Highlights

Automating Behavioral Analysis in Neuroscience: Development of an Open-Source Python Software for More Consistent and Reliable Results

Cerveira, A.J.D.O.[1], Ramalho, B.A.C.[1], de Souza, C.C.B., Spadaro, A.P., Ramos, B.A., Wichert-Ana, L., Padovan-Neto, F.E., de Lacerda, K.J.C.C.

- Development of an automated analysis protocol for the Morris Water Maze (MWM) and Open Field (OF) tests using the OpenCV library in Python.

- Utilization of the OpenCV library in Python for efficient and accurate tracking and navigation identification in the MWM and assessment of mice behavior in the OF test.

- Time-saving and reduction of human errors achieved through automated analysis, providing more consistent information about animal behavior during the tests.

- Advancement in experimental techniques, offering researchers a valuable tool for objective and efficient data analysis in neuroscience studies.

Automating Behavioral Analysis in Neuroscience: Development of an Open-Source Python Software for More Consistent and Reliable Results

Cerveira, A.J.D.O.[1a], Ramalho, B.A.C.[1b], de Souza, C.C.B.[a], Spadaro, A.P.[b], Ramos, B.A.[a], Wichert-Ana, L.[c], Padovan-Neto, F.E.[a], de Lacerda, K.J.C.C.[a,c]

[a]*Department of Psychology, Faculty of Philosophy, Sciences and Letters of Ribeirão Preto, University of São Paulo, Ribeirão Preto, Brazil,*
[b]*Department of Biomolecular Sciences, Faculty of Pharmaceutical Sciences of Ribeirão Preto, University of São Paulo, Ribeirão Preto, Brazil,*
[c]*Nuclear Medicine & Molecular Imaging Section, Image Science and Medical Physics Center, Internal Medicine Department and Postgraduate Program, Ribeirão Preto School of Medicine, University of São Paulo, Ribeirão Preto, Brazil,*

Abstract

Background: The application of automated analyses in neuroscience has become a practical approach. With automation, the algorithms and tools employed perform fast and accurate data analysis. It minimizes the inherent errors of manual analysis performed by a human experimenter. It also reduces the time required to analyze a large amount of data and the need for human and financial resources. ***Methods:*** In this work, we describe a protocol for the automated analysis of the Morris Water Maze (MWM) and the Open Field (OF) test using the OpenCV library in Python. This simple protocol tracks mice navigation with high accuracy. ***Results:*** In the MWM, both automated and manual analysis revealed similar results regarding the time the mice stayed in the target quadrant ($p = 0.109$). In the OF test, both automated and manual analysis revealed similar results regarding the time the mice stayed in the center ($p = 0.520$) and in the border ($p = 0.503$) of the field. ***Conclusions:*** The automated analysis protocol has several advantages over manual analysis. It saves time, reduces human errors, can be customized, and provides more consistent information about animal behavior during tests. We conclude that the automated protocol described here is reliable and provides consistent behavioral analysis in mice. This automated protocol could lead to deeper insight into behavioral neuroscience.

Keywords: Automated analysis using python, Morris water maze test, Open Field test, openCV image processing

1. Introduction

Automated analysis of experiments is an advanced approach that uses automated algorithms and tools to analyze the results of scientific experiments and uncover valuable insights from the data[1, 2, 3]. This approach presents several advantages over manual analysis, making it a common choice in many research and development areas. One of the programming languages that has been assisting in the development of software for automated data analysis is Python [4, 5, 6, 7]. Python is an open-source programming language that has been gaining popularity in research and development areas. One of the main reasons for this is its ease of use, flexibility, and processing power. This makes it an attractive option for automated analysis of experiments, as it allows researchers to develop sophisticated and efficient algorithms more quickly and easily[4, 5, 6, 7]. Using Python for the development of algorithms applied to automated experiment analysis offers several advantages because it has several libraries and modules that are specially designed for data analysis[8, 9]. Furthermore, Python is a well-documented programming language with a wide community of users, which means that there are a large number of online resources available to help developers learn and use the language. This includes open-source libraries, discussion forums, and online tutorials, which allow users to learn the language and develop advanced data analysis

Email addresses: ferpadovan@usp.br (Padovan-Neto, F.E.), kleython_lacerda@usp.br (de Lacerda, K.J.C.C.)

algorithms quickly and efficiently[10, 11, 12].

The OpenCV (Open Source Computer Vision Library)[13, 14, 15] is a powerful tool for image processing and computer vision, with comprehensive features, portability, ease of use, and a large community of users. Launched in 2000 by Intel, it is open source for computer vision and image processing[13]. Since then, it has become one of the most popular libraries for image processing and computer vision in various areas, such as robotics, artificial intelligence, gaming, medicine, and many others. One of the main qualities of OpenCV is its wide range of image-processing features[13, 14, 15]. The library supports a variety of image-processing tasks, such as object detection, face recognition, object tracking, motion analysis, and much more. In addition, it has highly optimized and efficient computer vision algorithms, allowing users to process images in real time. Another advantage of OpenCV is its portability. The library is compatible with several platforms, including Windows, Linux, macOS, Android, and iOS. This means that users can write code on one platform and use it on another without significant modifications, making the development of cross-platform applications easier[14]. OpenCV is also an easy-to-use library with clear and extensive documentation. It is written in C++ but has a Python interface, allowing users to develop Python applications without in-depth knowledge of C++[16].

In this work, we present a protocol for the Morris Water Maze (MWM) and the Open Field Test (OF) in mice and a Python algorithm for their automated analysis that can be used for recordings made with any type of digital camera. The automated analysis of these tests in Python using the OpenCV library is a technique that uses image processing tools to identify the behavior of mice in a controlled environment, performed through edge detection techniques[8], binarization, segmentation, and object tracking[13, 14, 15, 16, 17, 18, 19]. The MWM is a widely used neuroscientific assessment tool for evaluating spatial memory and animal's learning. However, the accuracy and objectivity of the data obtained from this test can be improved through the use of the OpenCV library, which enables the detection and tracking of mouse movements in the maze. Similarly, the OF test is frequently used for the evaluation of general locomotor activity, anxiety levels, and exploratory behavior. The OpenCV library offers the ability to mark the quadrants of the field and quantify how much time each mouse spends at the center area versus the edge of the environment, which is a variable commonly used to assess anxiety-like behavior[20]. The library could also assess other variables such as exploration and locomotor activity levels, which include the number of quadrants traversed by the animal during their movement[21, 23]. By utilizing the OpenCV library in these assessments, researchers can obtain more precise and reliable data to enhance their analysis and understanding of animal's behavior. Taken together, this automated approach saves time and resources, minimizes human observers bias in interpreting results, is efficient and reliable, and provides crucial information for neuroscience research[24, 25, 26].

2. Materials and Methods

2.1. *Animals*

Male B6129SF2/J mice (eight months old, n=8) were used in the MWM test. Male C57BL/6 mice (seven months old, n=10) were used in the OF test. Animals were housed in groups of three to five per cage in a temperature-controlled room (24 ± 1 C) with a 12/12 h light/dark cycle with free access to food and water[22]. All animals were tested during the light cycle and animal procedures were approved by the Ethics Committee for Animal Experimentation (CEUA) from the Faculty of Philosophy, Sciences and Letters of Ribeirao Preto at the University of Sao Paulo (protocol number **22.1.195.59.0**).

2.2. *Morris Water Maze*

Richard G. Morris, a psychologist, created the Morris Water Maze test in 1982[27]. The MWM has been widely used in neuroscience studies to evaluate the role of different brain regions in spatial learning[28]. The test comprises a circular platform that is submerged in water. Animals use distal cues to locate a submerged platform to escape from the water. For instance, this test has been used to evaluate the efficacy of drugs or other interventions in preclinical models of Alzheimer's disease[29, 30, 31, 32]. Its widespread use in neuroscience research and neurodegenerative disease investigation has contributed to the advancement of knowledge about brain function and the development of effective treatments for neurological diseases.

