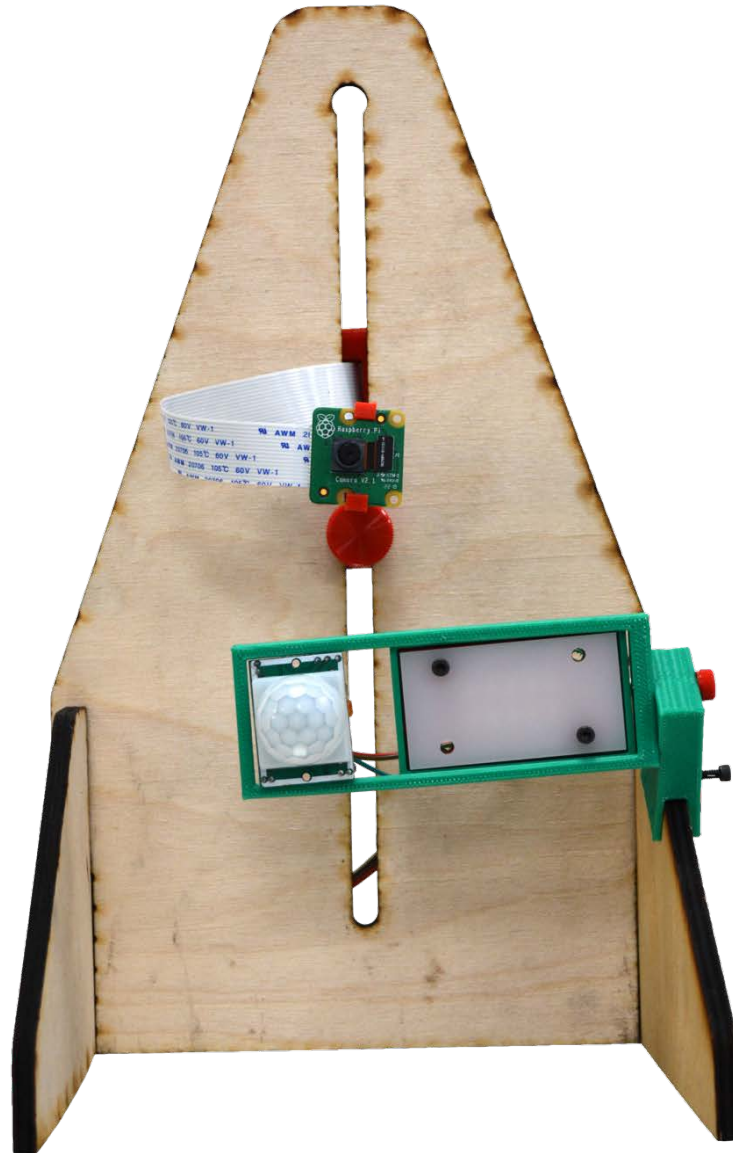


# PiSPY



The PiSpy is an affordable device used to automate the acquisition of images and videos. It was shared on bioRxiv in 2022 (<https://www.biorxiv.org/content/10.1101/2022.03.21.485129v1>). It uses a Raspberry Pi computer and camera to record video and images at specified time intervals or when triggered by an input signal, such as a motion sensor or force plate. Additionally, it can automate the control of LED lights. All settings for the PiSpy are controlled through a graphical user interface that uses the TKinter Python package. The following user guide will outline how to set up and operate the PiSpy. Our testing was with a Raspberry Pi3b or Pi4, which is primarily what this guide is written for, but with some modifications the PiSpy can also be used with older Raspberry Pi models.

## Table of Contents

<b>PiSpy Hardware Assembly.....</b>	<b>3</b>
<i>Fabrication of Components.....</i>	<i>3</i>
<i>Assembly Instructions .....</i>	<i>3</i>
<b>Setting up the Raspberry Pi Computer .....</b>	<b>7</b>
<i>Installing Raspberry Pi OS.....</i>	<i>7</i>
<b>Operating the graphical user interface .....</b>	<b>12</b>
<i>Input trigger.....</i>	<i>14</i>
<i>Light Control .....</i>	<i>14</i>
<i>Note on Red Lights at Night.....</i>	<i>15</i>
<i>Quick Capture .....</i>	<i>15</i>
<i>Preview .....</i>	<i>15</i>
<i>File Identifier .....</i>	<i>15</i>
<i>Running the PiSpy Program .....</i>	<i>15</i>
<b>Connecting GPIO sensors for input trigger.....</b>	<b>16</b>
<i>Connecting a PIR motion sensor .....</i>	<i>16</i>
<i>Connecting an IR break beam .....</i>	<i>17</i>
<i>Note on PIR motion sensors.....</i>	<i>18</i>
<i>Note on IR break beams.....</i>	<i>18</i>
<i>Note on BOARD vs BCM GPIO labeling .....</i>	<i>18</i>
<i>Note on code for input triggers.....</i>	<i>18</i>
<i>Using input trigger mode .....</i>	<i>19</i>
<i>Optional: connecting a Real-Time Clock (RTC) to the PiSpy.....</i>	<i>21</i>
<b>Adjusting Default PiSpy Settings .....</b>	<b>21</b>
<b>LED PCB Lighting .....</b>	<b>24</b>

## PiSpy Hardware Assembly

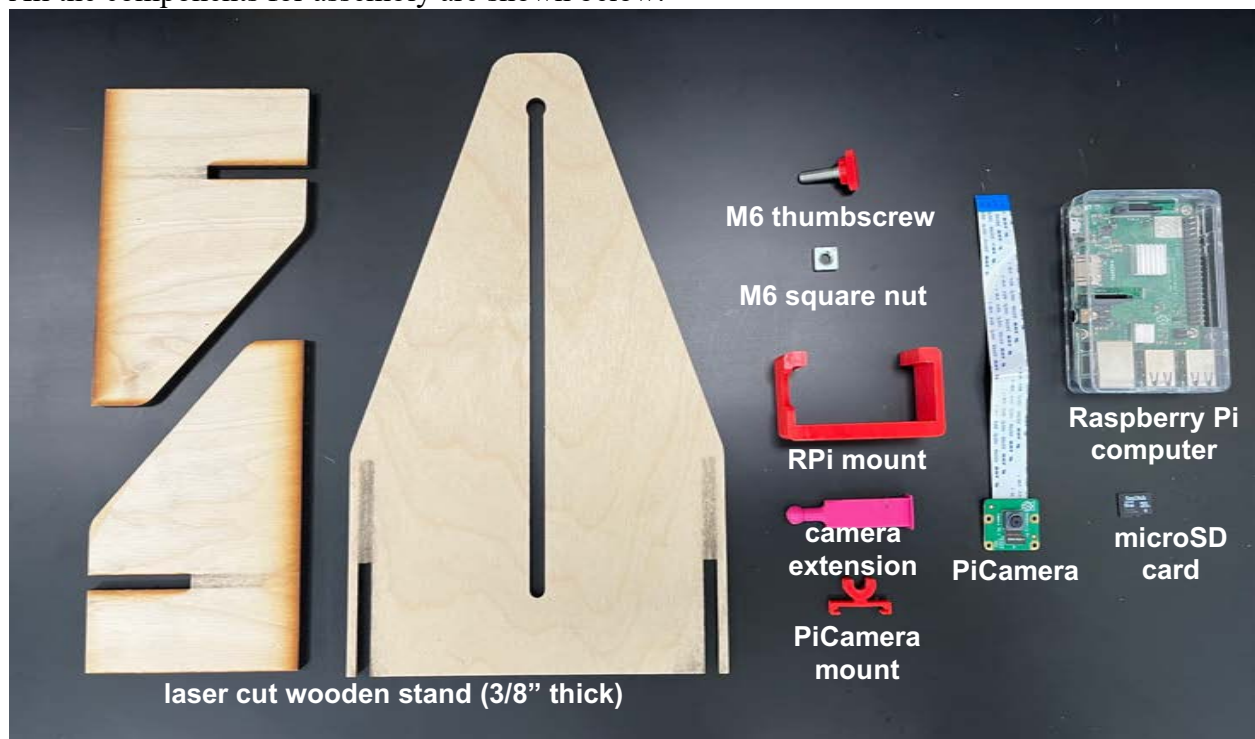
### *Fabrication of Components*

The hardware files associated with PiSpy can be downloaded from the GitHub repository (<https://github.com/gpask/PiSpy/tree/master/Hardware/Original%20PiSpy>).

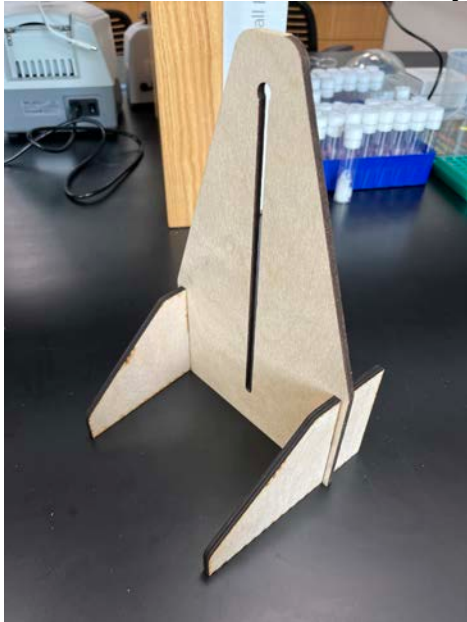
The wooden stand used to mount PiSpy can be laser cut from 3/8 inch plywood using the .svg file. The 3D-printed components were printed with polylactic acid (PLA) using the .stl files.

