# Motion Activated Scare System – MASS 2021

Andreas Seiler, SLU

## Aim

The MASS system is a programmable unit that, when receiving a trigger from a motion sensor and/or a train detector, activates external autocameras and then displays selected signals (acoustic and/or visual) to deter wildlife.

The system is designed for experimental purposes only. There are thus no requirements on robustness of the technical components. Life expectance is 2-3 field seasons only.

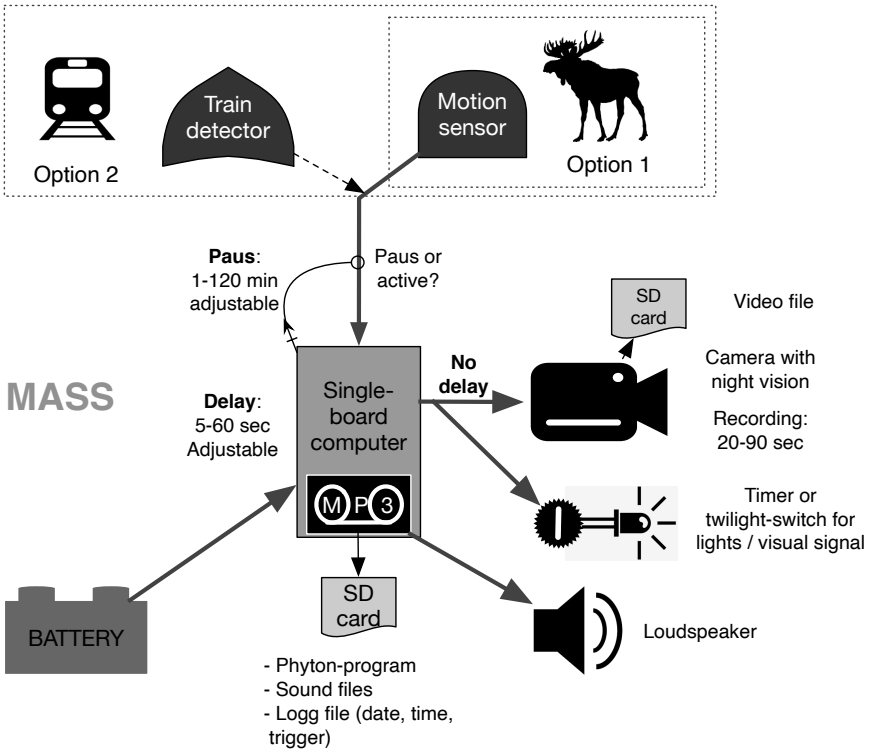



Figure 1. Sketch of MASS unit

# System

The MASS consists of (fig 1):

- Input 1: external motion sensor (Option 1 for use outside rail environment)
- Input 2: external motion sensor and train detector (Option 2: both sensors must be triggered to activate the system)
- Single-card computer (for example Raspberry Pi)
    - USB connections must be easily accessible (for SD card logs and sounds)
    - Internal clock; programmable
- Output 1 to external camera (alternative: camera is integrated)
- Output 2 to external loudspeaker
- Output 3 to external light unit
- Battery 12 V (external)

The Single-card computer with sound card, maybe camera and loudspeaker can be combined in a waterproof box; motion sensor, maybe camera, visual unit, and battery should be externally connected by cable.

# Components

## INPUT

### PIR sensor

- Two-way sensor that combines thermal detection with radar (e.g. Bosch OD850-F1E Series TriTech outdoor detectors).

### Train detector

- If used in railway environment, the system should only be triggered shortly before a train arrives and only if animals are present.
- Train detectors (https://schweizer-electronic.com/en/warning-systems/products/inductive-treadle-rsr123) are used in warning systems such as Minimel Lynx LOWS - Manually triggered radio warning system by Schweizer Electronics AG https://schweizer-electronic.com/en/warning-systems/temporary-warning-systems/minimel-lynx-lows-manually-triggered-radio-warning-system

## COMPUTER

- Raspberry Pi zero, 3 or 4; see e.g., link to ELFA.se.
- DC Converter Module 12V to 5V 3A 15W (ex from LightinBox)
- Realtime serial clock Raspberry Pi RTC DS1307 (ex from ELFA )
- Maybe: additional sound card RASP RASPYPLAY4 (see MP3-player)
- Cables:
    - from battery;
    - from motion sensor;
    - to camera;
    - to visual unit (light trigger)
    - to loudspeaker;
    - to external USD connector for SD card
-

- When input from sensors detected, the system activates camera and initiates the display of selected sound files according to the settings stored in a text-file (see below)
- For each activation a log entry is written, log file is on an external SD card.