2.2.1. *Behavioral Apparatus*

A circular polyethylene pool with a diameter of 100 cm and a height of 60 cm with non-reflective interior surfaces was used in this study. The pool was filled with water containing 100 g of skim milk powder (0.637g/L) at room temperature (25 +/- 1ºC) up to a level of 20 cm. The pool was divided into four quadrants based on the coordinates N, S, E, and W. The quadrants were named as: LT (Left-Top), LB (Left-Bottom), RT (Right-Top) e RB (Right-Bottom). Each quadrant contained a visual cue of a different shape and color (i.e., an orange square, a green circle, a blue star and a red cross respectively). A circular transparent platform with a diameter of 8 cm was placed 0.5 cm above the opaque water level in the target quadrant (i.e., LT quadrant).

2.2.2. *Training*

Each animal underwent to a session of four trials per day during five consecutive days. Each trial lasted 60 seconds, and it was followed by a 10-minute interval. During each trial, the animals were pseudo-randomly positioned in a different quadrant so they could face a different visual cue. If the mouse could not find the platform within 60 seconds, an experimenter gently guided the animal to the platform. Once on the platform, the experimenter ensured that each mouse stayed off the water for 15 seconds. After each trial, the mouse was dried and put back in its cage.

2.2.3. *Test*

The platform was eliminated during the test. Each mouse was left in the pool for a duration of 120 seconds. Even when the platform is absent, animals tend to spend more time in the target quadrant[33]. The percentage of time spent in the target quadrant is indicative of the level of spatial memory retention (also known as reference memory)[31]. We determined the amount of time each animal spent in the designated quadrant, as well as the number of crossings between quadrants.

2.3. *Open Field Test*

The open field is one of the most frequently used test in animal behavior research, especially in neuroscience[22, 34, 35]. The test was created by Calvin S. Hall in 1934[36] and has undergone numerous modifications to its experimental protocol and method, including changes to the dimensions and configuration of the testing arenas and the length of the experimental session[37]. The test distinguishes itself as a versatile evaluation instrument, able to provide valuable insights regarding locomotor activity, exploratory behavior, and anxiety levels[38].

2.3.1. *Behavioral Apparatus*

The experiment was conducted in a single session in a square arena (45 x 45 x 35 cm) made of Plexiglass. The test involves placing the animal in the center of the apparatus and allowing it to explore for 10 minutes. A video camera was positioned above the arena to record animal's activity.

2.4. *Automated algorithm for the analysis of the Morris Water Maze and Open Field test in Python using the OpenCV library*

OpenCV (Open Source Computer Vision Library) is an open source library designed to provide a more developer-friendly infrastructure for computer vision applications. It currently has more than 500 functions. It can, for instance, detect and recognize features, identify objects, extract three-dimensional object models, and identify and track moving objects. Integration with other libraries, such as Numpy and Matplotlib, increases processing capacity and applications. OpenCV is compatible with Windows, Linux, Android, and Mac OS, and it supports a wide range of programming languages, including Python, C++, Java, and MATLAB.

2.4.1. The algorithm

The algorithm requires the installation and proper configuration of the following libraries: OpenCV version 4.7.0, Tkinter version 8.6 or later, and Matplotlib version 3.5.3 or later. These can be installed in the terminal or command prompt using the following commands:

```
pip install opencv-python==4.7.0
pip install matplotlib>= 3.5.3 pip install python-tk
```

Wait until the libraries and their dependencies have been downloaded and installed. Depending on your operating system and configurations, it may be necessary to add administrative privileges or use a virtual environment to install the package.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import math
import tkinter as tk
from tkinter import filedialog
import os
```

The tkinter library then provides a dialog window for the user to select a video file. The file name is then extracted, and the extension is eliminated.

```
source_video = filedialog.askopenfilename()
filename = os.path.basename(source_video)
filename_WE = os.path.splitext(filename)[0]
```

The algorithm will prompt the user to choose which test type to analyze (MWM or OF). For MWM analysis, the user must select "1" and for OF analysis, "2". This response will be saved to the test variable. Identify the animal and specify the time the analysis should start (in seconds). Note that in the event that the recording of the video precedes the introduction of the animal in the apparatus by the experimenter, the algorithm provides the option for the user to select a specific time (in seconds) to start the analysis (i.e., the specific time that the animal has been introduced into the behavioral apparatus). Finally, specify the total analysis time (in minutes).

```
test = int(input("Please, press 1 to analyze the Morris Water
Maze or 2 to analyze the Square Open Field: "))
if test == 1:
        escapeLatency = int(input("To calculate the escape
latency, press 1. To do something else, press 2: "))
```

The variable identification stores the animal's identification number in the test.

```
identification = input("Type the animal identification: ")
```

The following variables will be initialized: *milliseconds*, *tan*, *totalTime*, and *finalTime*. *milliseconds* correspond to time the analysis should start. *totalTime* corresponds to the entire duration of the test. The user determines the duration of the test in minutes, and the variable *totalTime* converts it to milliseconds.

```
milliseconds = int(input("Please, enter the initial analysis
time in seconds: "))*1000
tan = milliseconds
totalTime = int(input("Please, enter the total analysis time in
minutes: "))*60000
finalTime = milliseconds + totalTime
cap.set(cv2.CAP_PROP_POS_MSEC, milliseconds)
```

If the directory containing the algorithm does not have a folder named "result", a new one will be created in the same directory. This folder will be populated with two text files. These text files will be labeled according to animal identification and will store the analyzed data.

```
if not os.path.exists('result'):
        os.makedirs('result')
```

```
arq = open(f'result/position-identification.txt', 'a')
data = open(f'result/data-identification.txt', 'a')
```

It is necessary to define variables that will subsequently be used in the code:

```
contrast = 80
cp = [0, 0, 0, 0]
tp = [0, 0, 0, 0]
posx = [ ]
posy = [ ]
pi = 0
pVerification = 0
r0 = 0
kx = [ ]
ky = [ ]
rc = [ ]
circle = []
elVerification = 0
tel = [0]
```

The "selectROIfromFrame" function permits the selection of a region of interest (ROI). This function returns the coordinates and dimensions of the specified region of interest.

```
def selectROIfromFrame(frame):
        box = cv2.selectROI("SELECT ROI", frame,
fromCenter=False, showCrosshair=False)
        return box
```

The "roi" function enables the selection of a region of interest using the cv2 library from OpenCV. In the top-left corner, where the user starts selecting the ROI corresponding to the arena area, the coordinate 0,0 is defined. When the user selects the entire arena, 4 horizontal and 4 vertical lines (depending on the height and width selected by the user) are created, forming 16 equal squares. The code pre-establishes that the 4 central squares represent the central area, while the remaining squares correspond to the border area. Note that this section is only executed for the OF test.

```
def roi(frame):
        roi = cv2.selectROI(frame)
        x, y, w, h = roi
        rect_width = int(w / 4)
        rect_height = int(h / 4)
        rect_coords = []
        for i in range(4):
                for j in range(4):
                        rect_x = x + i * rect_width
                        rect_y = y + j * rect_height
                        rect_coords.append((rect_x, rect_y))
        rc.extend((rect_coords[5],rect_coords[15]))
        cv2.destroyAllWindows()
```

The following segment of the algorithm is only executed for the MWM test. The "click_event" function is an external function that is called when a mouse event is detected in an image window displayed using the cv2 library from OpenCV. If it detects a left mouse button click, the function will print the x and y coordinates of the click location in the terminal and store these coordinates in a list. For the MWM, four initial clicks are required to designate the cardinal coordinates (N, S, E, and W). The position of each of these clicks is stored so that the algorithm can delineate the 4 quadrants. After each click, the corresponding letter is drawn on the selection window using the putText function from the cv2 library. Once all four clicks are registered, the function closes all the opened windows.