### *Assembly Instructions*

All the components for assembly are shown below:



First, assemble the wooden stand by sliding each support into the taller mount like so:



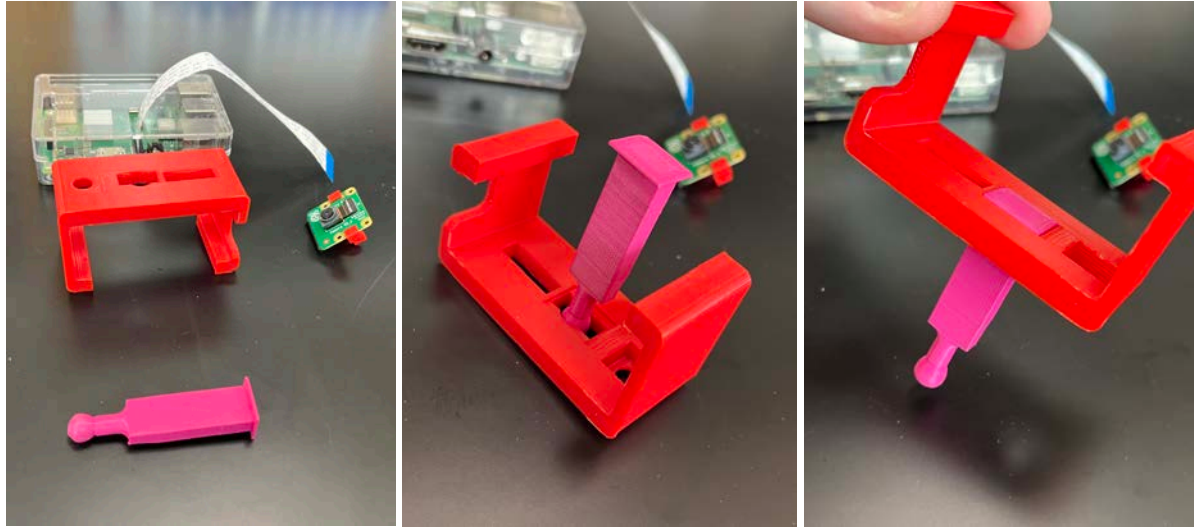
Next, detach the ribbon cable from the PiCamera and insert one end into the Raspberry Pi's camera port.



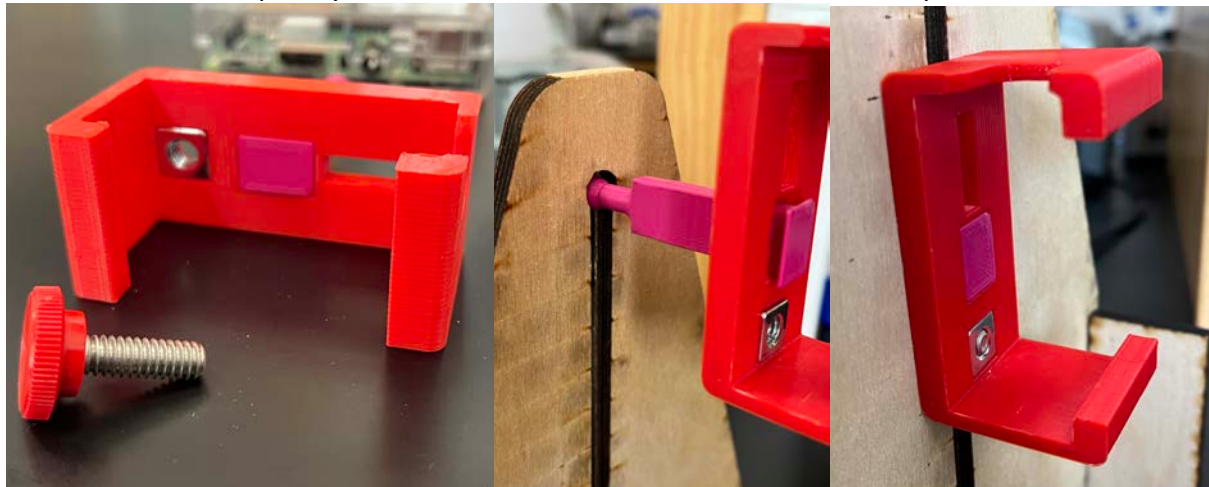
Then insert the PiCamera into the PiCamera mount (be sure not to block the ribbon cable connector).



Insert the rounded end of the camera extension through the middle opening of the RPi mount until it is flush with the inner surface.



Next, insert the M6 square nut into the RPi mount. Then push the camera extension through the top hole of the wooden mount. Use the M6 thumbscrew to attach the RPi mount to the stand, but do not fully tighten it. The end of the thumbscrew should be flush with the square nut to allow the Raspberry Pi to slide into the RPi mount in the next step.

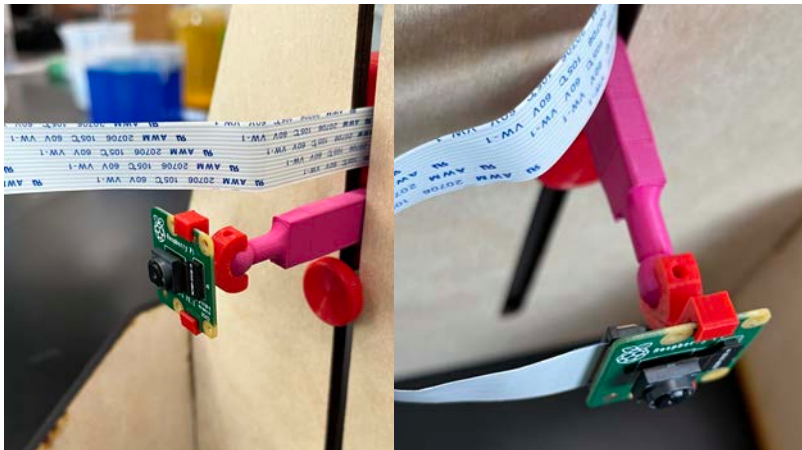




First insert the camera ribbon cable through the top rectangular opening in the RPi mount. Then slide the Raspberry Pi and case into the RPi mount. *(Note: If your specific case does not fit inside RPi mount, you can adjust the scale of that specific axis before 3D printing.)*



Attach the PiCamera and mount to the “ball” end of the camera extension and connect the ribbon cable.



Connect the power supply, HDMI cable, keyboard, and mouse. The microSD card will be used in the next section to load the PiSpy disk image and will be inserted as shown below:



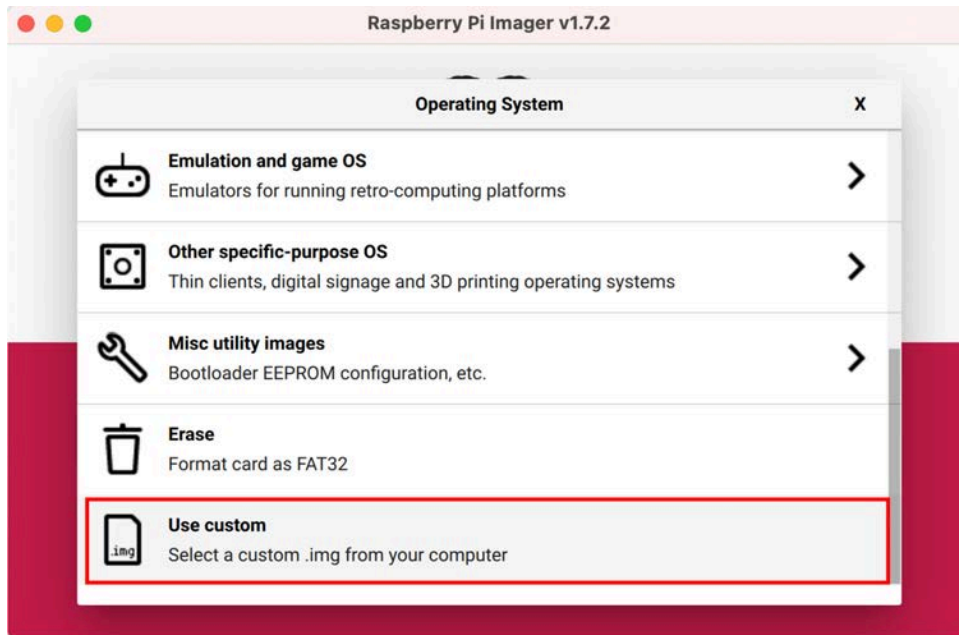
## Setting up the Raspberry Pi Computer

## Installing Raspberry Pi OS

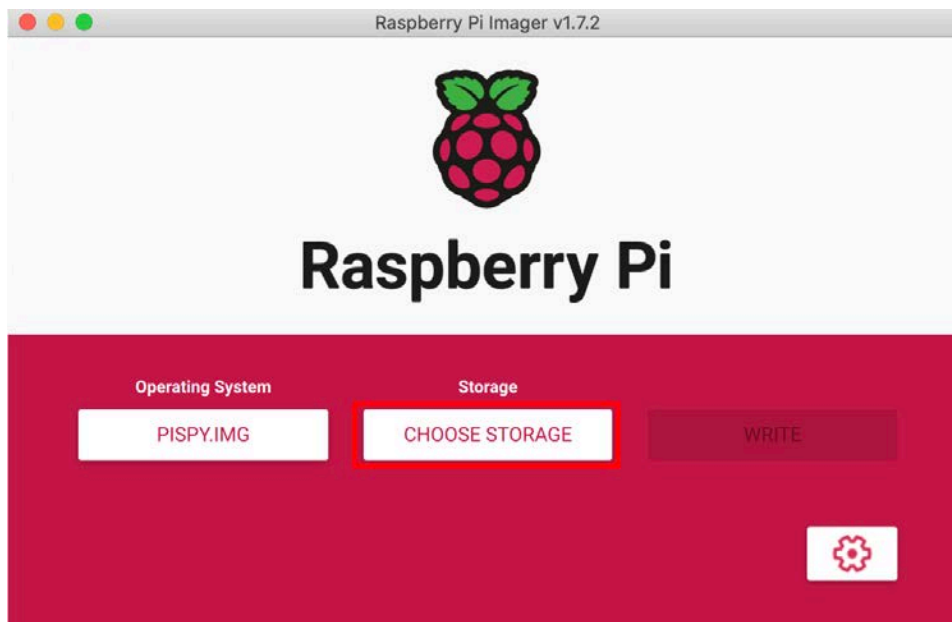
The first step to running PiSpy is to set up the Raspberry Pi OS operating system on the computer. To do so, on a computer with micro SD card reader, download the Raspberry Pi imager (<https://www.raspberrypi.com/software/>) and the PiSpy disk image from the GitHub repository (<https://github.com/gpask/PiSpy/blob/master/Software/PiSpy%20disk%20image.md>). Insert the micro SD card into the computer and open the imager.