## OUTPUT

### Camera

- Reconyx Hyperfire HPX2 wildlife camera with option for external trigger and external battery supply (https://www.reconyx.com/product/hyperfire-2-Professional-covert-ir-camera); connected with a 2 m cable to MASS unit.
- Alternative: any camera that allows for night vision with covert IR and that can be activated by the MASS sensor ; could also be integrated in the main board

### Loudspeaker

- Must have good (natural) range in frequency
- Max volume 100 dB (no specific requirements)
- Must be waterproof for outdoor use
- Horn speaker: https://se.rs-online.com/web/p/horn-speakers/8091120; 1700 SEK eller Speaker used by NTNU https://www.conrad.se/p/monacor-esp-215ws-ela-takhoegtalare-vit-1-st-1331663?gclid=Cj0KCQiAvbiBBhD-ARIsAGM48byvplpddTrYyEwkFF6Ai0uARqaFWFvBSuBd2lSQiV-6ZyjXUcC3sUQaAuKhEALw_wcB&gclsrc=aw.ds&utm_campaign=shopping-feed&utm_content=free-google-shopping-clicks&utm_medium=surfaces&utm_source=google&utm_term=1331663&vat=true

### Trigger for visual unit

- Designed in Norway. Triggerd by signal to camera!
- A simple trigger may suffice to switch on a street-lamp like illumination (simple LED lamp)
- Duration of illumination must be adjustable (20 -60 sec), lights are only active during night (twilight switch)
- Illumination: a simple 12V LED light suffices (https://www.biltema.se/bil---mc/bilbelysning/arbetsbelysning/)
- Powered by external 12V battery (similar to car battery with e.g. 120Ah) that can operate outside in winter

### *MP3-player*

- *integrated in the main board or as additional sound card*
- *starts with adjustable delay (5-120 sec) after that the PIR sensor (or and train detector) activated the main board*
- *programmable so that only specific files / folders are played at a given time:*
- *Selects sound files from folders on SD card (one folder per hour of day)*
- *MP3-player can also trigger the visual unit with the same delays as the sound is played (i.e. acoustic and visual signal are exchangeable…)*

# Operation logic

Option 1: If only motion is detected

- Test if system is still in pause-mode after previous trigger

- If not: continue below …

Option 2: If motion and train is detected

- Activate camera
    - Wait for specified delay, before
    - activating MP3-player and display selected sound-file
        - Sound file switches on the visual-unit (if so chosen)
            - Lights are turned on if timer or twilight switch allows
        - After sound file is completed, lights are switched off

**Paus in Option 1:** After sound file is played (and the lights have been turned off), the system is put in pause-mode for a defined time (1-180 min). New motion is ignored.

No paus is used in Option2.

**Sound files** are played according to a pre-defined order given by a system of hourly folders containing sound files (fig 2). It is through the order of folders and sound files that we control the experimental setup. This allows us to display different audible as well as silent sounds at different times of the day (dawn, dusk, night …).

**Lights** are switched on if light conditions are low (twilight switch) or the timer is activated AND a sound file is played (audible or silent). Duration of illumination is set by the length of the sound file; lights switch off when the sound is ended.

- To expose animals to light-only signals, all sound files must be silent.
- To allow the light to be turned on before a sound is played, sound files need to begin with a period of silence.
- To display only sounds, the light-trigger must be switched off (twilight trigger or timer).

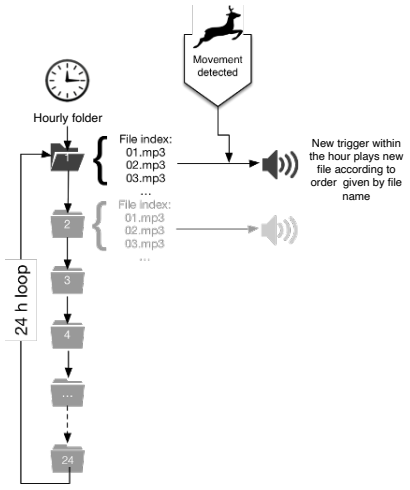



Figure 2. System of 12 hourly folders containing a series of sound files.