```
def click_event(event, x, y, flags, params):
        if event == cv2.EVENT_LBUTTONDOWN:
        kx.append(x)
        ky.append(y)
        if len(kx) == 1:
                loc = 'B'
        elif len(kx) == 2:
                loc = 'T'
        elif len(kx) == 3:
                loc = 'L'
        elif len(kx) == 4:
```

```
        loc = 'R'
        cv2.destroyAllWindows()
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img, loc, (x,y), font, 1, (0, 0, 255), 2)
    cv2.imshow('image', frame)
```

The "quadOF" function is used when the OF test is selected. It examines two coordinates (c0 and c1) to determine if they fall within the central square previously recorded in the "rc" variable. If c0 and c1 are located within this square, the function returns the letter 'C' to indicate that the coordinates are located within the central square. Otherwise, the function returns 'B' to indicate that the coordinates are on the square's border.

```
def quadOF(c0,c1):
    if rc[0][0] <= c0 <= rc[1][0] and rc[0][1] <= c1 <= rc[1][1]:
        lq = 'C'
    else:
        lq = 'B'
    return lq
```

When the selected test is the MWM, the "quadMMW" function employs four reference points to divide the image into four quadrants and determines in which quadrant the input coordinate is located. The function returns whether the coordinate is to the left (L) or right (R) and above (T) or below (B) the image's center.

```
def quadMMW(c0, c1):
    p1 = [kx[1],ky[1]]
    p2 = [kx[0],ky[0]]
    p3 = [kx[2],ky[2]]
    p4 = [kx[3],ky[3]]
    m1 = (p2[1]-p1[1])/(p2[0]-p1[0])
    m2 = (p4[1]-p3[1])/(p4[0]-p3[0])
    x = ((c1 - p2[1]) + m1*p2[0])/m1
    y = m2*c0 - m2*p3[0] + p3[1]
    if c0 >x :
        sw = 'R'
    else:
        sw = 'L'
    if c1 <y:
        sh = 'T'
    else:
        sh = 'B'
    return sw, sh
```

The "increase_brightness" function takes an image (in BGR format) and a brightness value that should be added to all pixels of the image. The function converts the image from BGR to HSV (Hue, Saturation, Value), increases the value of the pixels by the specified value units (by default, the value is set to 30), limits them to a maximum of 255, and then converts it back to BGR before returning the resulting image. This function can be used to increase the brightness of an image in a straightforward way.

```
def increase_brightness(img, value=30):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)
    lim = 255 - value
    v[v >lim] = 255
    v[v <= lim] += value
    final_hsv = cv2.merge((h, s, v))
    img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    return img
```

If the user wants to calculate the escape latency, the following methods have been inserted so that the user can define the region in which the escape platform is located and check if the ROI is enclosed in the circle:

```
def plataform():
    ret, frame = cap.read()
    frame = cv2.resize(frame, (new_width, new_height))
    frame = increase_brightness(frame, contrast)
    box = selectROIfromFrame(frame)
    cv2.destroyAllWindows()
    x, y, w, h = box
```

```
        circle.append(x)
        circle.append(y)
        circle.append(x + w // 2)
        circle.append(y + h // 2)
        circle.append(min(w, h) // 2)
def isCircle(c0,c1,actualTime):
        global elVerification
        distance = math.sqrt((c0 - circle[2])**2 + (c1 -
circle[3])**2)
        if distance <= circle[4] and elVerification == 0:
                tel[0] = actualTime
                elVerification += 1
```

The subsequent phase is a conditional structure that determines whether the module is being executed as the main program. The program resizes and brightens the initial image capture.

```
if __name__ == "__main__":
        ret, frame = cap.read()
        frame = cv2.resize(frame, (new_width, new_height))
        frame = increase_brightness(frame,contrast)
```

If the designated test is the MWM in a conditional structure, the program will display the image and wait for a computer mouse selection from the user. If test 1 (MWM) was not selected, the "roi" function is called to perform object detection in a ROI associated with the OF test.

```
        if teste == 1:
                cv2.imshow('image', frame)
                cv2.setMouseCallback('image', click_event)
                cv2.waitKey(0)
                cv2.destroyAllWindows()
        else:
                roi(frame)
```

The following snippet aims to capture the first frame of the video, resize the image, and increase its brightness. Then, a ROI is selected from the frame, ff the user selects the escape latency option, the function for region selection with the platform is called, and animal tracking is initiated using the OpenCV library.

```
        ret, first_frame = cap.read()
        first_frame = cv2.resize(first_frame, (new_width,
new_height))
        first_frame = increase_brightness(first_frame, contrast)
        box = selectROIfromFrame(first_frame)
        tracker = cv2.TrackerCSRT_create()
        ok = tracker.init(first_frame, box)
        cv2.destroyAllWindows()
        if test == 1 and escapeLatency == 1:
                plataform()
```

The algorithm updates the position of a tracked object using a tracking algorithm and draws a rectangle around it in the output image. In addition, the code stores the position's x and y coordinates in lists and writes them to one of the files created within the "results" folder. The progression of the algorithm's analysis varies depending on the test selected at the beginning of the analysis.

```
        while milliseconds <= finalTime:
                ret, frame = cap.read()
                frame = cv2.resize(frame, (new_width,
new_height))
                frame = increase_brightness(frame, contrast)
                milliseconds =
cap.get(cv2.CAP_PROP_POS_MSEC)
                vb = False
                if escapeLatency == 1:
                        cv2.circle(frame, (circle[2], circle[3]),
circle[4], (0, 0, 255), 2)
                if not ret:
                        break
```

We have implemented the ability to quickly and easily reselect a Region of Interest (ROI). The user can press the "r" key at any time to readjust the ROI selection. This saves time, effort, and increases user satisfaction by allowing quick refinements to achieve more accurate and satisfactory results.

```
        if cv2.waitKey(1) == ord('r'):
            ret, frame = cap.read()
            frame = cv2.resize(frame, (new_width, new_height))
            frame = increase_brightness(frame, contrast)
            milliseconds = cap.get(cv2.CAP_PROP_POS_MSEC)
            box = selectROIfromFrame(frame)
            tracker = cv2.TrackerCSRT_create()
            ok = tracker.init(frame, box)
            cv2.destroyAllWindows()
```

The code continuously updates the position of the tracked object. Additionally, it stores the x and y coordinates of the position in lists and writes the values to the files created within the ”result” folder.

```
        ok, box = tracker.update(frame)
        if ok:
            pt1 = (int(box[0]), int(box[1]))
            pt2 = (int((box[0] + box[2])), int((box[1] + box[3])))
            c0 = int((box[0] + box[2]/2.0))
            c1 = int((box[1] + box[3]/2.0))
            posicao = str(c0) + ',' + str(c1) + '\n'
            posx.insert(pi, c0)
            posy.insert(pi, c1)
            pi += 1
            arq.writelines(posicao)
            cv2.rectangle(frame, pt1, pt2, (255, 0, 0), 2, 1)
```

After that, if the selected test is MWM, it calls the ”quadMWM” function and stores the values of ”sw” and ”sh”. If the selected test is the OF, it calls the ”quadOF” function and stores the value of ”loc”.

