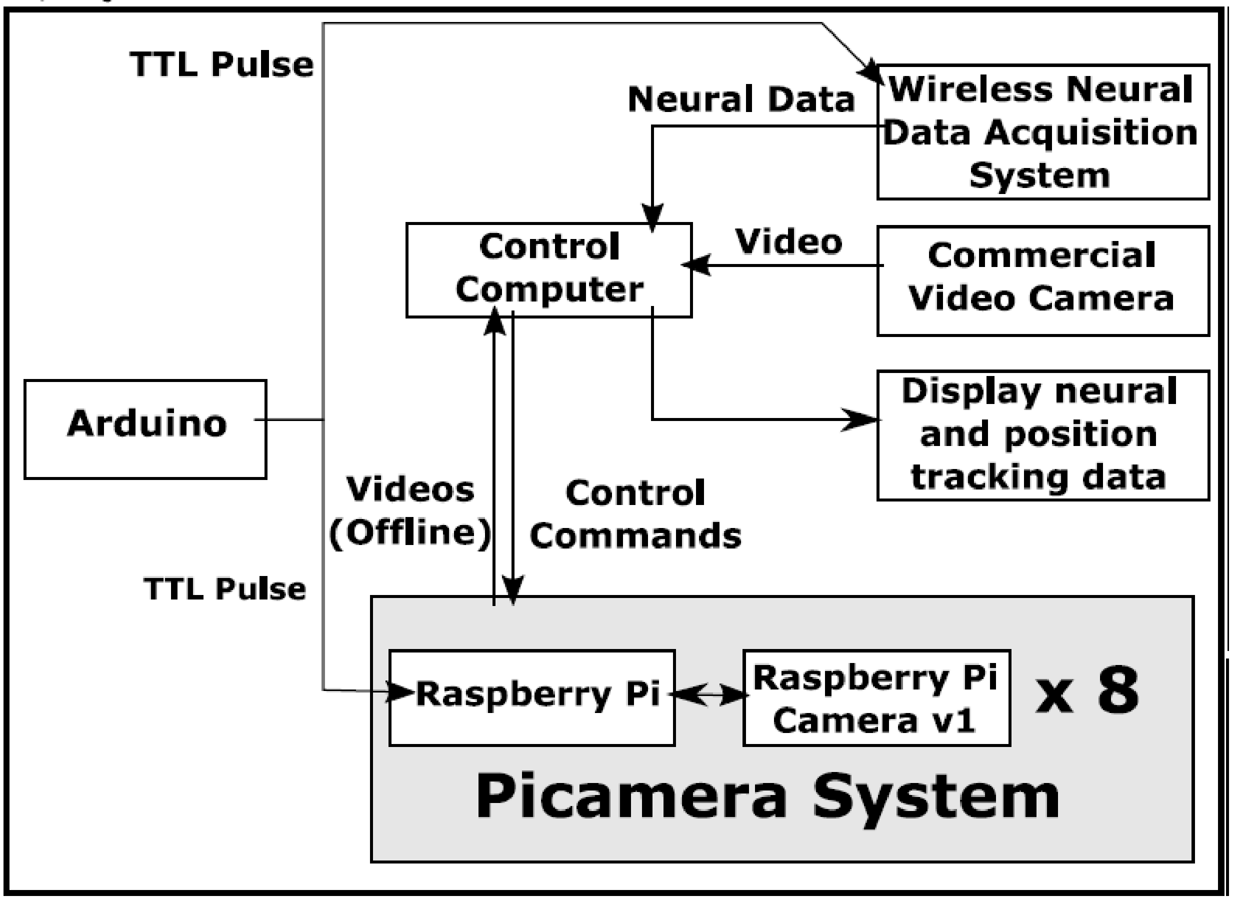
PicameraTrackingSystem.md

Recording System

```
from IPython.display import Image
```

```
Image(filename = 'system.png' )
```



Neural Recording Hardware

Comprises the following components:

- **Data Acquisition System**

  Recordings are performed using DL 4SX Cube-64 Package manufactured by Neuralynx Inc. The system comprises Cube-64 Headstage, DL 4SX Base and Cheetah Software.

- **Control computer**

  A standard personal computer (PC) with Windows 7 (Microsoft) operating system acts as the control computer which has been used to develop, test and operate the proposed system. The computer outfitted a Xeon processor @ 3.7 GHz and 6GB RAM.