# MASS 3 Configuration

## 1. Set up a new Raspberry Pi 3B

Set up the original Pi and connect to internet.

Activate Camera and VNC in system preference settings for later remote access

Connect a camera and (any) speaker

Test sound by playing any sound file.

Test camera function by writing in the terminal:

```
raspivid -o Desktop/video.h264
```

Shut down and connect JustBoom HAT only (without the RTC clock)

## 2. Install JustBoom Amp HAT

See guide: https://www.justboom.co/software/configure-justboom-with-raspbian/

In Terminal, run the following command

```
sudo nano /boot/config.txt
```

At the bottom of the opened file identify the line

```
dtparam=audio=on
```

Comment the line out by adding a # in front of it and add the JustBoom configuration so that the configuration shows as follows:

```
#dtparam=audio=on
dtparam=audio=off
#dtoverlay=i2s-mmap
dtoverlay=justboom-dac
```

The line ”dtoverlay=i2s-mmap” is used by the Jack server so that the JustBoom cards can also work with Sonic Pi. Since the Raspbian version Stretch, the overlay “i2s-mmap” is no longer available nor needed and you should keep this line inactivated. To check for OS version: run in terminal “`cat /etc/os-release”` – likely result: OS ”Buster” (released in 2019 and was replacing the earlier Stretch version).

**Then, reboot Raspberry Pi:**

```
sudo reboot
```

## 3. Install real time clock

Install software before connecting the RT clock.

See guide online: https://www.seeedstudio.com/blog/2020/07/07/raspberry-pi-rtc-tutorial-using-ds1307-and-ds3231-rtcs-with-raspberry-pi-m/.

Type the following command in your Raspberry Pi terminal to clone the pi-hats repository

```
git clone https://github.com/Seeed-Studio/pi-hats.git
```

When the download is finished, type the following command in your terminal.

```
cd pi-hats/tools
```

Install the drivers according to your RTC. If you have a DS1307, type the following

```
sudo ./install.sh -u rtc_ds1307
```

**Power off Raspberry Pi**

```
sudo shutdown -h now
```

**Now connect the RTC clock to JustBoom Hat** and p**ower up the Raspberry Pi**

You can use the following command to check whether the driver is installed successfully in cd pi-hats/tools ...

```
/install.sh -l
```

Set the hardware clock from the current system time

```
sudo hwclock -w
```

Access help for more usage

```
hwclock –help
```

Show actual timed and date:

```
sudo hwclock -r
```

## 4. Create autostart av MASS.py

Create two systemd-services: one that starts Raspbians soundserver PulseAudio and one that waits for the former and then starts MASS.py.

### 4.1 Systemd-service for Pulseaudio

Create file:

```
/etc/systemd/system/pulseaudio.service
```

And open it for editing:

```
sudo nano /etc/systemd/system/pulseaudio.service
```

Copy these lines into the file:

```
[Unit]
Description=PulseAudio system-wide sound server
After=avahi-daemon.service network.target

[Service]
Type=notify
ExecStart=/usr/bin/pulseaudio --daemonize=no --system --disallow-exit --realtime --no-cpu-limit --log-target=journal
ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target
```

**Commented [ASE1]:** not working version from juli 2021:
~~*Från bygganvisningen:*~~
~~*[Unit]*~~
~~*Description=PulseAudio system-wide sound server*~~
~~*After=avahi-daemon.service network.target*~~
~~*[Service]*~~
~~*Type=forking*~~
~~*ExecStart=/usr/bin/pulseaudio --daemon --system --disallow-exit --disallow-module-loading --realtime --no-cpu-limit --log-target=journal*~~
~~*ExecReload=/bin/kill -HUP $MAINPID*~~
~~*[Install]*~~
~~*WantedBy=multi-user.target*~~
*>>>> Attention – this script does not seem to work !<<<<<<*

### 4.2 Systemd-service för MASS

Create file:

```
/etc/systemd/system/MASS.service
```

and open it for editing:

```
sudo nano /etc/systemd/system/MASS.service
```

Copy these lines into the file:

```
[Unit]
Description=MASS service
After=pulseaudio.service

[Service]
Type=simple
User=root
ExecStart=strace python3 /home/pi/MASS/MASS.py

[Install]
WantedBy=default.target
```