```
        if teste == 1:
            sw,sh = quadMMW(c0,c1)
            if escapeLatency == 1:
                isCircle(c0,c1,(milliseconds-tan)/1000)
        else:
            loc = quadOF(c0,c1)
            cv2.rectangle(frame, rc[0], rc[1], (0, 0, 0), 2)
        else:
            milliseconds = cap.get(cv2.CAP_PROP_POS_MSEC) + 300
            cap.set(cv2.CAP_PROP_POS_MSEC, milliseconds)
            ret, frame = cap.read()
            frame = cv2.resize(frame, (new_width, new_height))
            frame = increase_brightness(frame, contrast)
            box = selectROIfromFrame(frame)
            tracker = cv2.TrackerCSRT_create()
            ok = tracker.init(frame, box)
            cv2.destroyAllWindows()
```

Structural conditions (“if”, “elif” and “else”) are created to check the positions of the coordinates related to the animal. Based on these positions, the text display’s positions and variables are defined. In the case of the MWM test, the function checks if the animal’s coordinate is in one of the four quadrants of the image and sets the text based on that quadrant. For example, if the animal’s coordinate is inside the target quadrant, the text will be displayed in the center of the image.

```
        font = cv2.FONT_HERSHEY_SIMPLEX
        if teste == 1:
            if sw=='R' and sh == 'T':
                lx = 438
                ly = 134
                k = 'RT ' + str(cp[0])
                p = 0
            elif sw=='R' and sh == 'B':
```

```
            lx = 398
            ly = 351
            k = ’RB ’ + str(cp[1])
            p=1
        elif sw==’L’ and sh == ’B’:
            lx = 176
            ly = 329
            k = ’LB ’ + str(cp[2])
            p=2
        else:
            lx = 216
            ly = 134
            k = ’LT ’ + str(cp[3])
            p=3
    else:
        if loc==’C’:
            lx = 300
            ly = 200
            k = ’Centro ’ + str(cp[0])
            p = 0
        else:
            lx = 300
            ly = 300
            k = ’Bordas ’ + str(cp[1])
            p=1
```

The variable ”p” represents the current quadrant where the animal is located, and ”pVerification” is a variable that stores the previously-verified quadrant. If ”p” is different from ”pVerification”, it means that the animal has entered a new quadrant, and the count of crossings between quadrants is incremented. In addition, the algorithm can determine how much time the animal spent in each quadrant. The time is then updated, and the loop continues to run until the user selects the ’q’ key.

```
    if (p != pVerification):
        cp[p] += 1
        mt = (milliseconds - tan)
        tp[pVerification] += (mt-r0)/1000
        r0 = mt
        pVerification = p
        vb = True
    cv2.putText(frame, k,(lx,ly), font, 1,(150, 9, 2), 2)
    cv2.imshow(’Tracking mice’, frame)
    cap.set(cv2.CAP_PROP_POS_MSEC,
milliseconds+100)
    if cv2.waitKey(10) == ord(’q’):
        break
if vb == False:
    tp[p] += (((milliseconds - tan)-r0)/1000)
```

If the test value is equal to 1 (MWM), the statistics for each of the four quadrants are written, including the number of times the animal crossed that quadrant (cp) and the total time in each of the quadrants (tp). If the test value is equal to 2 (OF), the statistics for the center and the edges are written.

```
if teste == 1:
    txt1 = ’#RT = ’ + str(cp[0]) + ’ #RB = ’ +
str(cp[1]) + ’ #LB = ’ + str(cp[2]) + ’ #LT = ’ + str(cp[3]) +
’\n’
    txt2 = ’tRT = ’ + str(tp[0]) + ’ tRB = ’ +
str(tp[1]) + ’ tLB = ’ + str(tp[2]) + ’ tLT = ’ + str(tp[3]) + ’
Escape latency = ’ + str(tel[0])
    p1 = [kx[0], kx[1]]
    p2 = [ky[0], ky[1]]
    p3 = [kx[2], kx[3]]
    p4 = [ky[2], ky[3]]
else:
    txt1 = ’#Center = ’ + str(cp[0]) + ’ #Borders = ’
+ str(cp[1]) + ’\n’
    txt2 = ’tCentro = ’ + str(tp[0]) + ’ tBordas = ’ +
str(tp[1])
print(txt1)
print(txt2)
arq.close()
data.writelines(txt1 + txt2)
data.close()
```

The last section of the algorithm is responsible for creating 3 plots using the Matplotlib library.

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
ax1.plot(posx,posy)
if teste == 1:
    ax1.plot(p1,p2)
    ax1.plot(p3,p4)
    ax2.bar(['RT', 'RB', 'LB', 'LT'],cp)
    ax3.bar(['RT', 'RB', 'LB', 'LT'],tp)
else:
    rect = patches.Rectangle(rc[0],
rc[1][0]-rc[0][0], rc[1][1]-rc[0][1], linewidth=1,
edgecolor='r', facecolor='none')
    ax1.add_patch(rect)
    ax2.bar(['Centro', 'Borda'],[cp[0],cp[1]])
    ax3.bar(['Centro', 'Borda'],[tp[0],tp[1]])
fig = plt.gcf()
plt.show()
fig.savefig(f'result/graph-identification.png',
format='png')
cv2.destroyAllWindows()
```

2.5. Statistical Analysis

Descriptive statistics were used to summarize the mean, standard deviation, median, and interquartile range. The Shapiro-Wilk test was utilized to verify the normality of the data. The confidence interval was set at 95%, and a significance level of 5% was adopted. Provided that the assumptions of normality were satisfied, the paired t-test was employed to assess the difference between the means of the two samples.

3. Results

3.1. Morris Water Maze analysis

Upon initiating the algorithm, the user is prompted with a dialog box, wherein they are required to select the specific test to be analyzed, with the MWM test being designated as option ”1”. Subsequently, it is imperative for the user to ascertain the animal's identity, specify the initial time of analysis in the video, and determine the overall duration of the analysis. Subsequently, the user is prompted to designate the cardinal directions of north (N), south (S), east (E), and west (W) by utilizing left mouse button clicks, as depicted in Figure 1. The N, S, E, and W coordinates were derived by clicking on the regions between the four visual cues affixed to the pool's wall. This procedure divides the pool into four equal quadrants, each associated with a reference point indicated by a visual cue. The N, S, E, and W coordinates are determined based on these reference points. Upon selecting the ”W” coordinate, all four coordinates' positions are promptly stored, and a subsequent window is initiated to proceed with the analysis.

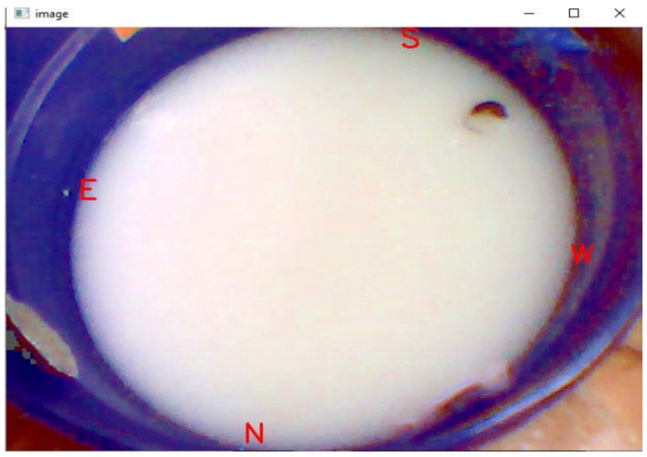

Figure 1: A screenshot depicting the initial window generated by the algorithm during the evaluation of the Morris Water Maze (MWM) test. In this example, the user has designated the four coordinates N, S, E, and W.

The algorithm determines the four quadrants, that is, left-top (LT), right-top (RT), right-bottom (RB), and left-bottom (LB), based on the coordinates specified by the user. Subsequently, the user is required to initiate the animal's selection screen by pressing the ”Enter” key. It is easy to establish a region of interest (ROI) by using a mouse on a computer to draw a rectangle, as illustrated in Figure 2.