Select “CHOOSE OS,”

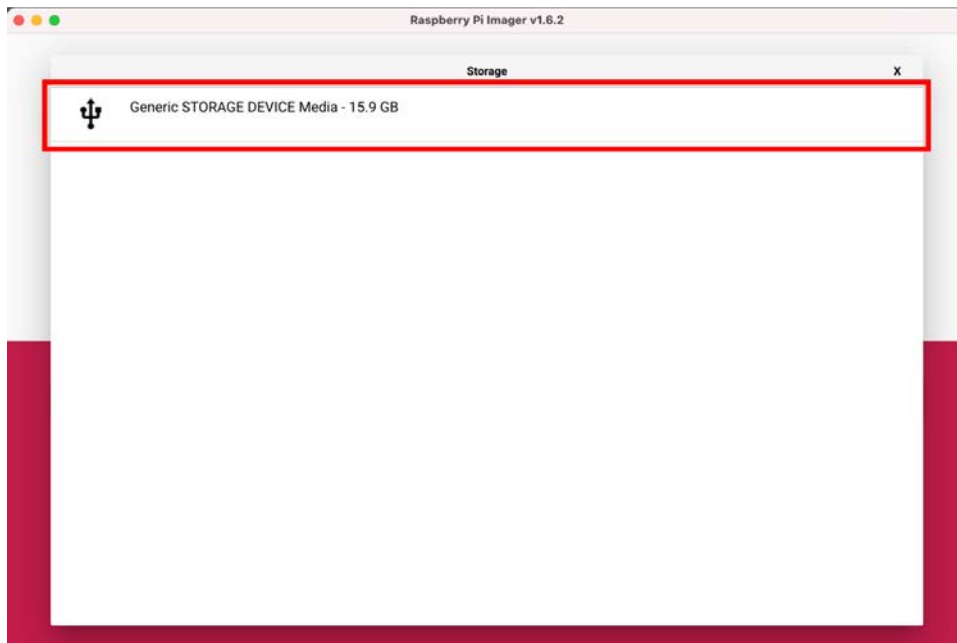


then select “Use custom,” and then select the “PiSpy.img” file you just downloaded.



Next, select “CHOOSE STORAGE”

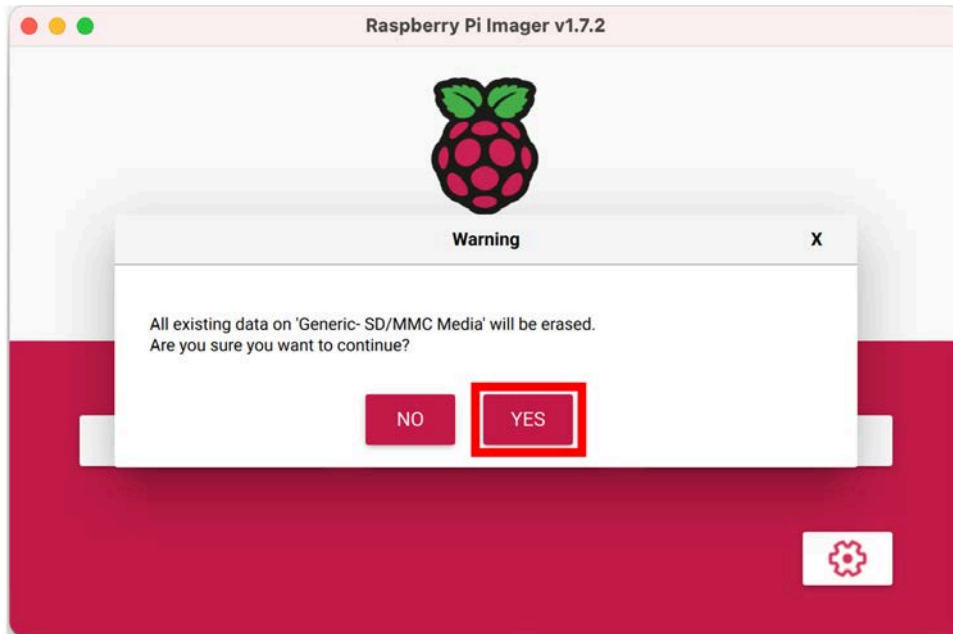




and select the micro SD card to be used for your PiSpy.

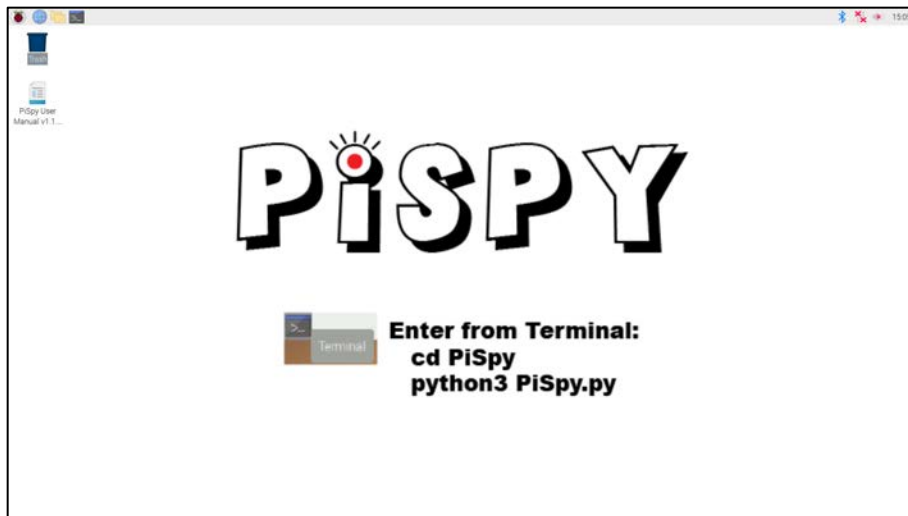


Finally, write the OS to the SD card by selecting “WRITE”

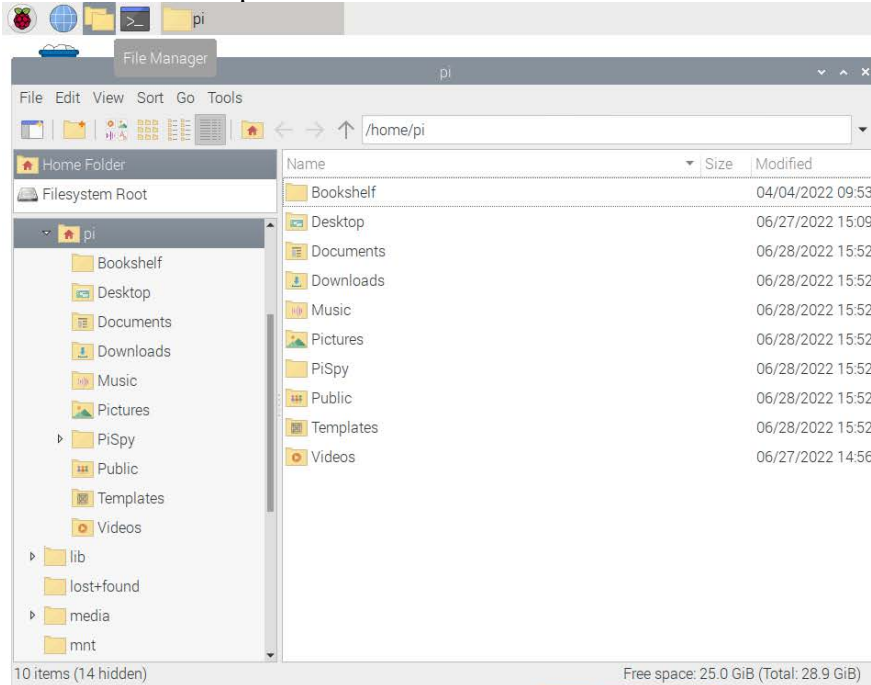


and confirming that this will erase any previous data on the SD card.

When complete, insert the micro SD card into the Raspberry Pi (as shown on page 7) and power it on. Once booted, your desktop should look like this:

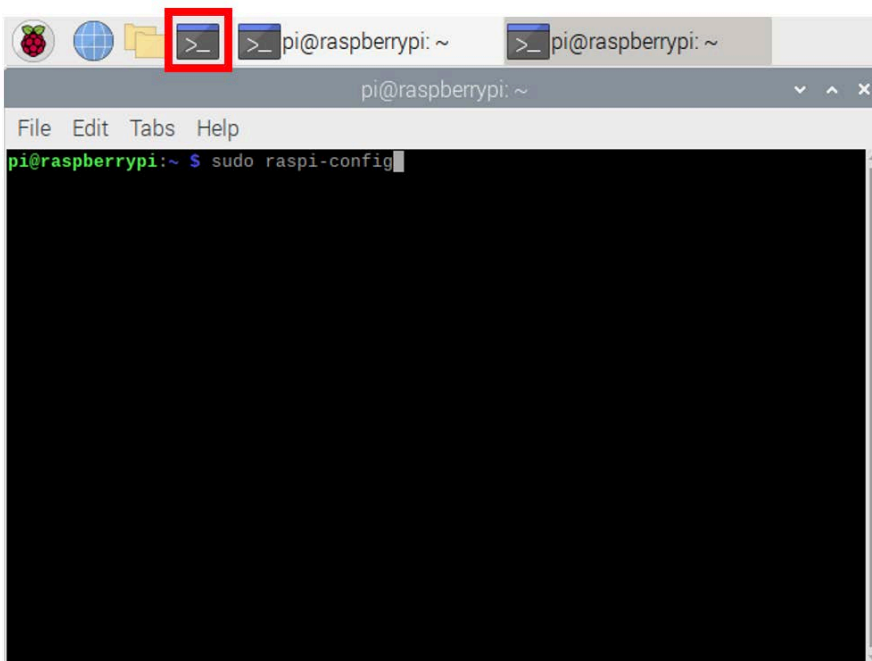


Navigate to the File Manager in the top left corner. You'll notice the folder containing the PiSpy code in the /home/pi folder.