**Commented [ASE2]:** fungerar inte:
~~*Från bygganvisningen:*~~
~~*[Unit]*~~
~~*Description=MASS*~~
~~*After=pulseaudio.service*~~
~~*[Service]*~~
~~*Type=idle*~~
~~*ExecStart= /usr/bin/python3 /home/pi/MASS/MASS.py*~~
~~*WorkingDirectory=/home/pi/MASS*~~
~~*User=pi*~~
~~*[Install]*~~
~~*WantedBy=multi-user.target*~~
*>>>> Attention – this script does not seem to work !<<<<<<*

### 4.2 Aktivation of MASS.service and pulseaudio.service

```
sudo systemctl daemon-reload
sudo systemctl enable pulseaudio.service
sudo systemctl enable MASS.service
```

## 5. Configuring PulseAudio

Run these commandos to let PulseAudio run in system-wide mode:

```
sudo usermod -aG audio pulse
sudo usermod -aG pulse-access root
sudo usermod -aG pulse-access pi

systemctl --user stop pulseaudio.socket
systemctl --user stop pulseaudio.service
systemctl --user disable pulseaudio.service
systemctl --user disable pulseaudio.socket

### This creates error message!
# Switch off start-pulseaudio-x11
sudo mv /etc/xdg/autostart/pulseaudio.desktop
/etc/xdg/autostart/pulseaudio.desktop.disabled
```

**Commented [ASE3]:**
*From manual: If you also add this change, the menue will disappear and the Raspberry desktop will become unusable. But this does not seem to be needed after all …*

*Edit file: /etc/pulse/client.conf*
*sudo nano /etc/pulse/client.conf*
*och ändra default-server till: default-server = /run/pulse/native*

## 6. Copy MASS folder to system

Now copy the MASS-folder containing MASS.py and the other scripts to

pi/home/MASS:

- MASS.py
- Config.py
- Audio_player.py
- Audio_recorder.py
- Video_recorder.py

## 7. Installation of python-libraries

MASS.py uses libraries for vlc och pyaudio:

```
sudo pip3 install python-vlc
```

PyAudio is python bindings for PortAudio:

```
sudo apt-get install libportaudio0 libportaudio2 libportaudiocpp0
portaudio19-dev
```

Now install PyAudio with pip3:

```
sudo pip3 install pyaudio
```

Install Video function:

```
sudo apt install ffmpeg
```

**Power off Raspberry Pi**

```
sudo shutdown -h now
```

## 8. Connect to external hardware

Now connect to all other hardware according to the wire-scheme below:

- USB microphone
- USB key with MASS folders
- Relay for lamp
- LAMP
- PIR-sensor

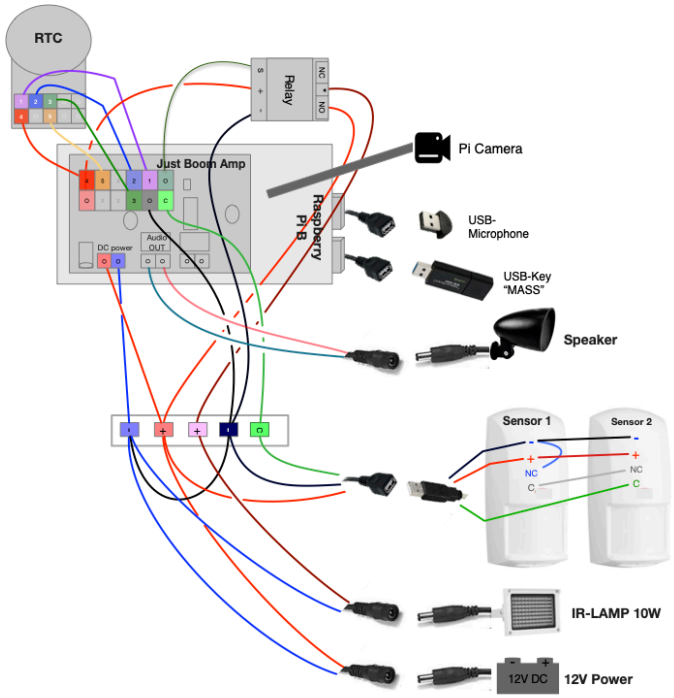
Start the complete system by connecting 12V power

The start-up takes a long time, 2 minutes or more.

You will hear the debug sounds: “Startup done”. (Sometimes even “Testing”.)

Now - the MASS is activated and armed.