Upon selection of the ROI, the user is required to press the ”Enter” key to initiate the algorithmic analysis. The algorithm displays the frequency of the mouse's crossing of the target quadrant (i.e., LT). Figure 3 illustrates that the mouse success-

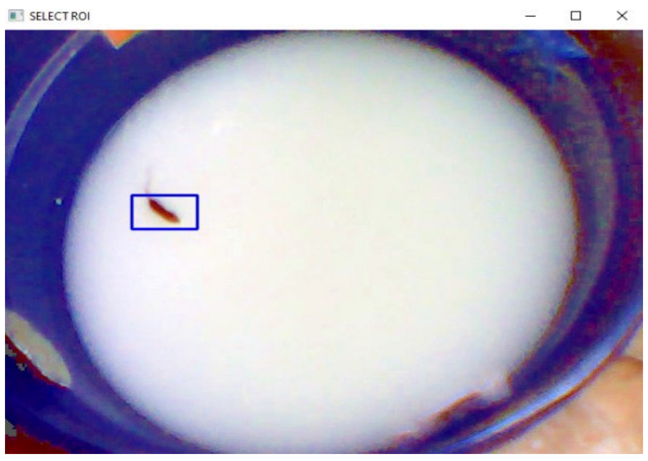

Figure 2: The MWM test test procedure generates a secondary window through the algorithm. The user is subsequently directed to define a region of interest (ROI) for this particular animal, as indicated by the rectangular frame depicted in the figure.

fully navigated to the target quadrant, as denoted by the ”LT” label in the figure. The numerical value ”2” depicted in the figure denotes that the mouse has successfully oriented itself towards the target quadrant on two distinct occasions.

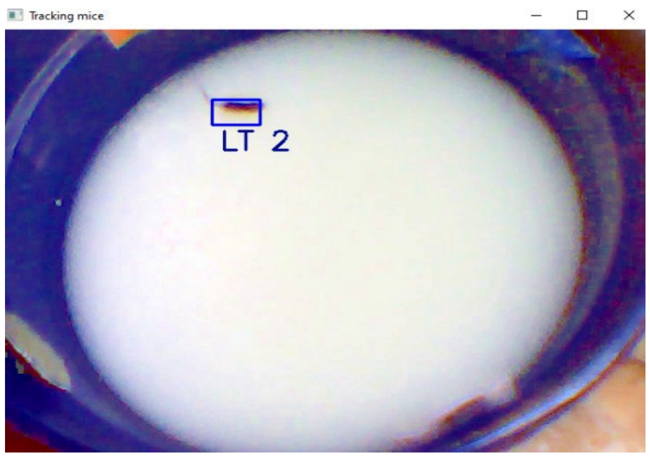

Figure 3: The image depicts a mouse being monitored by a blue rectangular region of interest (ROI). The numerical value of ”2” indicates the number of times the mouse crossed into the target quadrant (LT).

The software will map the center point of the animal’s Region of Interest (ROI) to determine its quadrant location. If the center point of the ROI falls precisely between two quadrants, the quadrant where the animal was previously located will be considered by the software for parameter analysis.

If a user needs to measure the escape latency of the animal, they must select the ROI for the platform. The escape latency will be calculated when the central coordinates of the animal’s ROI fall within the ROI for the platform. Although the ROI for the platform will remain visible throughout the entire test execution (as shown in Figure 4), the escape latency will only be calculated once.

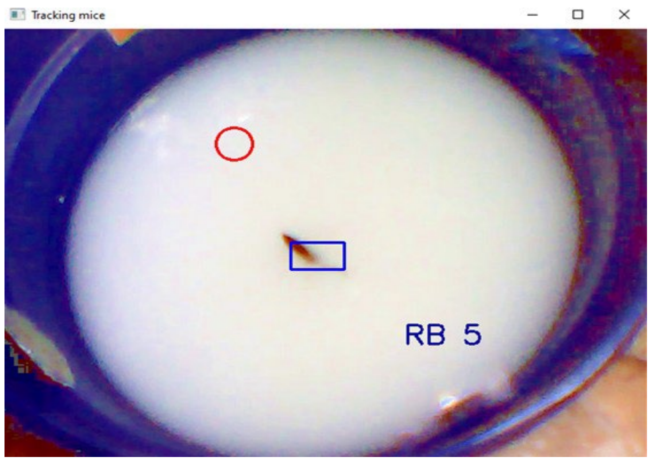

Figure 4: The image depicts a mouse being monitored by a blue rectangular ROI and a red circular escape ROI. The numerical value of ”5” indicates the number of times the mouse crossed into the target quadrant (RB). When the central coordinates of the animal’s ROI (blue square) reaches the ROI for the platform (red circle), the escape latency will be calculated.

Upon completion of the video analysis, the algorithm produces a novel interface displaying the results of the analysis (Figure 5; Supplementary video 1). The results indicate the mouse’s chosen trajectory over the course of a 120-second experiment (Figure 5, left). A chart (Figure 5, center) indicates how often the mouse visited each quadrant. Another chart (Figure 5, right) shows the duration (in seconds) that the mouse stayed in each quadrant.

Statistical analysis was conducted to compare the data obtained from manual and automated analysis methods (Figure 6). The results indicated no significant difference between the two methods in terms of the duration of time that the mice spent in the target quadrant (t (7) = 1.837, p = 0.109).

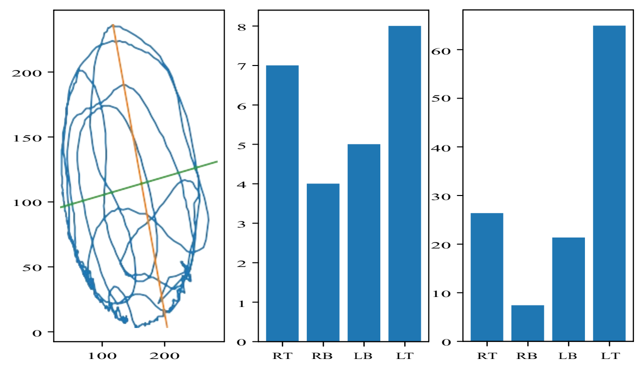

Figure 5: Capture a screenshot of the window displaying the analyzed data. The left graph depicts the mouse's pixel-based trajectory throughout the entirety of the testing period. The coordinates of the X and Y axes in video tracking have the reference point (0, 0) located in the upper left corner. Positive values of X indicate displacement to the right, while positive values of Y indicate displacement downwards, in relation to (0, 0). The first bar chart shows how frequently the mouse moved into the RT, RB, LB, and LT quadrants. The second bar chart illustrates the duration, measured in seconds, during which the mouse resided within each of the quadrants delineated in the experiment.

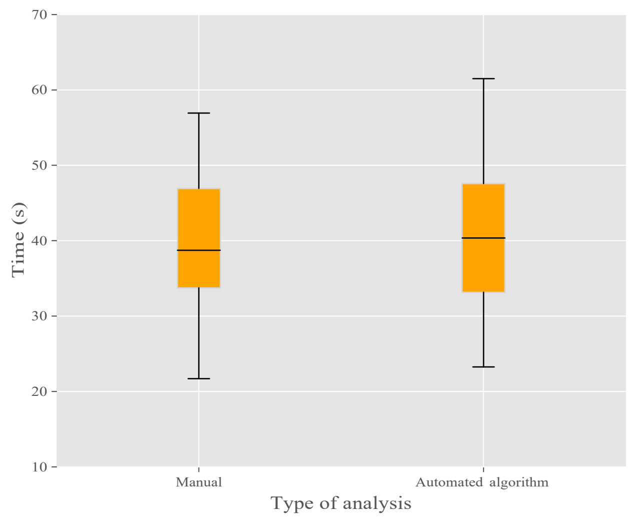

Figure 6: Boxplots were constructed to compare the results of manual analysis (left) and automated algorithm analysis (right) in relation to the duration of time that mice spent in the target quadrant. There was no difference between the manual and the automated analysis (p = 0.109, paired t-test). Data presented as median (line), 25–75% (box), min–max (error bars). Data are derived from n=8 B6126SF/J mice.