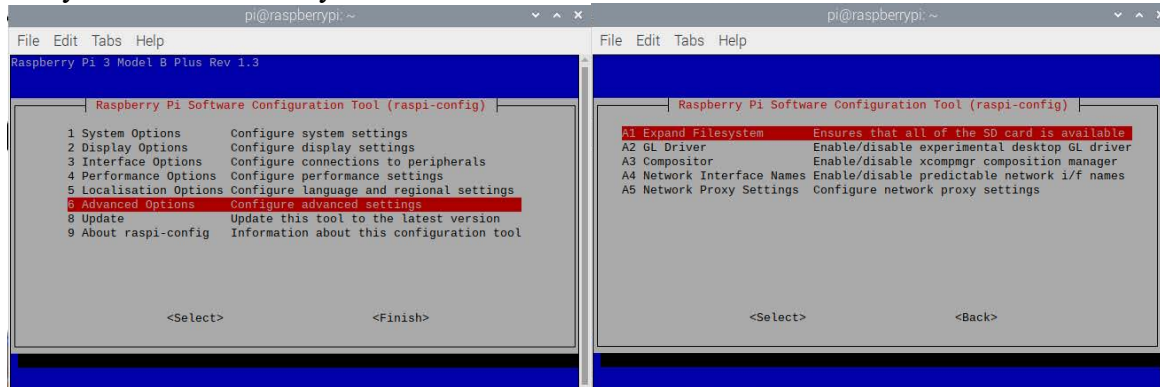


Next, open up Terminal in the top left corner and enter:

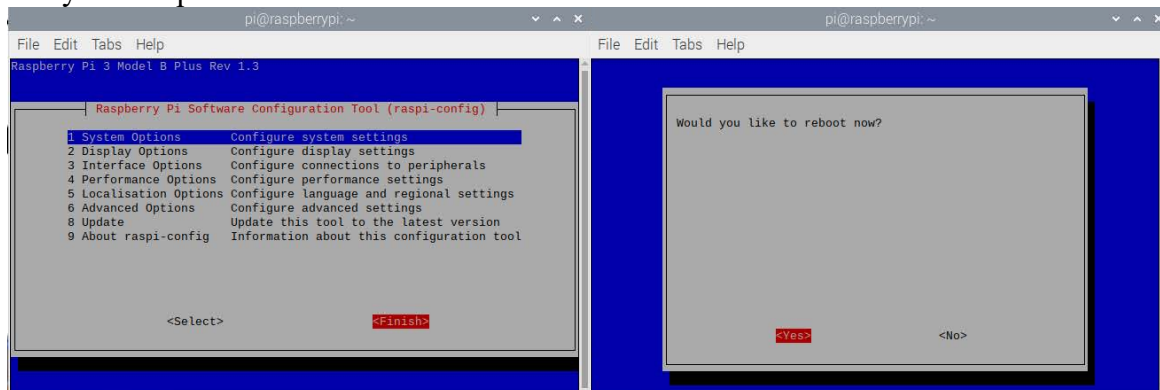
```
sudo raspi-config
```



This will open the following menu. Navigate toward “6 Advanced Options” and press the enter key. Then select “A1 Expand Filesystem” and press enter again. This will expand the accessible filesystem to the size of your micro SD card.



Navigate to “Finish” by pressing the right arrow button, and then select yes to reboot and start filesystem expansion.

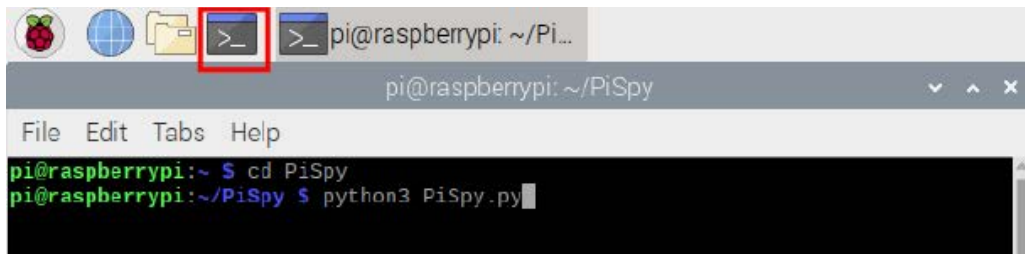


## Operating the graphical user interface

To access the GUI, open the command line by clicking on the terminal icon on the top bar of the screen. Then, enter the following commands, first to move to the PiSpy folder and then to open the PiSpy Control window, which is where the PiSpy is operated from:

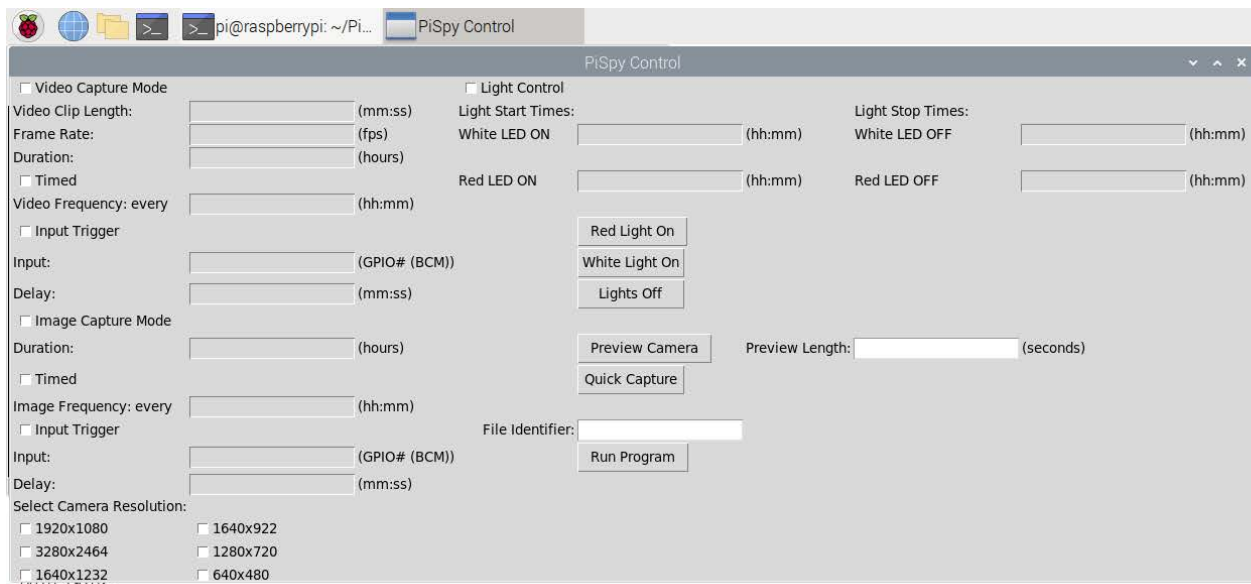
```
cd PiSpy
python3 PiSpy.py
```

If you have already entered those commands, you can use the up arrow to access previously used commands in terminal. Also note that the `cd PiSpy` will only work if the folder containing the PiSpy code is still named “PiSpy.” If the name is changed for some reason (as can happen when unzipping the file), either change the name back to PiSpy or enter the new name for the folder when typing that command.



To help remember how to access the GUI, we have included on the Github repository a suggested background for the PiSpy (“PiSpy desktop.png”) and included it in the disk image.

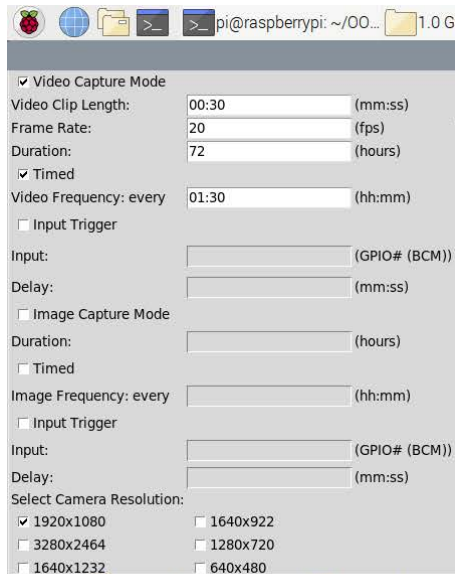
Once the GUI is opened, choose your desired capture mode (image or video) and frequency (timed or triggered) by checking the corresponding boxes. Only one mode and frequency combination (i.e. timed videos) should be selected at a time. Additionally, select the camera resolution by checking the box of the desired resolution, and, for video recording, specify the frame rate. If no resolution is selected, the PiSpy will by default record with a resolution of 1920 x 1080. This link (<https://picamera.readthedocs.io/en/release-1.12/fov.html#camera-modes>) lists the possible resolutions and framerates for the V1 and V2 PiCameras. Depending on the videos being recorded and the quality of the microSD card used, selecting frame rate/resolution combinations at the upper limits of the accepted range can result in skipped frames. If this occurs, lower either the resolution or the frame rate.



### *Timed recording:*

Timed recording can be used when the user wants images or videos to be recorded at fixed intervals. The image or video frequency should be input in the hh:mm format (using a 24 hour clock), and specifies how often images/videos are recorded. For instance, in the below image, 30 second videos with a frame rate of 20 fps and a resolution of 1920x1080 will be recorded every 1.5 hours for 3 days. Note that the frequency must be less than 24 hours.