- **Commercial camera (suppplied with electrophysiology system)**

Picamera System

The Picamera system was developed using 8 overhead static cameras built using Raspberry Pi Board and Raspicam module v1. Components of the video recording system:

- **Raspberry Pi**

  A credit card sized full computer which has 1GB RAM 900MHz quad-core Processor, 32GB SD card for booting.

  We are using the Raspbian OS on raspberry Pi. The image file for the respective can be downloaded from raspberry pi website (https://www.raspberrypi.org/downloads/). This image file is burnt on 32 GB memory card using the Win32Disk utility.

Note: All the cameras have pi as their user name and raspberry as their password. More information available on this website (https://www.raspberrypi.org/)

- **Raspberry Pi camera module v1**

  Raspicam is an Omni Vision OV5647 ColorCMOS QSXGA 5-megapixel sensor with f/2.9 aperture lens which is compatible with raspberry pi.

  The camera can record 1080p, 720p, 640x480p video with 30fps. Each camera has a field of view of 2.43m x 1.82m at 2.8m height. (https://www.raspberrypi.org/documentation/hardware/camera/README.md)

- **Arduino**

  An open-source microcontroller based physical computing platform that can be used for variety of projects.

- **Ethernet Hub**

  DGS-1024D (D-Link) ethernet hub was used to remotely operate each Raspberry Pi.

Software Required for Operating Picamera system

Following software will be required to operate and download data from cameras.

Note: For all these software, ensure that the Ethernet cable from the hub (to which all the Raspberry Pi cameras are connected) should be connected to the control computer.

Putty (https://www.putty.org/)

used to remotely access the picamera via network. All one needs to do is enter the ip address of the camera with user name and password.

MTPutty (http://ttyplus.com/multi-tabbed-putty/)

Allows to remotely access the picamera same as putty. The advantage of mtputty over putty is the multi execution mode. This mode allows broadcasting same command to all the Picamera system at the same time thus allowing to control all of them simultaneously using this.

Filezilla (https://filezilla-project.org/)

Filiezilla is the file sharing client which allows sftp, ftp (File Transfer Protocol) along with easy to use GUI. Using this software one can easily upload/ download files from any Picamera system. After starting the client just enter the pi ip address, user name and password.

Setting up internet on Raspberry Pi

Configuring wired internet settings on the Raspberry Pi.

Note: To setup internet access on Pi, you need root user privileges. To gain root user privileges, run the following command on terminal sudo su which will prompt for password, type the password and press enter. This will grant root user privileges.

Here are the steps:

- Open(or create) the file /etc/network/interfaces (command: cd /etc/network/interfaces)

- Add these lines to the file

  auto eth0

  iface eth0 inet static

  address 10.120.10.x**

  gateway 10.120.10.1

  network 10.120.10.0

  netmask 255.255.255.0

  broadcast 10.120.10.255

(**x is a number between 200 to 250, check IPs already being used otherwise there will be a conflict)

- Last step Add DNS server : Open /etc/resolve.conf and write

nameserver 10.16.25.15

- Run sudo apt-get update to check if internet is running properly

NOTE

For further settings. https://www.modmypi.com/blog/tutorial-how-to-give-your-raspberry-pi-a-static-ip-address

Ways to stream video

- **VLC**
  - Install VLC on both Pi and remote machine
  - Type this command on Pi terminal:
    raspivid -o - -t 0 | cvlc -v stream:///dev/stdin --sout '#standard{access=http,mux=ts,dest=:8080' :demux=h264
  - Open VLC player on remote Desktop and go to Media >>Open Network Stream
  - Add IP address of Pi : 8080(port number) and play
- **Zmodo and video composite output**
  - Look at the TTL&Zmodo.pdf in the parent directory

Network Time Protocol (NTP) server settings

Data acquisition system was used as the NTP server. The server was used to broadcast its time across all the Picamera sub-units. This allowed to maintain same time (not on video frame timestamp) on all the Picameras which was then used to naming to output video and timestamp file on each Picamera.

To set up NTP server, install ntp (sudo apt-get install ntp) on Pi (or other Linux based machines). Download the NTP software from Meinberg (https://www.meinbergglobal.com/english/sw/ntp.htm) for Windows based machines.

This link (http://www.satsignal.eu/ntp/setup.html) is useful for installation guidelines.