3.2. *Open Field test analysis*

Upon initiating the algorithm, the user is prompted with a response box, wherein they are required to select the specific test to be analyzed. The user must select option "2" to proceed with the OF test. Subsequently, it is imperative for the user to determine the animal's identity, specify the initial time of analysis in the video, and determine the overall duration of the analysis. Subsequently, a novel window launches wherein the user is required to designate the ROI that corresponds to the arena area. During the selection process, the user will have the ability to observe the four quadrants, as depicted in Figure 7. When an ROI is selected using the Python OpenCV library, it displays four squares with equal areas. However, the software selects the entire ROI area and divides it into 16 equal-sized quadrants. The four central squares that result from this division are referred to as the 'central region', while the remaining squares represent the 'edge area'. Upon completion of the selection process, the user must press either the "enter" or "space" keys.

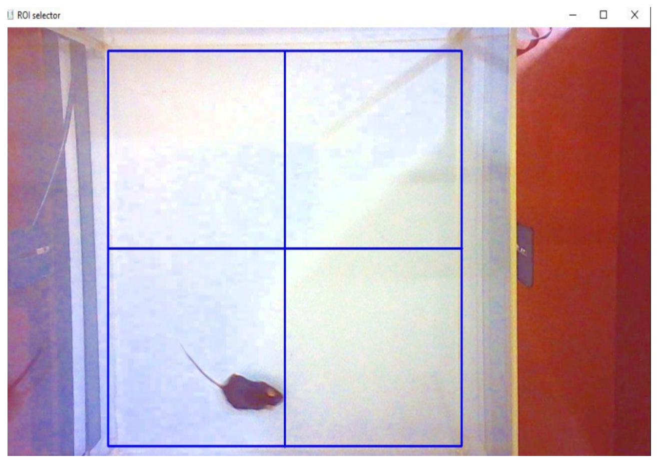

Figure 7: Screenshot of the first window that the algorithm opened during the OF test analysis. The user is instructed to choose the ROI that covers the entire area of the square arena. The user must click "Enter" to advance the algorithm's analysis after making the proper ROI selection.

The user should next choose the ROI that corresponds to the animal that will be monitored (Figure 8). The user should press "space" or "enter" once again.

Subsequently, the algorithm shall monitor the movement of the mouse throughout the entire duration of the examination. The measurements will provide information regarding the frequency of the animal's entry into both the central and edge areas of the arena. The illustration depicted in Figure 8 exemplifies a scenario in which a mouse traversed the edge of the arena

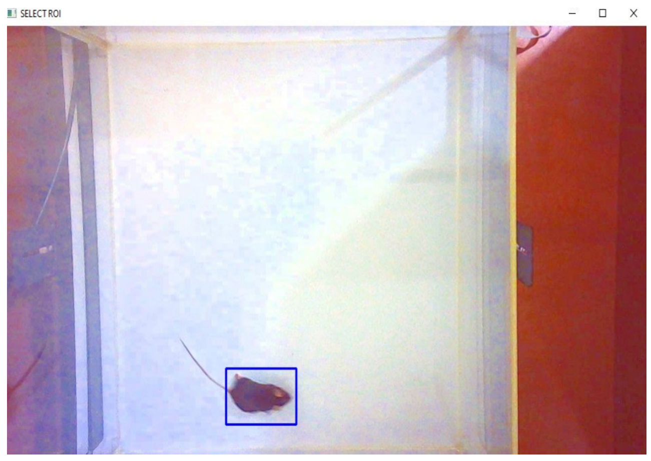

Figure 8: Screenshot of the second window that the algorithm opened. At this step, the user is told to choose an ROI that, ideally, should frame an image of the complete animal's body. The "Enter" key has to be pushed after the selection to start the algorithm's analysis.

on two separate occasions (i.e., "Edge2"). In the event that the animal exits the camera's field of view, the algorithm provides the user with the option to reselect the ROI. It is imperative to stress that in the event of any additional challenges encountered in animal tracking, the user may utilize the "R" key to reselect the ROI.

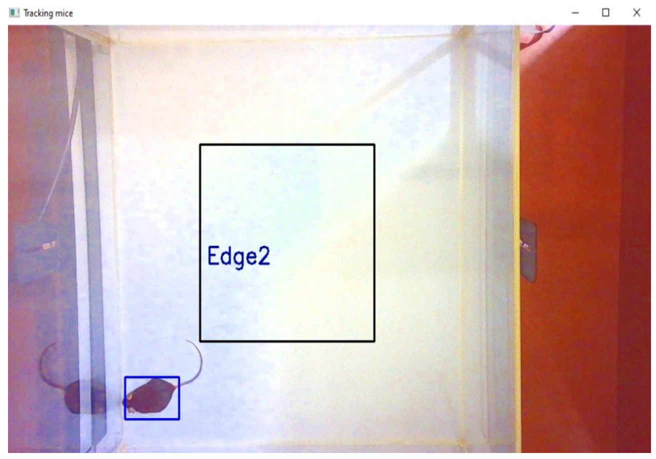

Figure 9: A visual representation of the algorithm's performance during the analysis of an OF test is depicted. A blue rectangle is utilized to track the movement of the mouse, while a numerical representation of the frequency of the mouse's traversal to the edges is concurrently exhibited. In this particular case, the mouse traversed the edge of the arena twice.

The software will map the center point of the animal's ROI to identify in which quadrant it is located. If the center point of the ROI is between two quadrants, the quadrant in which the animal was located will be considered for time and crossing counting.

Upon completion of the video analysis, the algorithm produces a novel interface that displays the outcomes of the analysis (Supplementary video 2). The results depict the path of the mouse throughout the experimental session (Figure 10, left). Additionally, the analysis shows how frequently the mouse enters both the center and the edge of the arena (Figure 10, center), as well as how long the mouse stays in each area (Figure 10, right).

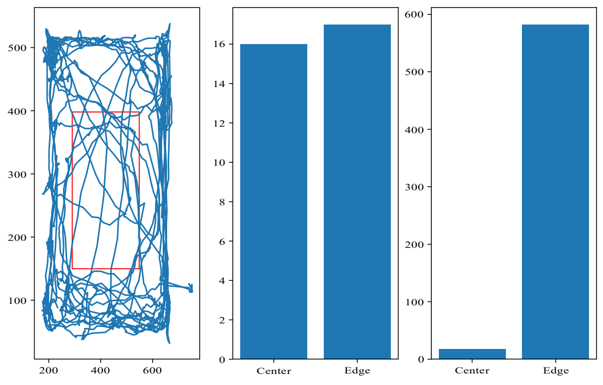

Figure 10: The output of data analysis. The animal's path (shown in pixels) during the course of the full test is shown in the left chart. The reference point (0, 0) for the X and Y axes in video tracking is situated in the top left corner. With respect to (0, 0), positive values of X and Y imply displacement to the right and downward, respectively. The chart in the middle displays the animal's frequency of entry into the arena's center and edges. The chart on the right indicates the duration of each entry in the center and edge regions of the arena.

Statistical analysis was conducted to compare the data obtained from both manual and automated analysis methods in terms of the time spent in the center (Figure 11) and on the edge (Figure 12) of the arena. For the manual analysis of the test, researchers divided the arena into 16 equally-sized quadrants, similarly to the automated analysis. The four central squares resulting from this division were referred to as the 'central region', while the remaining squares represented the 'edge area'. The results revealed no significant difference between the two methods in terms of the time that the mice spent in the center (t(9) = -0.670, p = 0.520, Figure 11) or on the edge (t(9) = 0.699, p = 0.503, Figure 12) of the arena.