**MASS3 wiring scheme**



## 9. Manual tests:

stop MASS.py

```
sudo pkill -f /home/pi/MASS/MASS.py
```

starta

```
~MASS.py
```

See which python-scripts are running:

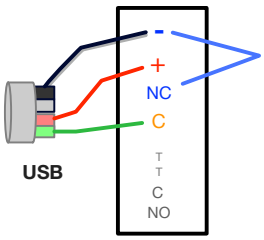
```
ps -aef | grep python
```

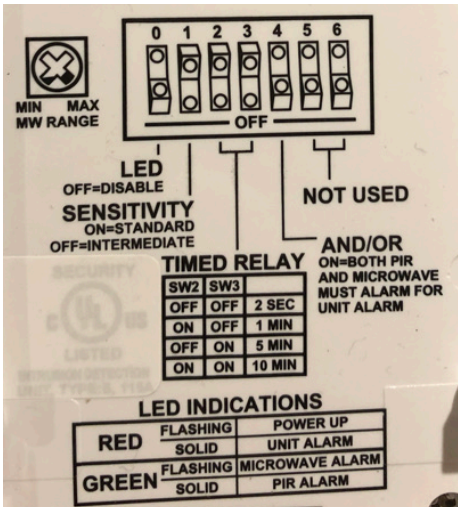
Check whether the signal from PIR-sensor arrives at the pi (save this text as .py document):

```
import pygame import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM) GPIO.setup(23,GPIO.IN)
GPIO.wait_for_edge(23,GPIO.RISING)

print("Yes, Pir-signal is read")
```

## 10. PIR-sensor

Använd USB-kabel med fyra ledare – anslut minus till svart och vitt kabel, plus till röd och grön till C (control). Kortslut minus och NC med kabel (blå).




DIP-switcharna på PIR:ens baksida:

| | |
|---|---|
| 0 | on |
| 1 | on |
| 2 | off |
| 3 | off |
| 4 | on |
| 5 | Not in use |
| 6 | Not in use |

## 11. SCRIPTS:

### 1. Scripts in etc/systemd/system/

**pulseaudio.service**

```
[Unit]
Description=PulseAudio system-wide sound server
After=avahi-daemon.service network.target

[Service]
Type=notify
ExecStart=/usr/bin/pulseaudio --daemonize=no --system --disallow-exit --realtime --no-cpu-limit --log-target=journal
ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target
```

**MASS.service**

```
[Unit]
Description=MASS service
After=pulseaudio.service

[Service]
Type=simple
User=root
ExecStart=strace python3 /home/pi/MASS/MASS.py

[Install]
WantedBy=default.target
```

### 2. Scripts in pi/home/MASS:

**Audio_player.py**

```
import os
```

```
import glob
import datetime as dt
from random import sample, randint
import vlc


class Soundlist(object):
    def __init__(self, data_location):
        self.hour = dt.datetime.now().hour
        self.folder = os.path.join(data_location, str(self.hour))
        self.sounds = sorted(glob.glob(os.path.join(self.folder, '*.mp3')))

    def draw_random(self):
        return(sample(self.sounds, 1)[0])

    def draw_and_drop_first(self):
        first = self.sounds[0]
        self.sounds.pop(0)
        return(first)

    def show(self):
        print(self.sounds)

class AudioPlayer(object):
    def __init__(self, volume):
        self.volume = round(volume)

    def play(self, path):
        self.vlc_instance = vlc.MediaPlayer(path)
        self.vlc_instance.audio_set_volume(self.volume)
        self.vlc_instance.play()

    def is_playing(self):
```

```
        if self.vlc_instance and self.vlc_instance.is_playing():
            return(True)
        else:
            return(False)
```