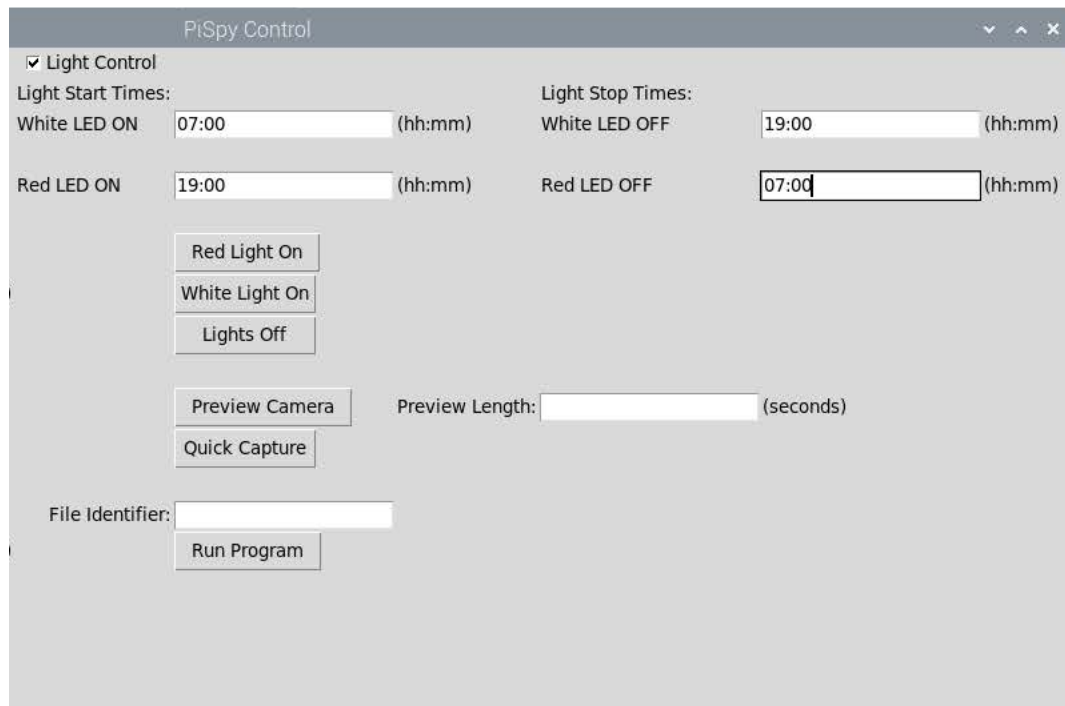


### *Input trigger*

Input trigger mode can be used when the user wants an external source such as a motion sensor or force plate to determine when images or videos are recorded. See **Connecting GPIO sensors for input trigger** for more information on how to use different sensors for input trigger mode.

### *Light Control*

If light control is desired, check the corresponding box and enter the times when each light should be turned on/off (hh:mm format, using a 24-hour clock). Using the red and/or white light buttons, turn on the light(s) you want on when the program starts running. In the image below, the white lights will be on from 7 AM to 7 PM, and the red lights will be on from 7 PM to 7 AM.



*Note on Red Lights at Night*

The default settings programmed into the PiSpy are what we found in testing to give the best quality night images/recordings. However, because of the PiCamera's auto white balance settings, if the subject isn't getting enough light or is sending back too much glare towards the camera, images/videos recorded at night can appear washed out. In addition to making changes to the PiSpy settings (see **Adjusting Default PiSpy Settings** below), changing the location and angle of the lights on the PiSpy, the proximity of the subject to the lens, and the height of the camera can all also help with improving night image quality.,

*Quick Capture*

If the user wants to manually take a picture or video, they can use the quick capture button. Select either video capture mode or image capture mode, and if taking a video also specify the clip length. Then, click the quick capture button to record.

*Preview*

To preview the output of the camera, the user can select the "preview" button. The preview length box allows the user to select how long the preview will last for (while the preview window is open, the user cannot interact with the PiSpy Control window). If no time is selected, the PiSpy will default to a 10 second preview.

*File Identifier*

Optionally, a file identifier can be entered to distinguish between different experiments and/or PiSpy devices. The input in this box will be placed before the default title (local timestamp), but an input here is not required.

*Running the PiSpy Program*

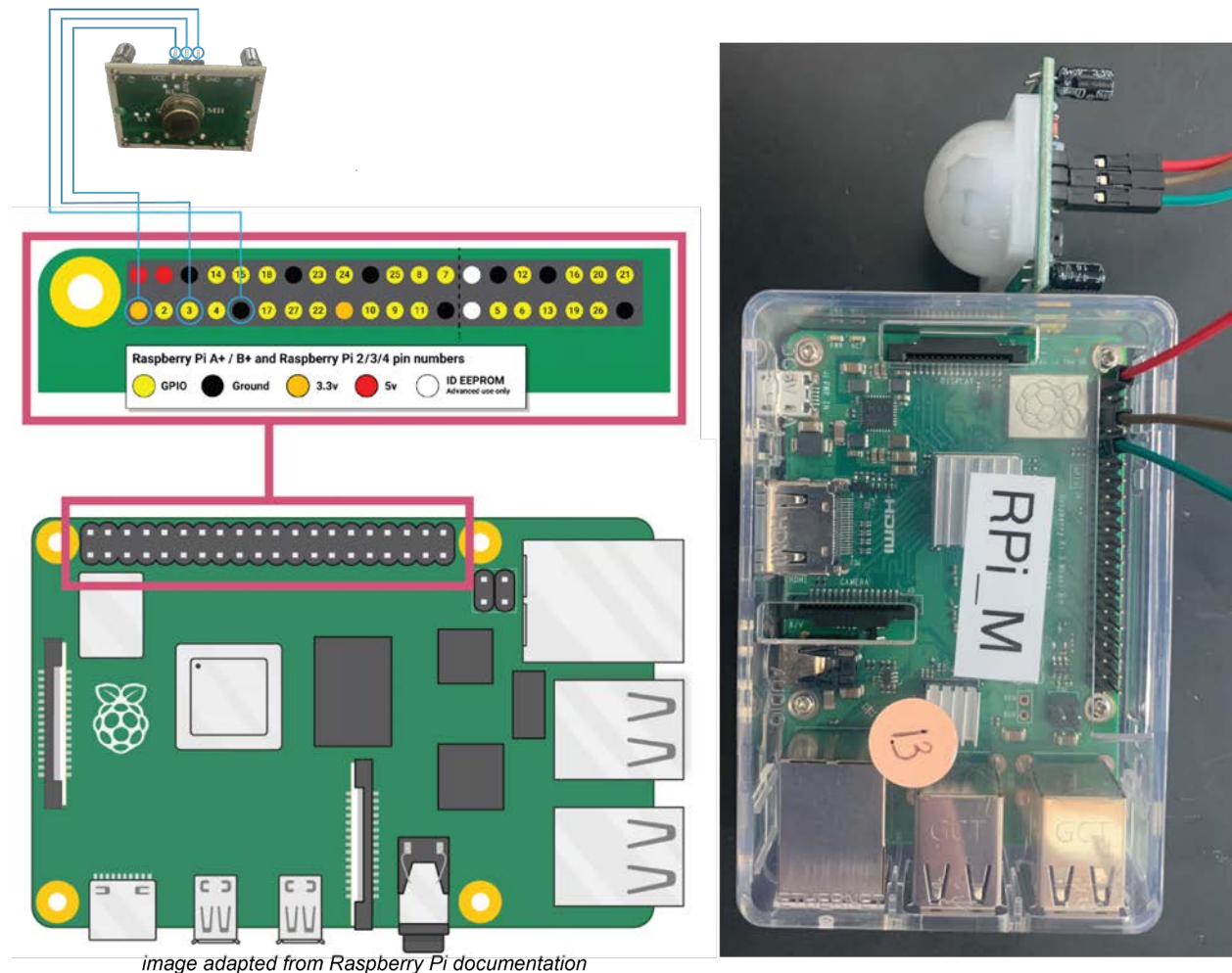
Once all the settings have been chosen, click the apply button to run the program.

Once the settings are applied, the PiSpy control window will close to run the program, and the chosen settings will be printed out to the terminal window. If the program needs to be stopped before the designated end time, perform a keyboard interrupt in terminal (Control + C) or close the terminal window by clicking the X in the top right corner of the window. This will keep any previously recorded images/videos, but stop the program from recording any more, allowing the user to relaunch the control window and select new settings if desired.

## Connecting GPIO sensors for input trigger

### *Connecting a PIR motion sensor*

A PIR motion sensor will have three pins: VCC, GND, and OUT. On some PIR motion sensors, the labels are under the cap; if this is the case, gently remove the cap to be able to identify the pins (the image below is a PIR sensor with the cap removed to show the pin labels). Using the image below as a reference (or by using the “pinout” command in terminal), connect the VCC pin to a 5V or 3.3V power supply pin on the Raspberry Pi, the GND pin to a ground pin on the Pi, and the OUT pin to any of the numbered GPIO pins on the Pi. GPIO board numbering differs between different models of the Raspberry Pi, so if you are not using the Raspberry Pi 3 you will need to consult a different chart when connecting the sensor. The OUT will output a voltage when the sensor is activated, which will be received by the Raspberry Pi and trigger the PiSpy to record an image/video (depending on selected settings). The diagram below is just a suggestion and does not need to be followed exactly; the sensor will work as long as GND is plugged into a ground pin, OUT to a GPIO pin, and VCC to a power supply.



### Connecting an IR break beam

An IR break beam has two components, an emitter and a receiver, but the wiring is similar to a motion sensor. The emitter emits a beam of IR light and has two wires, a red one that receives power from a 5V or 3.3V (the range will be better if using 5V) power supply on the Raspberry Pi, and a black one that connects to a ground pin. The receiver, which receives the signal from the emitter, has the same red and black wires which must be plugged into a different power and ground pin, but also has a white or yellow output wire that should be plugged into one of the general purpose GPIO pins. See the image below for reference. As noted above, the power can come from a 3.3V pin, but the range of the break beam will be smaller. Additionally, any GPIO pin can be used, not just 2 (just make sure to specify which one the input is plugged into when you run the PiSpy)