Here are the steps involved:

- When asked to specify the configuration settings, do not choose any of the pool based NTP servers. A warning message will appear when you press Next; ignore this.
- Next, when asked if you want to see the config file, click Yes, and add the following lines (after the line about drift files)

  server 127.127.1.0
  fudge 127.127.1.0 stratum 0

  This causes the system to use the local system clock as its server.
- Now in /etc/ntp.conf comment out the lines listing a pool based NTP server. Add the line :-

  server (IP address of computer to act as the main server) iburst to the document.
- Also, comment out the lines

  restrict 127.0.0.1
  restrict ::1.

- Save the file, and restart the NTP service with the following command at the command line: sudo service ntp restart

NOTE Check the date by typing date in the command line. Within a few minutes, the system should have updated itself to match the time on the master system.

Video acquistion scripts using Picamera Python library and changes required to get accurate timestamps

The code (StartAcquisition_gpio.py) displayed underneath is used for recording. Two threads run simultaneously, one collects the video related data (video + frame timestamps) whereas the other thread saves the ttl timestamps data. In this approach for each TTL input transition (detected using Rpi.GPIO library), timestamps using *Picamera python* library are logged.

```
# %load StartAcquisition_gpio.py
#import the necessary modules
import io
import time
import datetime as dt
from picamera import PiCamera
from threading import Thread, Event
from queue import Queue, Empty
import sys, getopt
import argparse
import RPi.GPIO as GPIO
import os

#set high thread priority
os.nice(20)

#camera parameter setting
WIDTH  = 640
HEIGHT = 480
FRAMERATE = 30
VIDEO_STABILIZATION = True
EXPOSURE_MODE = 'night'
BRIGHTNESS = 55
CONTRAST = 50
SHARPNESS = 50
SATURATION = 30
AWB_MODE = 'off'
AWB_GAINS = 1.4
#TTL Pulse BounceTme in milliseconds
BOUNCETIME=10
camId = str(4)

#video, timestamps and ttl file name
VIDEO_FILE_NAME = "cam" + camId + "_output_" + str(dt.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")) + ".h264"
TIMESTAMP_FILE_NAME = "cam" + camId + "_timestamp_" + str(dt.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")) + ".csv"
TTL_FILE_NAME = "cam"+ camId + "_ttl_" + str(dt.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")) + ".csv"

#running time variable intialization
runningTimeHours, runningTimeMinutes, runningTimeSeconds = 0,0,0

#set raspberry pi board layout to BCM
GPIO.setmode(GPIO.BCM)
#pin number to receive TTL input
pinTTL = 17
#set the pin as input pin
GPIO.setup(pinTTL, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
#add event detection (both falling edge and rising edge) script to GPIO pin
GPIO.add_event_detect(pinTTL, GPIO.BOTH, bouncetime=BOUNCETIME)

#video output thread to save video file
class VideoOutput(Thread):
    def __init__(self, filename):
        super(VideoOutput, self).__init__()
        self._output = io.open(filename, 'wb', buffering=0)
        self._event = Event()
        self._queue = Queue()
        self.start()

    def write(self, buf):
        self._queue.put(buf)
        return len(buf)

    def run(self):
        while not self._event.wait(0):
            try:
                buf = self._queue.get(timeout=0.1)
            except Empty:
                pass
            else:
                self._output.write(buf)
                self._queue.task_done()

    def flush(self):
        self._queue.join()
        self._output.flush()

    def close(self):
        self._event.set()
        self.join()
        self._output.close()

    @property
    def name(self):
        return self._output.name

#timestamp output object to save timestamps according to pi and TTL inputs received and write to file
class TimestampOutput(object):