**audio_recorder.py**

```
import pyaudio, wave
import datetime as dt
import time

class AudioRecorder:

    form_1 = pyaudio.paInt16 # 16-bit resolution
    chans = 1 # 1 channel
    samp_rate = 44100 # 44.1kHz sampling rate
    chunk = 4096 # 2^12 samples for buffer
    #record_secs = 10 # seconds to record
    dev_index = 1 # device index (
    wav_output_filename = '/home/pi/MASS/testljud.wav' # name of .wav file

    def start_record_audio(self, wav_output_filename,):
        self.wav_output_filename = wav_output_filename
        self.audio = pyaudio.PyAudio() # create pyaudio instantiation
        self.wavefile = wave.open(self.wav_output_filename,'wb')
        self.wavefile.setnchannels(self.chans)
        self.wavefile.setsampwidth(self.audio.get_sample_size(self.form_1))
        self.wavefile.setframerate(self.samp_rate)

        def callback(in_data, frame_count, time_info, status):
            self.wavefile.writeframes(in_data)
            return(in_data, pyaudio.paContinue)
```

```
        # create pyaudio stream
        self.stream = self.audio.open(format=self.form_1,
                                      rate=self.samp_rate,
                                      channels=self.chans,
                                      input_device_index=self.dev_index,
                                      input = True,
                                      frames_per_buffer=self.chunk,
                                      stream_callback = callback)
        print("recording")
        self.stream.start_stream()


    def stop_record_audio(self):
        # stop and close stream, terminate pyaudio instantiation
        self.stream.stop_stream()
        self.stream.close()
        self.audio.terminate()

        print("finished recording")
```

**config.py**

```
import configparser


class Config(object):
    def __init__(self, config_file):
        self.config_file = config_file
        self.parser = configparser.ConfigParser()
        self.parser.read(self.config_file)
        for name, value in self.parser.items():
            setattr(self, name, value)
```