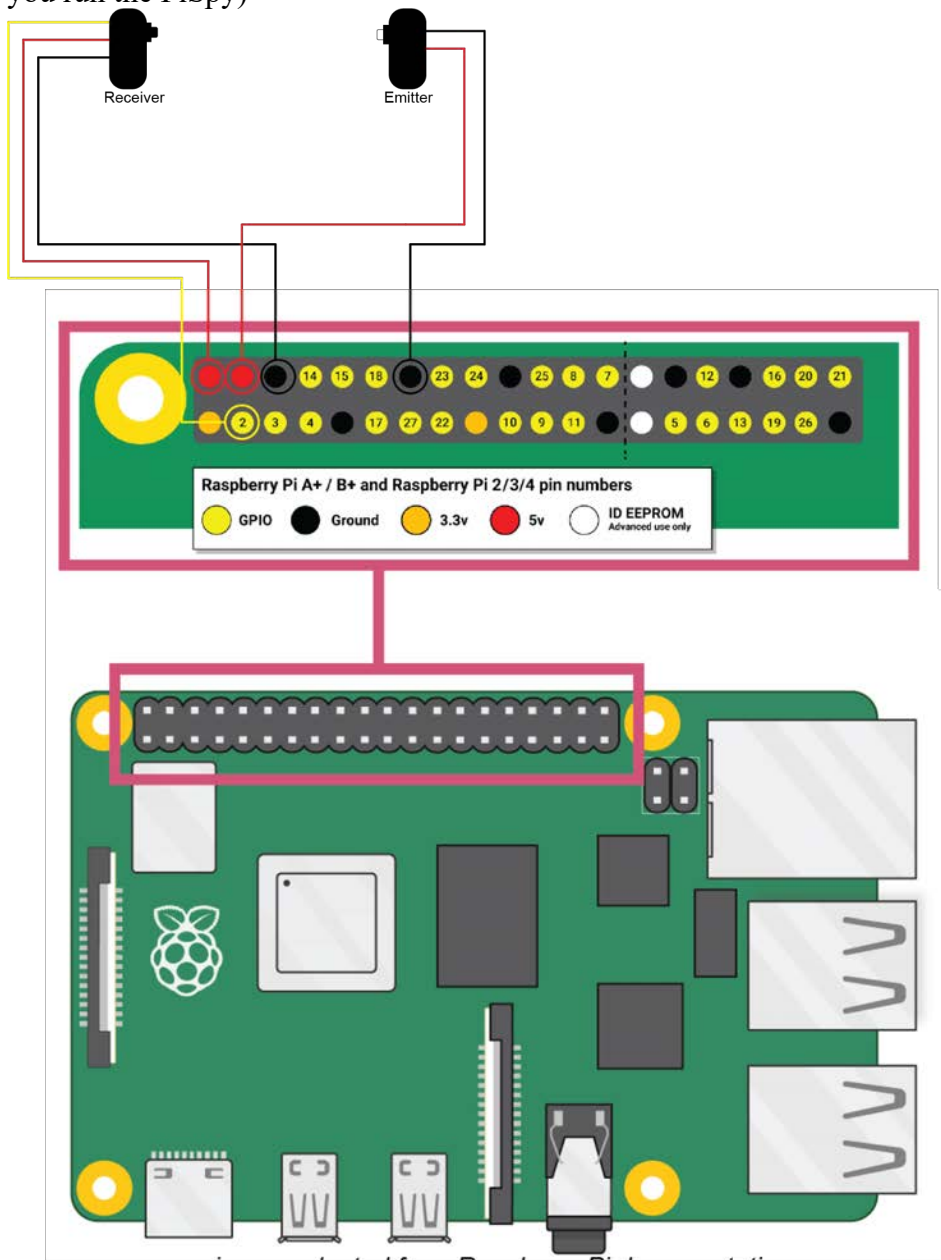


image adapted from Raspberry Pi documentation

*Note on PIR motion sensors*

In our testing, several commercially available and inexpensive PIR motion simply do not function. If the motion sensor does not correctly trigger image/video acquisition, double check that every connection is in the proper position and secure, and if that doesn't work, consider trying a different sensor.

*Note on IR break beams*

An IR break beam has two parts, an emitter that sends out IR light and a sensor that receives that signal and will signal to the computer when that beam is broken. In our testing, small objects (we were using ants) don't always reliably break the beam and trigger the sensor. To make the sensor sensitive enough to trigger when an ant passed it, we used LEGOs® to decrease the total area that the sensor could receive IR signal from, and placed a single index card in front of the sensor as well. The index card is partially transparent to IR light, so when nothing else was between the sensor and the emitter the beam was not broken, but when an ant passed it would trigger video acquisition. Our suggestion is to test break beam sensitivity with your desired subject before running any experiment to confirm that the beam will trigger, and if it does not increase the sensitivity, which can be done with DIY tools and further testing as we did.

*Note on BOARD vs BCM GPIO labeling*

There are two different ways Raspberry Pi GPIO pins can be referred to. BOARD labeling refers to the actual position of the pin on the GPIO board, whereas BCM (Broadcom chip-specific pin numbers) refers to the lower-level functions of the pins. The PiSpy uses the BCM labelling throughout its code, so this is how the input source must be referred to when using input trigger mode.

*Note on code for input triggers*

Different input triggers function in slightly different ways, and the code (in the "Input\_Trigger.py" file) must be slightly different for each of them. As currently deposited on GitHub, the code is written to work with an IR break beam, which has an output of 1 when the beam is not broken and 0 when the beam is broken. As advised in the Input\_Trigger.py code, if using a PIR motion sensor, which outputs a 0 when it does not sense motion and a 1 when it does, change lines 56 and 113 of the Input\_Trigger.py document from "if x == 0:" to "if i >= 1:" and comment out (place # signs before) lines 50-54 and 108-112. See the screenshots below for reference.

While we do not provide code/instructions for wiring of other types of input triggers besides PIR motion sensors and IR break beams, there are a considerable number of online resources available to help with any other types of sensor, many of which can be found directly on websites like Adafruit and raspberrypi.com when purchasing the sensors.



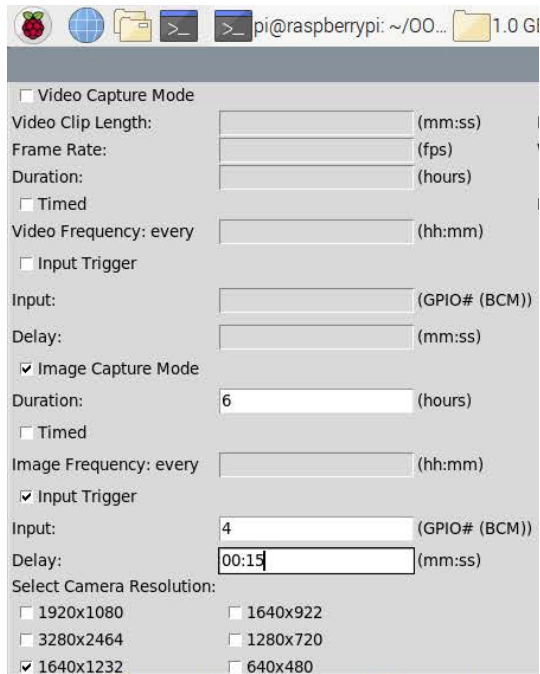
```

42 while days >= 0:
43     if datetime.now().strftime('%H%M') == end_time: #if reaches end time but there are still 1 or more days remaining, decrease days by 1
44         days = days - 1
45         self.time.delay(60, ROnTime, WOnTime, ROFFTime, WOFFTime) #delays for 1 minute to prevent loop from running again
46     i = GPIO.input(trigger)
47     lights.light_on(ROnTime, WOnTime, ROFFTime, WOFFTime)
48     x = 0
49     t = 0
50     while j < 750: #remove this while loop if using motion sensor. For a break beam, the output is 0 when the beam is broken and 1 when it is 1
51         #print(i) #this line can be helpful to make sure the break beam is working correctly (if uncommented, output should be repeatedly print
52         x = x + 1
53         j += 1
54         i = GPIO.input(trigger)
55         #print(GPIO.input(trigger))
56         if x==0: #break beam is broken, for motion sensor switch to if i == 1 (some sensors output a 1 when triggered and some output a 0, for other
57             print('capturing image')
58             timestamp = datetime.now().strftime('%Y_%m_%d_%H_%M_%S') #sets variable for current time (to seconds)
59             cam.capture("/home/pi/Pictures/{}.jpg".format(timestamp)) #takes image, saves to specified path with the timestamp as the format
60             cam.close() #close camera
61             self.time.delay(delay, ROnTime, WOnTime, ROFFTime, WOFFTime) #delays specified amount of time, but still checks if lights need to be :
62             cam = PiCamera() #re-opens camera
63             if GPIO.input(14) == 1: # if red light is on, use night settings
64                 lights.camNight(cam, resolution)
65             else: # otherwise, use day setting
66                 lights.camDay(cam, resolution)
67             print('ready to capture again')
68
69 def video_trigger(self, delay, trigger, duration, length, ROnTime, WOnTime, ROFFTime, WOFFTime, resolution, framerate): #manages sensing and acqui
70
71
99 while days >= 0:
100     if datetime.now().strftime('%H%M') == end_time: #if reaches end time but there are still 1 or more days remaining, decrease days by 1
101         days = days - 1
102         self.time.delay(60, ROnTime, WOnTime, ROFFTime, WOFFTime) #delays for 1 minute to prevent loop from running again
103     i = GPIO.input(trigger)
104     lights.light_on(ROnTime, WOnTime, ROFFTime, WOFFTime)
105     x = 0
106     t = 0
107     while j < 750: #remove this while loop if using motion sensor. For a break beam, the output is 0 when the beam is broken and 1 when it is 1
108         #print(i) #this line can be helpful to make sure the break beam is working correctly (if uncommented, output should be repeatedly print
109         x = x + 1
110         j += 1
111         i = GPIO.input(trigger)
112         if x==0: #break beam is broken, for motion sensor switch to if i == 1 (some sensors output a 1 when triggered and some output a 0, for other
113             print('capturing video')
114             timestamp = datetime.now().strftime('%Y_%m_%d_%H_%M_%S') #sets variable for current time (to seconds)
115             name = "/home/pi/Videos/{}.h264".format(timestamp) #begins recording, saves to specified path with the timestamp as the format
116             cam.start_recording(name) #begins recording, saves to specified path with the timestamp as the format
117             cam.wait_recording(length) #checks for exceptions- if error occurs the recording will stop
118             cam.stop_recording() #ends the recording. If there is an error it will raise the exception
119             rate = cam.framerate
120             cam.close()
121             print('done recording')
122             os.system('MP4Box -quiet -add {} ({name}).h264:fps={rate} //({name}).mp4'.format(name = name, rate = rate)) #converts to MP4 using MP4Box.
123             os.remove("/{name}.h264".format(name = name)) #removes h264 file. If GPAC cannot be installed, remove this line
124             self.time.delay(delay, ROnTime, WOnTime, ROFFTime, WOFFTime) #delays specified amount of time, but still checks if lights need to be :
125             cam = PiCamera() #re-opens camera
126             if GPIO.input(14) == 1: # if red light is on, use night settings
127                 lights.cam_night(cam, resolution, framerate)
128