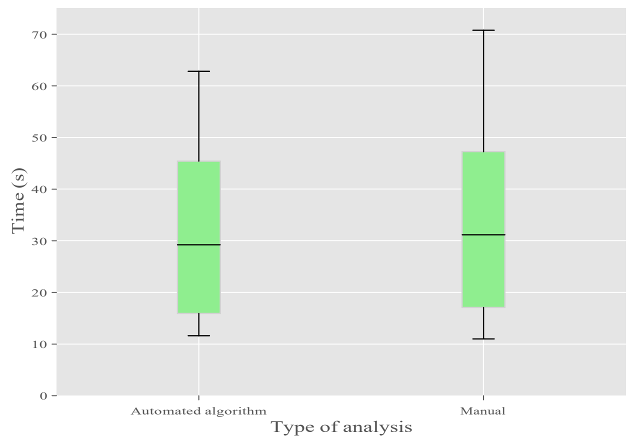

Figure 11: Two distinct methods of analysis are used to compare how long the animals stayed active at the center of the open field (OF). The results revealed that there was no statistically significant difference between the automated algorithm and the manual analysis (p = 0.520, paired t-test). Data presented as median (line), 25–75% (box), min–max (error bars). Data are derived from n=10 C57BL/6 mice.

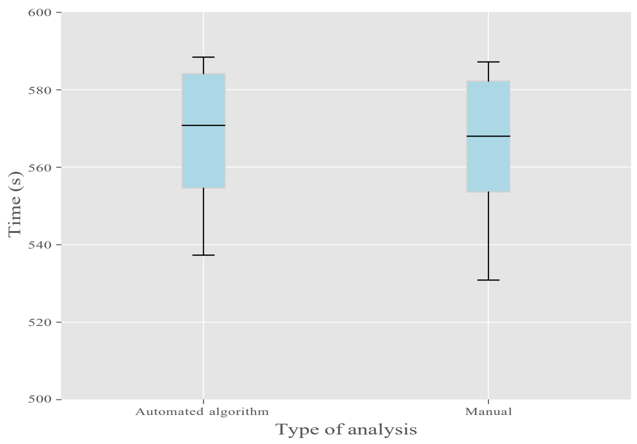

Figure 12: Two distinct methods of analysis are used to compare how long the animals stayed active at the edge of the open field (OF). The results revealed that there was no statistically significant difference between the automated algorithm and the manual analysis (p = 0.503, paired t-test). Data presented as median (line), 25–75% (box), min–max (error bars). Data are derived from n=10 C57BL/6 mice.

4. Discussion

Automated video analysis plays a vital role in various research fields like biology, engineering, and psychology. It provides precise and unbiased data on several parameters, including animal behavior. However, commercial software for video analysis can be expensive, application-specific, and proprietary, leading many researchers to develop their own algorithms using programming languages like Python or MATLAB. Although this approach can be more time-consuming, it offers researchers more flexibility and cost-effectiveness. Recognizing the importance of collaboration and open-source code, the scientific community has created libraries such as OpenCV, offering free and accessible tools for video analysis. This manuscript introduces an open-source algorithm, developed using OpenCV, that automatically interprets data from videos of animal behavior in the MWM test and the OF test. Our comparison between algorithmic and manual analyses revealed comparable results.

The MWM test is commonly used to assess rodents' cognitive capacity, particularly in spatial learning. Our program was able to track the animal in each quadrant of the video, calculate the time spent in each quadrant, and measure the escape latency. We found no significant difference between the results of our open-source algorithm and manual analysis, demonstrating the reliability and consistency of automated analysis. The OF test is another prevalent method for evaluating rodent behavior, including locomotor activity, anxiety levels, and exploratory behaviors. Our automated analysis using the open-source algorithm provided effective results when compared to manual analysis, with no significant difference in time spent in the center and edge areas of the OF.

Automated analysis offers significant advantages, such as the ability to process large amounts of data quickly and accurately, reducing the likelihood of human error. It facilitates the simultaneous exploration of multiple hypotheses, potentially leading to more comprehensive discoveries. Automated algorithms can potentially detect behavioral patterns and trends not immediately apparent to the human eye, uncovering crucial insights that manual methods might fail to notice. This efficiency can help lower costs and allow researchers to focus on strategic tasks [39, 40, 41].

The open-source software developed in this work has several advantages beyond being free. There are no restrictions on the recording device characteristics, and even smartphones are sufficient. Furthermore, unlike many proprietary software applications, which require precise, pre-determined device positioning, the video recording position is versatile and accepts videos taken from various angles. Also, video analysis using our software doesn't require high-performance machines or powerful hardware. Lastly, since the software is open-source and written in Python, it can be accessed and modified by anyone. This flexibility allows for the development of new methods that can be integrated separately into the main code, thereby adding functionalities not currently available in this version.

Nevertheless, it's important to note that this software has some limitations. For instance, the current version cannot differentiate between different regions of an animal's body, such as its head and tail. As a result, its capacity to analyze more intricate behaviors is somewhat restricted. Despite these limitations, the open-source software described in this study can be improved and extended with additional functionalities. Future enhancements might include quantifying the velocity and trajectory of an animal's movement, identifying distinct motion patterns, evaluating freezing behavior, and tracking specific actions like sniffing in a social interaction paradigm—functionalities commonly found in certain commercial software [32, 42, 43, 44, 45]. Our plan is to incorporate these capabilities into upcoming versions of the software. It is encouraged that researchers conduct follow-up studies to this manuscript that directly compare the results obtained using proprietary paid software to those obtained using the open-source software described herein.

5. Conclusion

To conclude, the algorithm described in this investigation represents a significant substitute for the manual analysis of animal behavior. Researchers from all over the world have easy access to the source codes and are able to modify and use them. Hence, the implementation of this automated analysis protocol will make a significant contribution to the progress of scientific research and facilitate the execution of more rigorous and accurate investigations in the domain of behavioral neuroscience.

References

[1] Siqueira, L. O., Ferrari, E. A. D. M., & Maia, J. M. (2006). Sistema automático de análise comportamental em pombos. Rev. bras. eng. biomed, 93-105.

[2] Houben, C., & Lapkin, A. A. (2015). Automatic discovery and optimization of chemical processes. Current opinion in chemical engineering, 9, 1-7.

[3] Crozara, M. G. N. (2017). Um sistema de código aberto para registro e análise de dados comportamentais categóricos, morfológicos e cinemáticos em animais de laboratório.

[4] Van Rossum, G. (2007, June). Python Programming Language. In USENIX annual technical conference (Vol. 41, No. 1, pp. 1-36).

[5] Lutz, M. (2001). Programming python. " O'Reilly Media, Inc.".

[6] Lutz, M. (2010). Programming python: powerful object-oriented programming. " O'Reilly Media, Inc.".

[7] Srinath, K. R. (2017). Python–the fastest growing programming language. International Research Journal of Engineering and Technology, 4(12), 354-357.

[8] Kadiyala, A., & Kumar, A. (2017). Applications of Python to evaluate environmental data science problems. Environmental Progress & Sustainable Energy, 36(6), 1580-1586.

[9] Hao, J., & Ho, T. K. (2019). Machine learning made easy: a review of scikit-learn package in python programming language. Journal of Educational and Behavioral Statistics, 44(3), 348-361.