**MASS.py**

```
import os
import glob
import shutil
import datetime as dt
import time
import threading
import RPi.GPIO as GPIO
from config import Config
from audio_recorder import AudioRecorder
from audio_player import Soundlist, AudioPlayer
from video_recorder import VideoRecorder


def pretty_timestamp(t):
    return(t.strftime("%Y-%m-%d %H:%M:%S"))


def simple_timestamp(t):
    return(t.strftime("%Y%m%d_%H%M%S"))


def find_USB_drive():
    USB_drive = ''
    start_time = time.time()

    while time.time() < start_time + 300:

        drives = glob.glob('/media/*/*')
        for drive in drives:
            if os.path.isfile(os.path.join(drive, 'input', 'MASS.ini')):
                USB_drive = drive
                print(f"Found USB drive at {USB_drive}")
                return(USB_drive)
            else:
```

```
                print(f"Didn't find USB drive at {drive}")
        time.sleep(10)
    return(USB_drive)


def create_dirs(*dirs):
    t0 = dt.datetime.now()
    while dt.datetime.now() < t0 + dt.timedelta(seconds = try_create_output_dir):
        try:
            for d in dirs:
                if not os.path.exists(d):
                    os.makedirs(d)
            if os.path.exists(d):
                break
        except:
            print('no dir created')
            time.sleep(10)


def setup_GPIO(PIR_sensor, IR_lamp):
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(PIR_sensor, GPIO.IN)
    GPIO.setup(IR_lamp, GPIO.OUT)
    GPIO.setup(PIR_sensor, GPIO.IN, pull_up_down=GPIO.PUD_UP)



def write_to_logfile(log_time, write_data):
    timestamp = pretty_timestamp(log_time)
    with open(log_file, "a") as f:
        f.write(f"{timestamp}\t{write_data}\n")
        f.close()
    print(write_data)

def start_IR_lamp(IR_lamp):
```

```
    GPIO.output(IR_lamp, 1)


def stop_IR_lamp(IR_lamp):
    GPIO.output(IR_lamp, 0)


def play_sound(audioplayer, sound, wait_before_play):
    time.sleep(wait_before_play)
    audioplayer.play(sound)
    write_to_logfile(dt.datetime.now(), f"Playing {sound}")


def wait_for_player(player):
    time.sleep(3)
    while player.is_playing():
        time.sleep(1)


def join_audio_video(video_file, audio_file, output):


    # Use ffmpeg to stich together audio and video
    cmd = str(f"ffmpeg -y -i {video_file} -i {audio_file}"
        f" -c:v copy -c:a aac -map 0:v:0 -map 1:a:0 -r {config.camera['framerate']}
{output} > /dev/null 2>&1")


    print(cmd)
    # Here I define a method to run as a thread
    def run_cmd():
        ret = os.system(cmd)
        if ret == 0:
            # Conversion successful, delete temp files
            write_to_logfile(dt.datetime.now(), f"File saved to {output}")
            os.remove(video_file)
            os.remove(audio_file)
        else:
```

```
            write_to_logfile(dt.datetime.now(), f"Error while converting file
{output}")


    # Start the thread in background. Continue with the rest of the program...
    convert_thread = threading.Thread(target=run_cmd())
    convert_thread.start()


# Find the USB drive - if it is not found, stop execution
USB_drive = find_USB_drive()
if USB_drive == '':
    exit()


today = dt.datetime.now().strftime("%Y-%m-%d")
start_time = dt.datetime.now()


# Input folders, files
input_folder = os.path.join(USB_drive, "input")
config_file = os.path.join(input_folder, "MASS.ini")


print("Looking for input files in", input_folder)
print("Looking for config file at", config_file)


# Read the config file
config = Config(config_file)


# Output folder
project_folder = os.path.join(USB_drive, "output")
# Temporary folder
temporary_output_folder = "/tmp"


# Create a log file for every start (so if we crash we don't loose previous logs)
log_file = os.path.join(project_folder, f"{simple_timestamp(start_time)}.log.txt")
```

```
# Create directories for output if they do not exist
try_create_output_dir = int(config.timing['try_create_output_dir'])
create_dirs(project_folder)

# If output directories were not created, stop execution
if not os.path.exists(project_folder):
    exit()

# Make a copy of the config file, for future reference
output_copy_config = os.path.join(project_folder,
f"{simple_timestamp(start_time)}.config.ini")
shutil.copy(config_file, output_copy_config)



# Setup GPIO
PIR_sensor = int(config.pins['PIR_sensor'])
IR_lamp = int(config.pins['IR_lamp'])
setup_GPIO(PIR_sensor, IR_lamp)

# Import players and recorders
audioRecorder = AudioRecorder()
audioPlayer = AudioPlayer(round(float((config.sound['volume']))))
videoRecorder = VideoRecorder(config.camera)

# Start up parameters
startup_test = input_folder + "/debug_sounds/startup_testing.mp3"
startup_done = input_folder + "/debug_sounds/startup_done.mp3"
startup_duration = float(config.timing['startup_duration'])
#startup_recording_time = float(config.timing['startup_recording_time'])
startup_sleep_between_events = float(config.timing['startup_sleep_between_events'])

# MASS parameters
shuffle = int(config.sound['shuffle'])
```

```
delay_after_camera = float(config.timing['delay_after_camera'])
recording_time = float(config.timing['recording_time'])
sleep_between_events = float(config.timing['sleep_between_events'])


# STARTING UP


write_to_logfile(start_time, "Starting up MASS")
write_to_logfile(start_time, f"Input in {input_folder}")
write_to_logfile(start_time, f"Video recordings in {project_folder}")


while dt.datetime.now() < start_time + dt.timedelta(seconds = startup_duration):


    trigger = GPIO.wait_for_edge(PIR_sensor, GPIO.RISING, timeout=10000)


    # Checking 100 times every 1us -> Prevent false triggering
    for i in range(100):
        if GPIO.input(PIR_sensor) != GPIO.HIGH:
            trigger = None
            break
        else:
            time.sleep(0.000001)


    if trigger is None:
        pass
    else:
        now = dt.datetime.now()
        now_unix = now.timestamp()
        print(f"Motion detected, playing {os.path.basename(startup_test)}")
        write_to_logfile(now, f"Motion detected")


        audio_output_file = os.path.join(temporary_output_folder,
f'mass_{now_unix}.wav')
```

```
        video_output_file = os.path.join(temporary_output_folder,
f'mass_{now_unix}.h264')
        output_file = os.path.join(project_folder, f'{simple_timestamp(now)}.mp4')
        video_label = f'{pretty_timestamp(now)}\n{os.path.basename(startup_test)}'



        # Start audio recording, IR lamp, video recording and
        # soundplayer (0 s waiting before playing sound)
        audioRecorder.start_record_audio(audio_output_file)
        start_IR_lamp(IR_lamp)
        videoRecorder.start_record_video(video_output_file, video_label)
        play_sound(audioPlayer, startup_test, 0)

        # Keep recording while player is busy
        #time.sleep(startup_recording_time)
        wait_for_player(audioPlayer)

        # Stop video recording, IR lamp, audio recording
        videoRecorder.stop_record_video()
        stop_IR_lamp(IR_lamp)
        audioRecorder.stop_record_audio()
        join_audio_video(video_output_file, audio_output_file, output_file)

    # Pause before enabling motion detection
    time.sleep(startup_sleep_between_events)


audioPlayer.play(startup_done)
wait_for_player(audioPlayer)

# STARTUP IS DONE, RUNNING MASS
soundlist = Soundlist(input_folder)
while 1:
    trigger = GPIO.wait_for_edge(PIR_sensor, GPIO.RISING)
```

```
    now = dt.datetime.now()
    now_unix = now.timestamp()

    write_to_logfile(now, f"Motion detected")

    # Variables indicating if a new soundlist is needed
    new_hour = now.hour != soundlist.hour
    soundlist_is_empty = len(soundlist.sounds) == 0

    if new_hour or soundlist_is_empty:
        # New hour, new list of sounds
        soundlist = Soundlist(input_folder)

    # If no shuffling: draw the first sound from the list, without replacement
    if shuffle == 0:
        sound = soundlist.draw_and_drop_first()
    # If shuffling: draw a random sound from the list, with replacement
    elif shuffle == 1:
        sound = soundlist.draw_random()

    #write_to_logfile(now, f"Playing {sound}")

    audio_output_file = os.path.join(temporary_output_folder,
f'mass_{now_unix}.wav')
    video_output_file = os.path.join(temporary_output_folder,
f'mass_{now_unix}.h264')
    output_file = os.path.join(project_folder,
f'{simple_timestamp(now)}_{os.path.basename(sound)}.mp4')
    video_label = f'{pretty_timestamp(now)}\n{os.path.basename(sound)}'

    # Run function "play_sound" in separate thread (= non blocking)
    # The function includes waiting time (delayAfterCamera)
```

```
    # before playing the sound
    player_thread = threading.Thread(target=play_sound, args=[audioPlayer, sound,
delay_after_camera])
    player_thread.start()


    # Start audio recording, IR lamp, video recording
    audioRecorder.start_record_audio(audio_output_file)
    start_IR_lamp(IR_lamp)
    videoRecorder.start_record_video(video_output_file, video_label)


    # Keep recording dudring recording_time
    time.sleep(recording_time)


    # Stop video recording, IR lamp, audio recording
    videoRecorder.stop_record_video()
    stop_IR_lamp(IR_lamp)
    audioRecorder.stop_record_audio()
    join_audio_video(video_output_file, audio_output_file, output_file)


    # Pause before enabling motion detection again
    time.sleep(sleep_between_events)
```