```

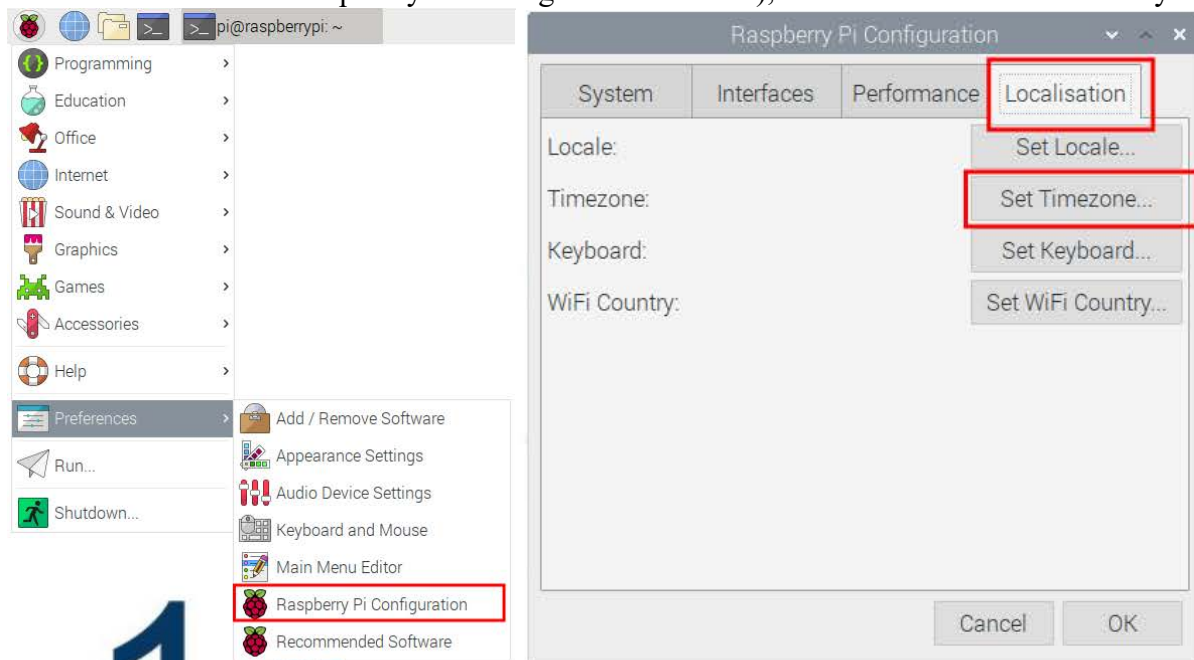
### Using input trigger mode

In the PiSpy control window, specify which GPIO pin the trigger source is connected to, using the BCM format (the white text in the above diagram). Additionally, specify in the mm:ss format how long a delay is desired between image or video captures. Note that many sensors (i.e. PIR motion sensors) also have a mechanically adjusted delay setting, so this can also be adjusted to set the device's delay. The PiSpy will use whichever delay is longer. In the image below, 1640x1232 images will be recorded when triggered by a sensor plugged into GPIO 4 with a 15 second delay. Recording will occur for 6 hours.



### Setting the time on a Raspberry Pi

There are two ways to set the time on a Raspberry Pi. If the computer is connected to the internet and its time zone is set (which is done when the pi is first booted up, or can be done in the localisation tab of the Raspberry Pi Configuration window), the time will set automatically.



To set the time manually, open a terminal window and type the command (replacing the specifics with whatever the current date/time is):

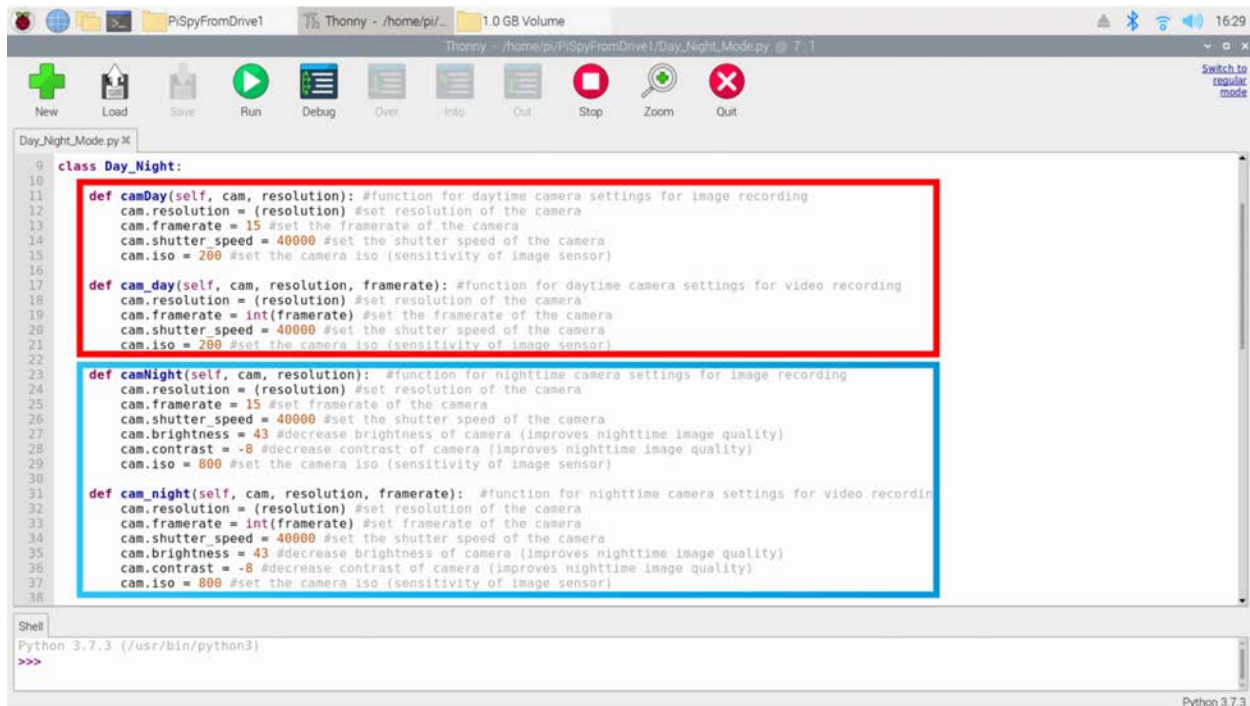
```
sudo date -s "Tue Jun 15 22:10:11 UTC 2021"
```

### Optional: connecting a Real-Time Clock (RTC) to the PiSpy

If, for connectivity, security, or other reasons it will not be possible to connect your PiSpy to the internet, a physical RTC can be connected to the GPIO pins to maintain accurate timing on the device. There are several different RTC options available that we have tested and confirmed to work with the PiSpy, but note that some of these use the same GPIO pins that other optional PiSpy add-ons (such as lights or motion sensors) require. Adafruit has a thorough and easy-to-follow tutorial for connecting different RTCs to a Raspberry Pi, and is also a good source to purchase these devices if they are required for your project: <https://learn.adafruit.com/adding-a-real-time-clock-to-raspberry-pi>.

### Adjusting Default PiSpy Settings

By default, the PiSpy camera will have an ISO of 200 during day recording (no lights on or white lights on), and an ISO of 800 during night recording (red lights on). At night, brightness (set to 43 on a scale from 1 to 100) and contrast (set to -8 on a scale from -100 to 100) are also lowered, as this helped improve image quality with the red LED lights on. If the user wants to adjust any of these settings, open the `Day_Night_Mode.py` file in the PiSpy folder. Then, make the desired adjustments under the `camDay` function (to change day settings) or the `camNight` function (to change night settings). Note that there are two `camDay` and two `camNight` functions (one for when framerate is specified and one for when it is not). To ensure a change in settings will always be implemented, make sure to adjust settings in both of these functions. In the screenshot below, the functions highlighted in red control day settings, and the ones in blue control night settings. Note, however, that image quality, especially at night, is also effected by the physical setup of the Pi. See *Note on Red Lights at Night* above for slightly more detail on this.



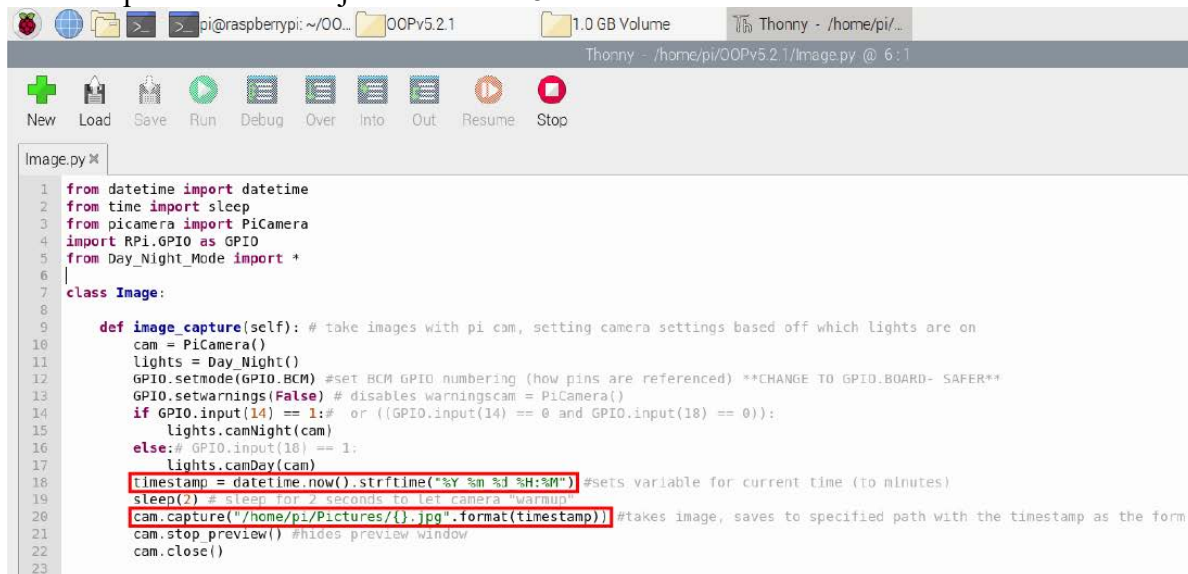
```

9 class Day_Night:
10
11     def camDay(self, cam, resolution): #function for daytime camera settings for image recording.
12         cam.resolution = (resolution) #set resolution of the camera
13         cam.framerate = 15 #set the framerate of the camera
14         cam.shutter_speed = 40000 #set the shutter speed of the camera
15         cam.iso = 200 #set the camera iso (sensitivity of image sensor)
16
17     def cam_day(self, cam, resolution, framerate): #function for daytime camera settings for video recording
18         cam.resolution = (resolution) #set resolution of the camera
19         cam.framerate = int(framerate) #set the framerate of the camera
20         cam.shutter_speed = 40000 #set the shutter speed of the camera
21         cam.iso = 200 #set the camera iso (sensitivity of image sensor)
22
23     def camNight(self, cam, resolution): #function for nighttime camera settings for image recording
24         cam.resolution = (resolution) #set resolution of the camera
25         cam.framerate = 15 #set framerate of the camera
26         cam.shutter_speed = 40000 #set the shutter speed of the camera
27         cam.brightness = 43 #decrease brightness of camera (improves nighttime image quality)
28         cam.contrast = -8 #decrease contrast of camera (improves nighttime image quality)
29         cam.iso = 800 #set the camera iso (sensitivity of image sensor)
30
31     def cam_night(self, cam, resolution, framerate): #function for nighttime camera settings for video recording
32         cam.resolution = (resolution) #set resolution of the camera
33         cam.framerate = int(framerate) #set framerate of the camera
34         cam.shutter_speed = 40000 #set the shutter speed of the camera
35         cam.brightness = 43 #decrease brightness of camera (improves nighttime image quality)
36         cam.contrast = -8 #decrease contrast of camera (improves nighttime image quality)
37         cam.iso = 800 #set the camera iso (sensitivity of image sensor)
38

