Appendix A:

AHF_TagReader (Murphy et al. 2016) : Python code for communication between Raspberry Pi and RFID reader. Each part of the scripts are described by comments. By using this class, Raspberry Pi can communicate with the RFID reader and get the serial number of a RFID-tag as a decimal value as well as a Tag-In-Range signal indicating that the RFID-tag is held in the reading range of the RFID. The hardware parts and assembly were described before. This class requires Python 3.4+ and Pyserial library. Raspberry Pi 2 has been used here, however if you want to use Raspberry Pi 3 you need to specify a different serial port (follow this website: http://spellfoundry.com/2016/05/29/configuring-gpio-serial-port-raspbian-jessie-including-pi-3/)

#!/usr/bin/python
#
#-*- coding: utf-8 -*-

import serial

class AHF_TagReader:

    """
    Class to read values from a Innovations RFID tag reader, such as ID-20LA
    """
    def __init__(self, serialPort, doChecksum = False):
        """
        Makes a new AHF_TagReader object
        :param serialPort: serial port tag reader is attached to, /dev/ttyUSB0 or /dev/ttyAMA0 for instance
        :param doCheckSum: set to calculate the checksum on each tag read
        """
        try:
            self.serialPort = serial.Serial(str(serialPort), baudrate=9600)
        except IOError as anError:
            print ("Error initializing TagReader serial port.." + str(anError))
            raise anError
        if (self.serialPort.isOpen() == False):
            self.serialPort.open()
        self.serialPort.flushInput()
        # set boolean for doing checksum on each read
        self.doCheckSum = bool(doCheckSum)

    def clearBuffer(self):
        """
        Clears the serial buffer for the serialport used by the tagReader
        """
        self.serialPort.flushInput()
def readTag(self):
    """
    Reads a hexadecimal RFID tag from the serial port using a blocking
    read and returns the decimal equivalent

    RFID Tag is 16 characters: STX(02h) DATA (10 ASCII) CHECK SUM (2
    ASCII) CR LF ETX(03h) 1 char of junk, 10 of hexadecimal data, 2 of
    hexadecimal check sum, 3 of junk

    :returns decimal value of RFID tag
    :raises IOError: if serialPort not read
    raises ValueError: if checksum or conversion from hex to decimal
    fails
    """
    serialJunk = self.serialPort.read(1)
    serialTag = self.serialPort.read(10)
    serialCheckSum = self.serialPort.read(2)
    serialJunk = self.serialPort.read(3)
    if serialJunk.__len__() < 3:
        self.serialPort.flushInput()
        raise IOError
    try:
        decVal = int(serialTag, 16)
        except ValueError as anError:
            print("TagReader Error converting tag to integer: " + str(serialTag) + ': ' + str(anError))
            self.serialPort.flushInput()
            raise ValueError
    else:
        if self.doCheckSum == True:
            if self.checkSum(serialTag, serialCheckSum)== True:
                return decVal
        else:
            print("TagReader checksum error: " + str(serialTag) + ': ' + str(serialCheckSum))
            self.serialPort.flushInput()
            raise ValueError
    else:
        return decVal

def checkSum(self, tag, checkSum):
    """
    Sequentially XOR-ing 2 byte chunks of the 10 byte tag value will
give the 2-byte check sum

    :param tag: the 10 bytes of tag value
    :param checksum: the two bytes of checksum value
    :returns: True if check sum calculated correctly, else False
    """
    checkedVal = 0
    try:
        for i in range(0,5):
            checkedVal = checkedVal ^ int(tag [(2 * i) : (2 * (i + 1))], 16)
        if checkedVal == int(checkSum, 16):
return True
else:
    return False
except Exception:
    return False

def __del__(self):
    if self.serialPort is not None:
        self.serialPort.close()

if __name__ == '__main__':
    serialPort = '/dev/ttyAMA0'
    doCheckSum = True
    nReads = 3
    try:
        tagReader = AHF_TagReader(serialPort, doCheckSum)
        for i in range (0, nReads):
            print(tagReader.readTag())
        print('Read ' + str(nReads) + ' tags')
    except Exception:
        print('Tag reader not found, check port ' + serialPort)
Appendix B:

Scale.py: Python class that interacts with a custom-written C++ module (HX711, code available on request) to control an HX711 load cell amplifier. When run independently, i.e., not included from another module, the `__main__` function provides code in order to calibrate and test the load cell we need a chunk of python code that’s developed by J. Boyd. After running this class, by insetting Entering “1” the load cell, while there is with no load weight on the load cell it except the chamber, will record the be turned to zero (unloaded state -tare weight). By inserting 2, Entering “2” load cell makes gets an average of 10 readings from the load cell; entering and also “3” does a is for one time reading. Finally, “-1” exits the program.

```python
import HX711 # the c++ module, HX711.cpython-34m.so
from array import array

"""HX711 is a c++ module that does the GPIO clocking and data reading from the HX711 load cell in c++ speed using the wiringPi library. The c++ approach was necessary as the Python GPIO code was not fast enough to be reliable - the HX711 sleeps if the clock pin is held high for 60 ms, and this would sometimes happen when running the GPIO with Python."
"""

class Scale:
    """Class to operate a scale based on a HX711 load cell"
    """
    def __init__(self, dataPin, clockPin, gmPerUnit, initGPIO):
        self.hx711ptr = HX711.new (dataPin, clockPin, gmPerUnit, initGPIO)
        self.threadArray = array ('f', (0 for i in range (0,100)))
        self.arraySize = 100

    """Records the tare value, which will be subtracted from subsequent scale reads. The tare value is stored int he c++ object
    nAvg: number of readings to average together to get tare
    printVal: if set, will print the tare value as well as save it"
    def tare(self, nAvg, printVal):
        HX711.tare (self.hx711ptr, nAvg)
        if printVal == True:
            print ('Tare value is', self.getTareVal())

    """Returns an averaged weight value in grams with tare subtracted"
    def getAveragedWeight(self):
        # implementation...
```

Returns an averaged weight value in grams with tare subtracted
and scaling applied
nAvg: the number of weight reading to average to get returned value

def weigh (self, nAvg):
    return HX711.weigh (self.hx711ptr, nAvg)

Returns weight from a single reading, in grams with tare and scaling applied
def weighOnce (self):
    return HX711.weigh (self.hx711ptr, 1)

Reads the HX711 once and returns a signed integer value without tare or scaling applied.
def readIntValue (self):
    return HX711.readIntValue ()

Resizes the array that is passed to the C++ module and is used for reading the scale rapidly from an independent thread
def threadSetArraySize (self, newSize):
    self.threadArray = array ('f', (0 for i in range (0,newSize)))
    self.arraySize = newSize

Tells the C++ thread to start reading from the HX711 and placing results in the array. When this function returns, the thread continues reading until stopped or the array is full
def threadStart (self, size):
    if size > self.arraySize:
        size = self.arraySize
    HX711.weighThreadStart(self.hx711ptr, self.threadArray, size)

Tells the C++ thread to stop reading and filling the array
def threadStop (self):
    return HX711.weighThreadStop (self.hx711ptr)

Returns the number of weights already in the array, but does not tell the thread to stop
def threadCheck (self):
    return HX711.weighThreadCheck (self.hx711ptr)

Sets the scaling in grams/24-bit A/D units. Value is stored in the C++ object
def setScaling (self, newScaling):
    HX711.setScaling (self.hx711ptr, newScaling)

    """
    Gets the grams/24-bit A/D unit scaling stored in the C++ object
    """
    def getScaling (self):
        return HX711.getScaling (self.hx711ptr)

    """
    Gets the tare value in raw 24-bit A/D units from C++ object
    """
    def getTareVal (self):
        return HX711.getTareValue (self.hx711ptr)

    """
    Returns number of the GPIO pin used by the Pi to generate clock signal
    """
    def getClockPin (self):
        return HX711.getClockPin (self.hx711ptr)

    """
    Returns number of the GPIO pin from which the Pi reads the data
    """
    def getDataPin (self):
        return HX711.getDataPin (self.hx711ptr)

    """
    Makes sure the HX711 is ready to start weighing, waking it from the low power state if needed
    """
    def turnOn (self):
        HX711.turnOn (self.hx711ptr)

    """
    Puts the HX711 into a low power state
    """
    def turnOff (self):
        HX711.turnOff (self.hx711ptr)

    """
    Main function that runs when not included from another module, used for testing, calculating scaling, and simple weighing
    """
    if __name__ == '__main__':
        DATAPIN = 23
        CLOCK = 24
        gmPerUnit = 7.152e-05
        nAvg = 10
        scale = Scale(DATAPIN, CLOCK, 7.152e-05, True)
        event = 0
        while event > -1:
event = int(input('Enter 0 to tare, 1 to weigh with avg, 2 to weigh once, 3 to turn OFF, 4 to turn ON, 5 for threaded read, -1 to quit:'))
    if event == -1:
        break
    elif event == 0:
        scale.tare(5,True)
    elif event == 1:
        print('Weight =', scale.weigh(10), 'g')
    elif event == 2:
        print('Weight =', scale.weighOnce(), 'g')
    elif event == 3:
        scale.turnOff()
    elif event == 4:
        scale.turnOn()
    elif event == 5:
        scale.threadStart(scale.arraySize)
    nReads = scale.threadCheck()
    while nReads < scale.arraySize:
        print("Thread has read ", nReads, " weights, last reading was ", scale.threadArray[nreads-1])
        nReads = scale.threadCheck()}
Appendix C:

AutoMouseWeight.py: the python program with which the Raspberry Pi controls the RFID reader and the HX711 load cell amplifier, saving recorded weights, RFIDs, and time stamps to binary data files. Note that pin number 23 and 24 were used herein for data transmitting and clock, respectively. Also pin number 18 was connected to TIR pin of RFID. If Raspberry Pi 3 is being used in this project serial port must be changed.

#!/usr/bin/python

#.*-coding: utf-8-*-
from AHF_TagReader import AHF_TagReader
from Scale import Scale
import RPi.GPIO as GPIO
from array import array
from time import time, localtime, timezone, sleep
from datetime import datetime

KSECSPERDAY = 86400

KTIMEOUTSECS=0.05

KMAXWEIGHTS = 20
tag =0
tirPin = 18
serialPort = '/dev/serial0'
tagReader = AHF_TagReader(serialPort, False)

....
Threaded call back function on Tag-In-Range pin 
Updates tag global variable whenever Tag-In-Range pin toggles
Setting tag to 0 means no tag is present 
....
def tagReaderCallback (channel):
global tag
if GPIO.input (tirPin) == GPIO.HIGH: # mouse just entered 
tag = tagReader.readTag ()
else: # mouse just left 
tag = 0

....
....
def main():

dataPin=23
clockPin=24
gmPerUnit=7.14e-05
initGPIO=True
tirPin=18

GPIO.setmode (GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup (tirPin, GPIO.IN)
GPIO.add_event_detect (tirPin, GPIO.BOTH)
GPIO.add_event_callback (tirPin, tagReaderCallback)

scale = Scale (dataPin, clockPin, gmPerUnit, initGPIO)
scale.threadSetArraySize (200)
scale.tare(5, True)

"""
A new data file is opened for each day, with a name containing the
current date
"""

today = datetime.today()
filename = 'LCC_' + str (today.year) + '_' + '{:02}'.format(today.month) + '_' + '{:02}'.format (today.day)
outFile=open (filename, 'ab')

"""
Weight data is saved as grams, in 32 bit floating point format. Each run of
data is prefaced by metadata, also in 32 bit floating point format, of the last 6 digits of the RFID code, and the time
of the run, in seconds since the start of the day. The RFID code is printed as a negative value to make it easy for
analysis code to find the start of each run
"""

metaData = array ('f', [0.0])
startofday=((int((time() - timezone + localtime().tm_isdst * 3600)/KSECSPERDAY)) * KSECSPERDAY) +
timezone - (localtime().tm_isdst * 3600)
while True:
    try:
        global tag
        while tag==0:
            """Loop with a brief sleep, waiting for a tag to be read
or a new day to start, in which case a new data file is made
"""
            if (time() > startofday + KSECSPERDAY):
                startofday += KSECSPERDAY
                outFile.close()
                today = datetime.today()
                filename = 'LCC_' + str (today.year) + '_' + '{:02}'.format(today.month) + '_' + '{:02}'.format (today.day)
            outFile=open (filename, 'ab')
        else:
            sleep (KTIMEOUTSECS)
# Place the data for ID tag and time into first 2 points of array
thisTag = tag
print ('mouse = ', thisTag)
metaData[0]= -(thisTag%100000)
metaData[1]=time()-startofday
# tell the C++ thread to start filling the array
scale.threadStart (scale.arraySize)
nReads = scale.threadCheck()
lastRead=0

"""
Keep reading weights into the array until a new mouse is read by
the RFID reader, or the last read weight drops below 2 grams, or
the array is full, then print the array to the file
"""
while (tag == thisTag or (tag == 0 and scale.threadArray [nReads-1] > 2)) and nReads < scale.arraySize:
    if nReads > lastRead:
        print (nReads, scale.threadArray [nReads-1])
        lastRead = nReads
    nReads = scale.threadCheck()
    nReads = scale.threadStop()
    metaData.tofile (outFile)
    scale.threadArray[0:nReads-1].tofile(outFile)
except KeyboardInterrupt:
    outFile.close()
    GPIO.cleanup()
    break

if __name__ == '__main__':
    main()
Appendix D:

The electronic schematic is depicted in this appendix, showing the wiring between Raspberry Pi cobbler +, RFID breakout, HX711 and load cell.