**Video_recorder.py**

```
from picamera import PiCamera, Color
import getpass


class VideoRecorder(object):
    def __init__(self, cam_config):
        self.camera = PiCamera()
        self.camera.resolution = (int(cam_config['resolution_w']),
int(cam_config['resolution_h']))
        self.camera.framerate = int(cam_config['framerate'])
```

```
        self.camera.exposure_mode = cam_config['exposure_mode']
        self.camera.awb_mode = cam_config['awb_mode']
        self.camera.brightness = int(cam_config['brightness'])
        self.camera.saturation = int(cam_config['saturation'])
        self.camera.rotation = int(cam_config['rotation'])
        self.camera.ISO = int(cam_config['ISO'])
        self.camera.annotate_text_size = int(cam_config['annotate_text_size'])
        self.camera.annotate_background = Color(cam_config['annotate_background'])
        self.camera.annotate_foreground = Color(cam_config['annotate_foreground'])

    def start_record_video(self, output_filename, label):
        self.camera.annotate_text = label
        self.camera.start_recording(output_filename)

    def stop_record_video(self):
        self.camera.stop_recording()
```

**3. In MASS-usb drive /input:**

**MASS.ini**

```
; This is the manual configuration of the MASS service. Keep this file on the USB-
key "MASS/input/MASS.ini".

; Do not change unless hardware is changed
[pins]
PIR_sensor = 23
IR_lamp = 24

; adjust camera timing in seconds
[timing]
try_create_output_dir = 300
startup_duration = 60
```

```
startup_recording_time = 5
startup_sleep_between_events = 10
; sound delay
delay_after_camera = 5
;total recording time
recording_time = 30
;pause between events
sleep_between_events = 5

; adjust camera settings
[camera]
resolution_w = 1280
resolution_h = 720
framerate = 30
exposure_mode = night
awb_mode = auto
brightness = 70
contrast = 70
saturation = -100
ISO = 800
rotation = 0
annotate_text_size = 20
annotate_background = black
annotate_foreground = white

; adjust volume and file selection
[sound]
volume = 70
shuffle = 0
```