```

By default, the PiSpy will title images and videos with the time they were recorded (in the format year\_month\_day\_hour:minute), and will store them to the default Pictures folder (/home/pi/Pictures).

To change either of these settings for image recording, open the “Image.py” file. To change the location the images are stored at, replace /home/pi/Pictures in line 20 with the path to the desired folder (make sure to leave the /{}.jpg after the path, as that is what specifies the file name). To change the file name, add any desired text before or after the {} in line 20 (the {} is a placeholder that tells Python where to put the timestamp in the filename). The format of the timestamp itself can be adjusted in line 18.

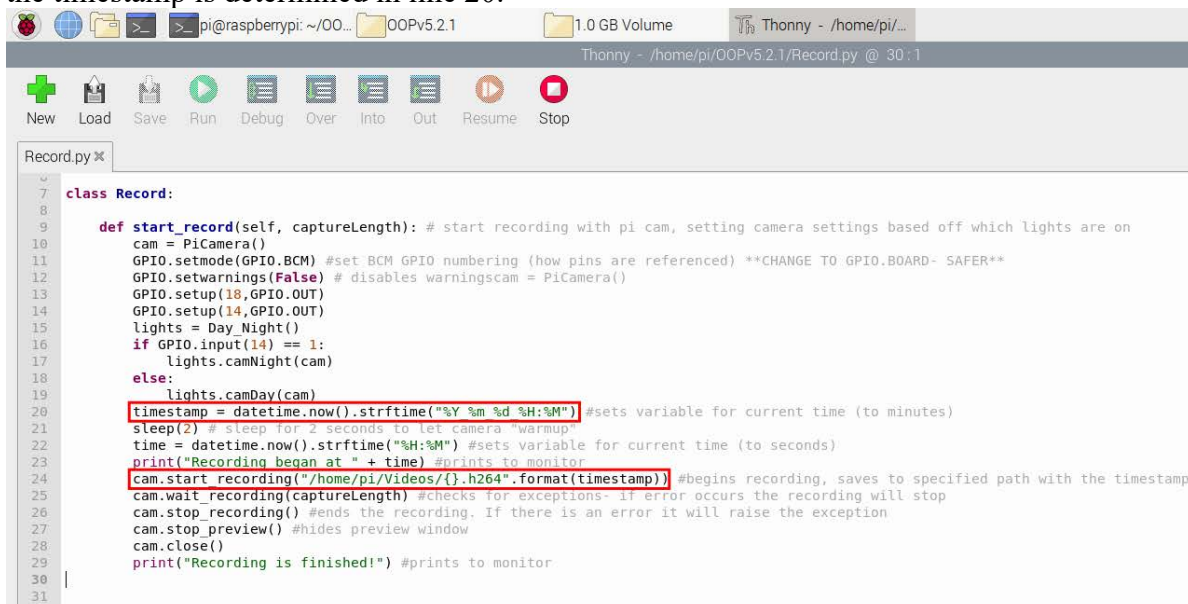


```

1 from datetime import datetime
2 from time import sleep
3 from picamera import PiCamera
4 import RPi.GPIO as GPIO
5 from Day_Night_Mode import *
6
7 class Image:
8
9     def image_capture(self): # take images with pi cam, setting camera settings based off which lights are on
10         cam = PiCamera()
11         lights = Day_Night()
12         GPIO.setmode(GPIO.BCM) #set BCM GPIO numbering (how pins are referenced) **CHANGE TO GPIO.BOARD- SAFER**
13         GPIO.setwarnings(False) # disables warningscam = PiCamera()
14         if GPIO.input(14) == 1: # or ((GPIO.input(14) == 0 and GPIO.input(18) == 0)):
15             lights.camNight(cam)
16         else: # GPIO.input(18) == 1:
17             lights.camDay(cam)
18         timestamp = datetime.now().strftime("%Y %m %d %H:%M") #sets variable for current time (to minutes)
19         sleep(2) # sleep for 2 seconds to let camera "warmup"
20         cam.capture("/home/pi/Pictures/{}.jpg".format(timestamp)) #takes image, saves to specified path with the timestamp as the form
21         cam.stop_preview() #hides preview window
22         cam.close()
23

```

For video recording, open the “Record.py” file. The same steps as above can be used to adjust the file’s name and location, except the location/name is determined in line 24, and the format of the timestamp is determined in line 20.



```

7 class Record:
8
9     def start_record(self, captureLength): # start recording with pi cam, setting camera settings based off which lights are on
10         cam = PiCamera()
11         GPIO.setmode(GPIO.BCM) #set BCM GPIO numbering (how pins are referenced) **CHANGE TO GPIO.BOARD- SAFER**
12         GPIO.setwarnings(False) # disables warningscam = PiCamera()
13         GPIO.setup(18,GPIO.OUT)
14         GPIO.setup(14,GPIO.OUT)
15         lights = Day_Night()
16         if GPIO.input(14) == 1:
17             lights.camNight(cam)
18         else:
19             lights.camDay(cam)
20         timestamp = datetime.now().strftime("%Y %m %d %H:%M") #sets variable for current time (to minutes)
21         sleep(2) # sleep for 2 seconds to let camera "warmup"
22         time = datetime.now().strftime("%H:%M") #sets variable for current time (to seconds)
23         print("Recording began at " + time) #prints to monitor
24         cam.start_recording("/home/pi/Videos/{}.h264".format(timestamp)) #begins recording, saves to specified path with the timestamp
25         cam.wait_recording(captureLength) #checks for exceptions- if error occurs the recording will stop
26         cam.stop_recording() #ends the recording. If there is an error it will raise the exception
27         cam.stop_preview() #hides preview window
28         cam.close()
29         print("Recording is finished!") #prints to monitor
30
31

```



By default, PiSpy uses the a bitrate of 6000000 (6 mb per second), as we found in our testing that this provided sufficient video quality while keeping file sizes small and minimizing frame skipping in the videos. To change this to a different value, update the value specified in line 44 of "Record.py."

```

Day_Night_Mode.py  Image.py  Import_Trigger.py  PiSpy.py  Record.py ×  Time_Lists.py
25  class Record:
26
27      def start_record(self, ID, captureLength, resolution, framerate): #captures video, using settings based off which lights are on
28          cam = PiCamera()
29          GPIO.setmode(GPIO.BCM) #set BCM GPIO numbering (how pins are referenced)
30          GPIO.setwarnings(False) # disables warnings
31          GPIO.setup(18,GPIO.OUT) #tells computer that GPIO pins used for red/white lights are outputs
32          GPIO.setup(14,GPIO.OUT) #tells computer that GPIO pins used for red/white lights are outputs
33          lights = Day_Night() #initiates lights class
34          if GPIO.input(14) == 1: #if red lights are on, use night settings
35              lights.cam_night(cam, resolution, framerate)
36          else: #otherwise, use day settings
37              lights.cam_day(cam, resolution, framerate)
38          timestamp = datetime.now().strftime("%Y_%m_%d_%H_%M_%S") #sets variable for current time (to minutes)
39          timestamp = ID + timestamp
40          sleep(2) # sleep for 2 seconds to let camera "warmup"
41          time = datetime.now().strftime("%H:%M:%S") #sets variable for current time (to seconds)
42          print("Recording began at " + time) #prints to monitor
43          name = "/home/pi/Videos/{}".format(timestamp)
44          cam.start_recording(name + ".h264", bitrate = 6000000) #begins recording, saves to specified path with the timestamp as the format
45          cam.wait_recording(captureLength) #checks for exceptions- if error occurs the recording will stop, otherwise records for specified
46          cam.stop_recording() #ends the recording. If there is an error it will raise the exception
47          time = datetime.now().strftime("%H:%M:%S") #sets variable for current time (to seconds)
48          cam.stop_preview() #hides preview window
49          rate = cam.framerate
50          cam.close()
51          os.system("MP4Box -quiet -add //{name}.h264:fps={rate} //{name}.mp4".format(name = name, rate = rate)) #converts to MP4 using MP4

```



## LED PCB Lighting

Designs for white/red LED Printed Circuit Boards (PCBs) are on the PiSpy GitHub repository and can be used to purchase PCBs from fabrication businesses. One or more LED PCBs must be connected to the GPIO board on the Raspberry Pi for lighting control. When the Pi is off, connect the GPIO pins to the corresponding pins on the LED PCB(s) as shown in the diagram below. For GPIO positions, please refer to the pinout diagram above. Alternatively, the pinout command in terminal will display a diagram of the GPIO for your Raspberry Pi model.

If using other lighting outputs, please note the GPIO pins of interest and that the GUI code may need to be modified.