[10] Welcome to Python.org. (2023, May 11). Retrieved May 17, 2023, from Python.org website: https://www.python.org/

[11] Community. (2023). Retrieved May 17, 2023, from Python.org website: https://www.python.org/community-landing/

[12] Python Software Foundation. Retrieved May 19,2023. Python 3.9.7 documentation: The Python Standard Library. Retrieved from https://docs.python.org/3/library/index.html

[13] Druzhkov, P. N., Erukhimov, V. L., Zolotykh, N. Y., Kozinov, E. A., Kustikova, V. D., Meerov, I. B., & Polovinkin, A. N. (2011). New object detection features in the OpenCV library. Pattern Recognition and Image Analysis, 21, 384-386.

[14] Farkhodov, K., Lee, S. H., & Kwon, K. R. (2020, February). Object Tracking using CSRT Tracker and RCNN. In BIOIMAGING (pp. 209-212).

[15] Guennouni, S., Ahaitouf, A., & Mansouri, A. (2014, October). Multiple object detection using OpenCV on an embedded platform. In 2014 Third IEEE International Colloquium in Information Science and Technology (CIST) (pp. 374-377). IEEE.

[16] Sobral, A. (2013, June). BGSLibrary: An opencv c++ background subtraction library. In IX Workshop de Visao Computacional (Vol. 27, p. 24).

[17] Xie, G., & Lu, W. (2013). Image edge detection based on opencv. International Journal of Electronics and Electrical Engineering, 1(2), 104-106.

[18] Kukreja, V. (2022, March). Segmentation and Contour Detection for handwritten mathematical expressions using OpenCV. In 2022 international conference on decision aid sciences and applications (DASA) (pp. 305-310). IEEE.

[19] Uke, N., & Thool, R. (2013). Moving vehicle detection for measuring traffic count using opencv. Journal of Automation and Control Engineering, 1(4).

[20] Kraeuter, AK., Guest, P.C., Sarnyai, Z. (2019). The Open Field Test for Measuring Locomotor Activity and Anxiety-Like Behavior. In: Guest, P. (eds) Pre-Clinical Models. Methods in Molecular Biology, vol 1916. Humana Press, New York, NY.

[21] Walsh, R. N., & Cummins, R. A. (1976). The open-field test: a critical review. Psychological bulletin, 83(3), 482.

[22] Lacerda, K. J. C. C. D. (2021). Caminhadas aleatórias com memória enviesada e suas aplicações em medicina e biologia (Doctoral dissertation, Universidade de São Paulo).

[23] Gould, T.D., Dao, D.T., Kovacsics, C.E. (2009). The Open Field Test. In: Gould, T. (eds) Mood and Anxiety Related Phenotypes in Mice. Neuromethods, vol 42. Humana Press, Totowa, NJ.

[24] Sejnowski, T. J., Koch, C., & Churchland, P. S. (1988). Computational neuroscience. Science, 241(4871), 1299-1306.

[25] Sejnowski, T. J., Churchland, P. S., & Movshon, J. A. (2014). Putting big data to good use in neuroscience. Nature neuroscience, 17(11), 1440-1441.

[26] Heindl, S., Gesierich, B., Benakis, C., Llovera, G., Duering, M., & Liesz, A. (2018). Automated morphological analysis of microglia after stroke. Frontiers in cellular neuroscience, 106.

[27] Morris, R. G., Garrud, P., Rawlins, J. A., & O'Keefe, J. (1982). Place navigation impaired in rats with hippocampal lesions. Nature, 297(5868), 681-683.

[28] Daniel, J. M., & Koebele, S. V. (2015). The maze book: theories, practice, and protocols for testing rodent cognition (pp. 411-419). H. A. Bimonte-Nelson (Ed.). Totowa, NJ, USA:: Humana Press.

[29] Bromley-Brits, K., Deng, Y., & Song, W. (2011). Morris water maze test for learning and memory deficits in Alzheimer's disease model mice. JoVE (Journal of Visualized Experiments), (53), e2920.

[30] Hort, J., Andel, R., Mokrisova, I., Gazova, I., Amlerova, J., Valis, M., ... & Laczó, J. (2014). Effect of donepezil in Alzheimer disease can be measured by a computerized human analog of the Morris water maze. Neurodegenerative Diseases, 13(2-3), 192-196.

[31] Webster, S. J., Bachstetter, A. D., Nelson, P. T., Schmitt, F. A., & Van Eldik, L. J. (2014). Using mice to model Alzheimer's dementia: an overview of the clinical disease and the preclinical behavioral changes in 10 mouse models. Frontiers in genetics, 5, 88. doi: 10.3389/fgene.2014.00088

[32] Zhang, L., Liu, C., Wu, J., Tao, J. J., Sui, X. L., Yao, Z. G., ... & Qin, C. (2014). Tubastatin A/ACY-1215 improves cognition in Alzheimer's disease transgenic mice. Journal of Alzheimer's disease, 41(4), 1193-1205.

[33] D'Hooge, R., & De Deyn, P. P. (2001). Applications of the Morris water maze in the study of learning and memory. Brain research reviews, 36(1), 60-90.

[34] Katz, R. J., Roth, K. A., & Carroll, B. J. (1981). Acute and chronic stress effects on open field activity in the rat: implications for a model of depression. Neuroscience & Biobehavioral Reviews, 5(2), 247-251.

[35] Vuralli, D., Wattiez, A. S., Russo, A. F., & Bolay, H. (2019). Behavioral and cognitive animal models in headache research. The journal of headache and pain, 20(1), 1-15.

[36] Hall, C. S. (1934). Emotional behavior in the rat. I. Defecation and urination as measures of individual differences in emotionality. Journal of Comparative Psychology, 18(3), 385–403. https://doi.org/10.1037/h0071444

[37] Walsh, R. N., & Cummins, R. A. (1976). The open-field test: A critical review. Psychological Bulletin, 83(3), 482–504. doi:10.1037/0033-2909.83.3.482

[38] Gould, T. D.; Dao, D. T.; Kovacsics, C. E. (2009). The open field test. Mood and anxiety related phenotypes in mice: Characterization using behavioral tests, p. 1-20

[39] Hong, W., Kennedy, A., Burgos-Artizzu, X. P., Zelikowsky, M., Navonne, S. G., Perona, P., & Anderson, D. J. (2015). Automated measurement of mouse social behaviors using depth sensing, video tracking, and machine learning. Proceedings of the National Academy of Sciences, 112, E5351–E5360

[40] Chao, R., Macía-Vázquez, G., Zalama, E., Gómez-García-Bermejo, J., & Perán, J.-R. (2015). Automated tracking of drosophila specimens. Sensors, 15, 19369–19392.

[41] Creton, R. (2009). Automated analysis of behavior in zebrafish larvae. Behavioural brain research, 203, 127–136

[42] Medlej, Y., Salah, H., Wadi, L., Saad, S., Bashir, B., Allam, J., ... & Obeid, M. (2019). Lestaurtinib (CEP-701) modulates the effects of early life hypoxic seizures on cognitive and emotional behaviors in immature rats. Epilepsy & Behavior, 92, 332-340.

[43] Salah, H., Abdel Rassoul, R., Medlej, Y., Asdikian, R., Hajjar, H.,

Dagher, S., ... & Obeid, M. (2021). A modified two-way active avoidance test for combined contextual and auditory instrumental conditioning. Frontiers in Behavioral Neuroscience, 15, 682927.

[44] Medlej, Y., Salah, H., Wadi, L., Saad, S., Asdikian, R., Karnib, N., ... & Obeid, M. (2019). Overview on emotional behavioral testing in rodent models of pediatric epilepsy. Psychiatric Disorders: Methods and Protocols, 345-367.

[45] Salah, H., Medlej, Y., Karnib, N., Darwish, N., Asdikian, R., Wehbe, S., ... & Obeid, M. (2019). Methods in emotional behavioral testing in immature epilepsy rodent models. Psychiatric Disorders: Methods and Protocols, 413-427.