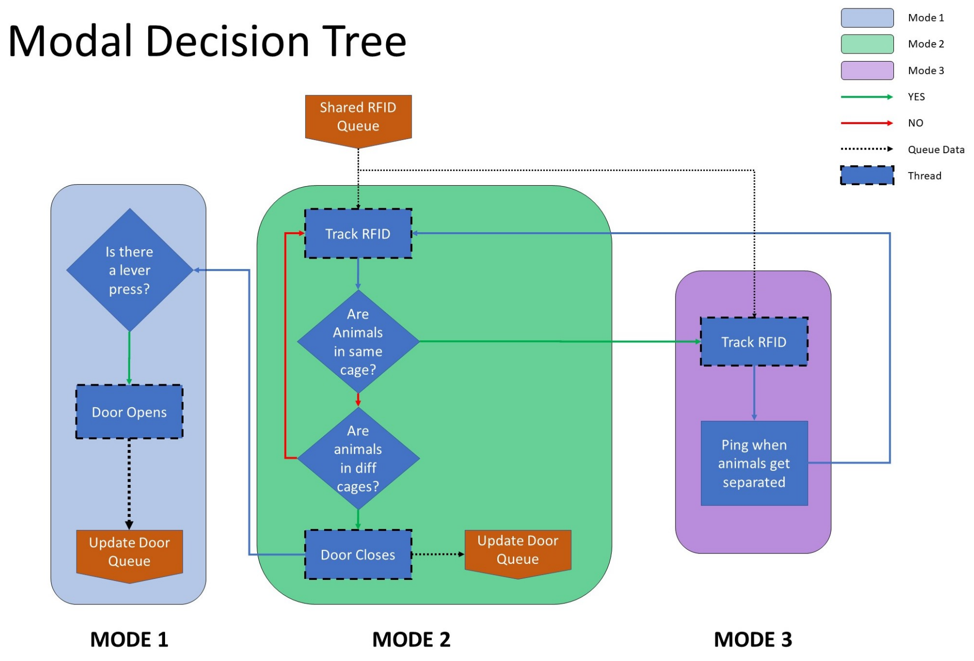


Figure 4. Modal Decision Tree

C. RFID Logic

The RFID process was a bit trickier to implement due to the constraint that it needed to be continuously searching for RFID pings from either animal, not interrupted by the modal logic outlined previously, and also able to transmit real-time data to the modal logic if needed. This was done by putting all of the RFID logic on a separate process from the modal logic, and within the RFID process, each RFID tracker (of 4) is put on its own thread. The logic for each thread is shown in the diagram below.

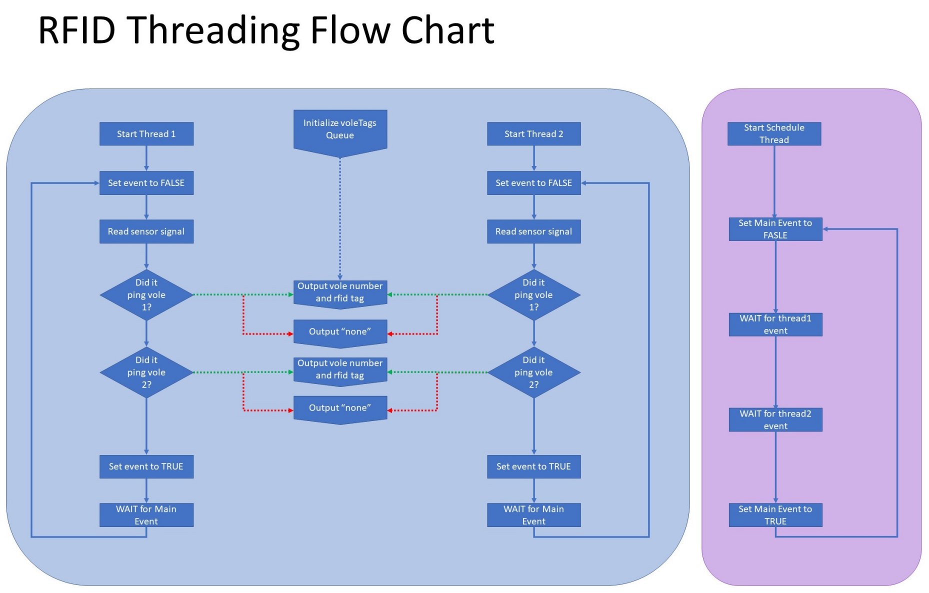


Figure 5. RFID Thread Logic

This shows a simple logic flow where some setup steps occur first before going into a loop where it checks if a vole has pinged, outputs the result of the check to a shared data queue, and then waits for the other threads to end before starting the loop over. This ensures that the timing on each of the threads outputting data to the shared queue is relatively similar. The difference in time between the threads ending at this point is small enough to be negligible.

In this section, the timing is managed by a separate thread, one that is called a schedule thread. This is connected to the other RFID threads through events and event listener objects in python. The logic here is that each thread must set its event value to TRUE, and once it does, the schedule thread recognizes it and moves on to listening for the next thread, or setting the main event to TRUE. At the end of each RFID thread, there is a *wait()* command that waits for the main event to be set to true before restarting the loop. Another benefit of this is that more RFID threads can be added with the same logic, and in the schedule thread, just another step is added to make sure the new thread's event is set to TRUE as well. This way, the system can scale up or down easily.

III. Conclusion

With this logic in place, the software side of the homecage system should be able to run autonomously and continuously with the necessary safety checks in place.