We've been connecting the teensy to the nidaq on the pcb board and reading out data into MATLAB through the daq. As long as you run the quietOp.ino teensy should continue to automatically process sensor data coming in. You can use a simple daq like the usb-6001. https://www.ni.com/en-us/shop/model/usb-6001.html

I've attached a initDAQ file that reads in sensor data as 3 analog input channels. You would need to call this function during the initialization code of virmen, and then during the runtime on each virmen iteration you can access the daq data as daqData(1:3).

Yes you need a movement function to determine how sensor changes are converted to changes of mice in the virtual environment. Noah's function should work and allow you to change forward position, lateral position and view angle as the mouse moves.

Thanks for your interest in mouseVR. In the Harvey Lab we use two different ways of reading hardware signals into virmen on this setup.

Use a National Instruments DAQ in conjunction with analog and digital inputs/outputs from the behavior PCB. There is a header on the board to pug a USB DAQ in, or you can use the BNC plugs to interface with it with a board of your choice. The teensy is still required for interfacing with the ball sensors and outputting PWM signals which are then smoothed by an RC filter into analog voltages that can be read in using any DAQ of your choice. The instructions for setting this up are under the ball sensor software section of the readme. That will take you as far as getting analog signals coming out proportional to pitch roll and yaw velocities of the ball, but getting those into virmen is beyond the scope of the guide. Typically it would be advisable to use a background operation to read in 3 analog channels and assign to global variables. Unfortunately with the new versions of the nidaq toolbox in matlab I believe these background callbacks are limited to 10Hz which is too slow, so I am not sure exactly how to approach this (in part because I did not do things that way). Perhaps a simple solution would be to spawn a separate headless instance of matlab that continuously reads in values and writes them to a UDP port, which could be read by the instance running virmen. But I have not tried that, I used the next solution.

Read in directly from the teensy using serial communication. The basic scheme is that on every iteration of virmen you send a command to the teensy telling it the duration of time to open each reward valve (usually 0, since valves are closed most of the time). That will trigger the teensy to send back a string containing the current data from the board, which is read in on the next iteration. So every iteration you read incoming data once and write the valve state once. If you want better control over timing you can use the usb nidaq, but note that it has some of its own limitations. Specifically, on the arduino end, the arduino listens for a string from the computer that indicates how long in milliseconds to open the reward valve for. Most of the time this will be "000000". That message serves to trigger the reward valve when needed, but also prompts the arduino to return a string with all the behavioral data (lick count (an incremented lick count, NOT binary), ball signals, reward valve states, sync pulses). In the virmen runtime function, virmen checks if there is data on the teensy port. If it is there (which it is 99.9% of the time), I read it in and parse the string into its values, then write out a new serial command (usually "000000") which will trigger any reward if needed and prompt the arduino to send back behavioral data to be read in on the next iteration. This means that behavioral data is at most 1 iteration old (16.7ms), which is usually fine for the tasks I have run. You could wait on the same iteration for the teensy to respond, but serial communication isn't super fast and the teensy is doing a lot of other stuff which means its own runtime loop is not the fastest. Thus if you wait for the teensy to reply on the same iteration, you might drop a frame, which is worse in my opinion.

I have attached some example code. It may not work directly out of the box but is a good starting point.

Files and description:

- behaviorPCB\_v1\_2.ino

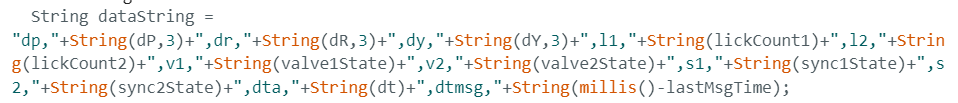
code to upload to the teensy using the same instructions as on the repo.

The important things here are that it keeps a running count of licks so you that never miss a lick. It checks for a message on the serial port expecting 6 integers. the first 3 are the number of milliseconds to open valve 1, the second 3 are the number of milliseconds to open valve 2. i.e. 010000 would open valve 1 for 10 milliseconds and would not open valve 2.

To use this, compile and upload to teensy using the same instructions as for the behavior code listed in the repo docs. Note down which COM port the teensy is on.

The data string is formatted as follows:

String dataString = "dp,"+String(dP,3)+",dr,"+String(dR,3)+",dy,"+String(dY,3)+",l1,"+String(lickCount1)+",l2,"+String(lickCount2)+",v1,"+String(valve1State)+",v2,"+String(valve2State)+",s1,"+String(sync1State)+",s2,"+String(sync2State)+",dta,"+String(dt)+",dtmsg,"+String(millis()-lastMsgTime);



Note that it returns cumulative lick count (it behaves as a counter channel), not number of licks since last message, so within virmen, you should keep track of the last cumulative reading and check whether it has increased or not to see if mouse has licked since last reading

- teensyTest.m

Shows basic process of connecting to teensy, controlling the valves, and getting data back

- parseTeensyMessage.m

function for parsing the data string sent back from teensy to matlab

- calibrateRewardTeensy.m

First part of this you will not be able to run as it depends on some of my other functions.

But after line

%% CALIBRATE REWARD TEENSY - MINIMAL DEPENDENCIES

you should be able to run the code.

The basic calibration process for the gravity-fed solenoid reward system is as follows:

- create a list of valve opening durations.

- determine for each valve opening duration how much liquid comes out the spout (e.g. run that pulse duration X times to yield an accurately measurable quantity of reward liquid, then divide that by X to get per-pulse mL quantity)

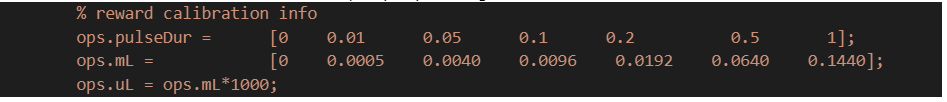
- create a second list that lists how much liquid per pulse. eg

% reward calibration info

ops.pulseDur = [0 0.01 0.05 0.1 0.2 0.5 1];

ops.mL = [0 0.0005 0.0040 0.0096 0.0192 0.0640 0.1440];

ops.uL = ops.mL\*1000;



- store that info e.g. in a function you can call from various virmen experiments

when delivering reward, on each virmen iteration you can construct the appropriate valve command for how much liquid you want to delivery. e.g.

switch units

case 'uL' % default

pulseDur = interp1(ops.uL,ops.pulseDur,amount,'linear','extrap'); % pulse duration

case 'mL'

pulseDur = interp1(ops.uL,ops.pulseDur,amount\*1000,'linear','extrap'); % pulse duration

case 'pulseDur'

pulseDur = amount;

end

pulseDur(pulseDur>0.999) = 0.999;

vr.serialCommand = [sprintf('%03i',round(pulseDur\*1000)) sprintf('%03i',round(pulseDur\*1000))];

disp(vr.serialCommand);

vr.reward = vr.reward + amount;

return

文本

描述已自动生成

-exampleCodeTeensySerialCommunication

Demonstrates how you would check for bytes on the serial port on each iteration, read in the data, send the new serial command. I tried to keep it very minimal

You can use pitch roll and yaw values from parse teensy message within your movement function. you will need to apply appropriate gain value(s). The gain can be calibrated so that each virmen unit is a defined distance unit. This is done by knowing the circumference of the ball and rotating it x number of times, recording movement in vr, then calculating the gain that results in the desired number of units moved in VR.

Note I'm no longer in the harvey lab and I don't have regular access to a mouse vr rig so testing example code such as the stuff i'm sending you is difficult for me. Assume that there will be some troubleshooting and testing required. good luck and let me know if you have any questions.