    def __init__(self, camera, video_filename, timestamp_filename, ttl_filename):
        self.camera = camera
        self._video = VideoOutput(video_filename)
        self._timestampFile = timestamp_filename
        self._ttlFile = ttl_filename
        self._timestamps = []
        self._ttlTimestamps = []

    def ttlTimestampsWrite(self, input_pin):
        inputState = GPIO.input(input_pin)
        GPIO.remove_event_detect(input_pin)
        if self.camera.frame.timestamp is not None:
            self._ttlTimestamps.append((inputState, self.camera.timestamp, self.camera.frame.timestamp, time.time(),
                                        time.clock_gettime(time.CLOCK_REALTIME)))
        else:
            self._ttlTimestamps.append((inputState, self.camera.timestamp, -1, time.time(),
                                        time.clock_gettime(time.CLOCK_REALTIME)))
        #print(inputStatem, self.camera.timestamp, self.camera.frame.timestamp)
        GPIO.add_event_detect(pinTTL, GPIO.BOTH, bouncetime=BOUNCETIME)

    def write(self, buf):
        if self.camera.frame.complete and self.camera.frame.timestamp is not None:
            self._timestamps.append((
                self.camera.frame.timestamp,
                self.camera.dateTime,
                self.camera.clockRealTime
                ))
        return self._video.write(buf)

    def flush(self):
        with io.open(self._timestampFile, 'w') as f:
            f.write('GPU Times, time.time(), clock_realtime\n')
            for entry in self._timestamps:
                f.write('%d,%f,%f\n' % entry)
        with io.open(self._ttlFile, 'w') as f:
            f.write('Input State, Timestamp, GPU Times, time.time(),
                    clock_realtime\n')
            for entry in self._ttlTimestamps:
                f.write('%f,%f,%f,%f,%f\n' % entry)

    def close(self):
        self._video.close()

parser = argparse.ArgumentParser()
parser.add_argument("-hr","--hours",type=int,help="number of hours to record")
parser.add_argument("-m","--minutes",type=int,help="number of minutes to record")
parser.add_argument("-s","--seconds",type=int,help="number of seconds to record")
args = parser.parse_args()

runningTimeHours = float(args.hours)
runningTimeMinutes = float(args.minutes)
runningTimeSeconds = float(args.seconds)

totalRunningTime = runningTimeHours*60*60+runningTimeMinutes*60+runningTimeSeconds

with PiCamera(resolution=(WIDTH, HEIGHT), framerate=FRAMERATE) as camera:
    camera.brightness = BRIGHTNESS
    camera.contrast = CONTRAST
    camera.sharpness = SHARPNESS
    camera.video_stabilization = VIDEO_STABILIZATION
    camera.hflip = False
    camera.vflip = False

    #warm-up time to camera to set its initial settings
    time.sleep(2)

    camera.exposure_mode = EXPOSURE_MODE
    camera.awb_mode = AWB_MODE
    camera.awb_gains = AWB_GAINS

    #time to let camera change parameters according to exposure and AWB
    time.sleep(2)

    #switch off the exposure since the camera has been set now
    camera.exposure_mode = 'off'

    output = TimestampOutput(camera, VIDEO_FILE_NAME, TIMESTAMP_FILE_NAME,
                             TTL_FILE_NAME)
    GPIO.add_event_callback(pinTTL, output.ttlTimestampsWrite)
    try:
        camera.start_preview()
        # Construct an instance of our custom output splitter with a filename  and a connected socket
        print('Starting Recording')
        camera.start_recording(output, format='h264')
        print('Started Recording')
        camera.wait_recording(totalRunningTime)
        camera.stop_recording()
        camera.stop_preview()
        print('Recording Stopped')
    except KeyboardInterrupt:
        output.close()
        print('Closing Output File')
    except:
        output.close()
        print('exception! output file closed')
    finally:
        output.close()
        print('Output File Closed')
        GPIO.cleanup()
```

Changes required for accurate timestamping

Following things need to be taken care for the code above to work properly:

1. Download picamera python library source code from github https://github.com/waveform80/picamera
2. Picamera library is written in a way such that each frame is timestamped in relative (*reset*) manner i.e. first frame is timestampped at 0. But for our purpose we decided to use *raw* timestamps i.e. actual timestamps rather than first frame timestamps subtracted from consecutive frames.
3. Here are the timestamps used:
   - **camera.frame.timestamp** - The timestamp is measured in microseconds (millionths of a second). When the camera's clock mode is `'reset'` (the default), the timestamp is relative to the start of the video recording. When the camera's :attr:~`PiCamera.clock_mode` is `'raw'`, it is relative to the last system reboot. See :attr:~`PiCamera.timestamp` for more information.
   - **camera.timestamp** - The camera's timestamp is a 64-bit integer representing the number of microseconds since the last system boot. When the camera's :attr:`clock_mode` is `'raw'` the values returned by this attribute are comparable to those from the :attr:`frame` :attr:~`PiVideoFrame.timestamp` attribute.
4. Rpi.GPIO module is used to detect TTL events (falling and rising edges) and record above timestamps for each TTL event. We use a bounce time for 85ms for 100ms width.

STEPS

- Change the following line in 'PicameraLibraryModified/picamera/camera.py' file

  ```
  cc.use_stc_timestamp = clock_mode
  ```

  to

  ```
  cc.use_stc_timestamp = mmal.MMAL_PARAM_TIMESTAMP_MODE_RAW_STC
  ```

OR

- Change

  `def` **`init`**`(self, camera_num=0, stereo_mode='none', stereo_decimate=False,resolution=None, framerate=None, sensor_mode=0,led_pin=None,clock_mode='reset')`

  to

```
 def __init__(self, camera_num=0, stereo_mode='none', stereo_decimate=False,resolution=None, framerate=None,
sensor_mode=0, led_pin=None,clock_mode='raw')
```

in the 'PicameraLibraryModified/picamera/camera.py' file.

- Add following code to 'PicameraLibraryModified/picamera/camera.py' file after * def _get_timestamp (self):* function

```
def _get_timeofday(self):
    self._check_camera_open()
    return time.time()
dateTime = property(_get_timeofday, doc="""
    Retrieves the system time according to the python time.time() module
    """)

def _get_clockRealTime(self):
    self._check_camera_open()
    return time.clock_gettime(time.CLOCK_REALTIME)
clockRealTime =  property(_get_clockRealTime, doc="""
    Retrieves the system time according to the CLOCK_REALTIME
    """)
```

Operating Picamera system for recording

Ensure that all the Raspberry Pi have the *Picamera Python library* (*http://picamera.readthedocs.io/en/release-1.13/install.html*) installed on them

Steps to follow while setting up cameras for recording:

- Ensure that all the cameras have following wires plugged in:- Power cable to power hub, ethernet cable to ethernet hub, cables for digital IO and video output to Zmodo.

- Check whether the ethernet cable from the hub is connected to the Data Acquisition system before switching on the raspberry pi. The reason behind this is if you switch on the pi without first connecting to the acquisition machine, time across pi and acquisition machine won’t be same.

- Once the above check is cleared, switch on the raspberry pi. Connect to each of them using the MTPutty in multiexec mode.

- To check if all the cameras have been switched on with the same date time information, type date in the command terminal of each pi. This command will tell you the date and time on each camera. If the date command on all the pi cameras doesn't show you exact same time, you can do one of the following:

  - wait for some time for the camera time to synchronize with control computer.
  - Restart all the cameras and make sure that the ethernet cable from the ethernet hub is plugged into control computer CPU before restart.

- For each pi, navigate to the folder where StartAcquisition_gpio.py file is stored. This is the script required for starting to acquire on each camera.

- As soon as you are prepared to record, run the StartAcquisition_gpio.py using the following command:

```
sudo nohup python3 StartAcquisition.py -hr hours -m minutes -s seconds

    -hr: number of hours of recording
    -m: number of minutes of recording
    -s: number of seconds of recording
```

So an example run for 2 hours 30 minutes and 20 seconds recording would be:

```
sudo nohup python3 StartAcquisition_gpio.py -hr 2 -m 30 -s 20
```

nohup (no hangup) instructs the OS to run the script in background. It is important to use nohup because all non essential software except the Cheetah acquisition software running on the DAQ computer needs to be shut down while collecting neural data to avoid interference with neural data acquisition.

- The StartAcquisition_gpio.py will let the cameras record for mentioned amount of time. At the end of recording, we have 3 files for each pi camera used:
    - Timestamp file for each frame (.Csv)
    - TTL timestamp file for each transition (.Csv)
    - video file (.h264)

The files are named as cam*_video_datetime, cam*_timestamp_datetime and cam*_ttl_timestamp with * getting replaced by camera number and datetime getting replaced with the date time values of when the recording started.

- All of these files can be downloaded using the Filezilla Client. All one needs to do is enter ip address, username and password and then right click and select download on the files that need to be downloaded.

Script to stop recording

Use the killrecording.py script stops the video recording process

```
# %load killrecording.py
#!/usr/bin/python

import os, subprocess, signal

p = subprocess.Popen(['ps','-ax'], stdout = subprocess.PIPE)
out, err = p.communicate()

for line in out.splitlines():
    if 'StartAcquisition' in line:
        pid = int(line.split(None, 1)[0])
        os.kill(pid, signal.SIGKILL)
```