**Automatic running wheel reward system – User manual**

This user manual aims to understand the concept of the automatic running wheel reward system used for rats created by Manus Pankow and to explain multiple bugs which occurred during the building process. In the end, every event (the pulses of the rotary encoder, when the speed threshold is attained, when the reward is delivered) can be seen in the open-ephys program, where the auxiliary channels input from the system could be combined with any kind of recording data. For the comparison between the behavior of the rat and the recording data, the following parts are needed:

Setup: For the general setup these components are used:

Running wheel (A): The running wheel with the build in rotary encoder was built in the Forschungswerkstatt. Inside the running wheel is a rotary encoder. In total there are 4 Pins of the rotary Encoder which u can see in the bottom of the website. (<https://www.pixelelectric.com/electronic-modules/miscellaneous-modules/hall-switches-keypads/lpd3806-600-pulse-rotary-encoder/?setCurrencyId=2>)

The following model is used: LPD3806-600BM-G5-24C (600 pulses per revolution) While using different rotary encoders, make sure to also change the number of PPR in the code!

The one big wire from the rotary encoder(A) in reality consists of four smaller wires which are connected like the following:

Ein Bild, das Maschine, Stromversorgung, Schaltung enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Diagramm, Screenshot, Plan, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

E

B

C

D3

D2

A

D1

Arduino (B): To connect the rotary encoder(A) with the Arduino UNO (B) in the bottom right/middle of the picture, exactly four wires are needed. Pin A & B (green & white wire) are connected to Pin 2 & 4 on the Arduino. In addition to that there is a red and a black wire from the rotary encoder connected to the 5V Pin & the ground Pin to the Arduino.

The Arduino is powered by the USB port from a laptop. If you’re using the acquisition board with a laptop that’s running off battery power, you will have a “floating” ground. This will cause your signals to look extremely noisy. Always use the power supply for the laptop!

Output Pins (C): ALL output Pins are connected to the I/O board, which is then connected via HDMI to HDMI (analog input) with the acquisition board.

Pin 11 & 12 are the corresponding output Pins of Pin A & B from the rotary encoder.

Pin 8 is an output Pin from Arduino for the speed threshold.

Pin 6 is an output Pin from Arduino for the valve. This output is decupled from the real reward system and is in the high state whenever a reward is delivered.

Everything is also labeled additionally on the outside of the box.

Valve pathway (D): From the output Pin 7 the whole valve pathway starts. After a certain time above the speed threshold, Pin 7 is set to high. This starts the following cascade. From Pin 7 a standard Arduino relay is activated (D1) because the DC/DC step up converter (D2) needs more current than the Arduino can output! The following model is used (<https://eckstein-shop.de/MT3608DC-DCAdjustableStep-upSpannungsreglerBoostVoltageRegulator2V-24V?ws_oss_lieferland=DE&gad_source=1&gclid=Cj0KCQiArby5BhCDARIsAIJvjIRxsw6ZDx8m1vQoDbd3y1Z2nadxy4mFv4IotMlFTY8imjczFmGuIQoaAifpEALw_wcB>) which turns 5V into 6V for the valve opening mechanism. For the step up converter to work properly the screw for voltage regulation needs to be turned in a specific way 15 times like it is described on the website above!

The output Pin (D3) on the top left connected with green wires to the DC/DC step up converter is not going to the open ephys board, but to the valve.

The valve opens precisely at 6V and has a response time of 30ms max. (<https://www.nresearch.com/Images/Valves/PDF/161T010.pdf>)

The tubing goes to a 3D printed Design. It is important that the bottom part is glued and perfectly sealed with superglue. Otherwise air bubbles get into the tube and the reward comes in irregularly.

Arduino Buzzer (E): There is also a buzzer implemented whenever the valve opens. Therefore Pin 10 is activated. It stays open until the valve stays open. Leave the box closed and put the buzzer inside for optimal noise level.

Open ephys: The version of open ephys is version 0.6.7. For the acquisition board the setup needs the ADC setting activated, DSP; 30.0 KS/s) after that source the LFP Viewer sink comes into place. At the end regarding the record node all of the files will be saved in a separate file (Manus Recordings) on data D:.

In the LFP viewer in the ADC channels the speed threshold is visible. One channel is for the valve opening and in 2 ADC channels are Pins A & B of the rotary encoder.

Code:

// Define the pulses per revolution (PPR) of your encoder

const int PPR = 600;  // Replace this with the actual PPR of your rotary encoder

const int stepsPerRevolution = PPR \* 4;  // 4 state changes per pulse (quadrature)

// Wheel circumference in centimeters (for speed calculation)

const float wheelCircumference = 60.0; // cm

// Variables to store the revolution count and previous states

int stepCount = 0;    // Tracks the number of steps in the current revolution

int totalStepCount = 0; // Tracks the total number of steps for speed calculation

int revCountClockwise = 0;     // Tracks the number of clockwise revolutions

int revCountCounterClockwise = 0;  // Tracks the number of counterclockwise revolutions

int totalRevolutions = 0;  // Tracks the total number of revolutions (independent of direction)

int prevInputState2 = HIGH;  // Stores the previous state of pin 2

int prevInputState4 = HIGH;  // Stores the previous state of pin 4

// Timer variables for speed calculation

unsigned long lastTime = 0; // Time when speed was last calculated

const int interval = 1000;   // Interval in milliseconds (1 second)

float speedRPM = 0;          // Variable to hold speed in RPM

float speedKmh = 0;          // Variable to hold speed in km/h

// Variables for reward system

const int valvePin = 7;        // Pin connected to the valve

unsigned long valveOpenTime = 0;    // Time when the valve was opened

const unsigned long valveDuration = 1500;  // Valve open duration (in milliseconds)

bool valveOpen = false;        // Flag to indicate if valve is currently open

// New addition: Pin 8 for Open Ephys I/O Board connection

const int ephysPin = 8;  // Pin connected to Open Ephys I/O board

const float speedThresholdKmh = 3.0;  // Speed threshold updated to 3 km/h

// New addition: Pin 6 to deliver voltage when the valve is open

const int indicatorPin = 6;  // Pin to provide a constant signal when the valve is open

const unsigned long indicatorMinDuration = 1000; // Minimum duration (1 second) for Pin 6 HIGH

unsigned long indicatorEndTime = 0; // Tracks when to turn off indicatorPin after valve closes

// New variables for timing logic

unsigned long speedAboveThresholdStartTime = 0;  // Start time when speed is above the threshold

unsigned long lastRewardTime = 0;  // Tracks the time when the last reward was given

const unsigned long requiredAboveThresholdTime = 10000;  // Required time above threshold to open valve (10 seconds)

const unsigned long rewardCooldown = 5000;  // Minimum time between rewards (5 seconds)

bool speedAboveThreshold = false;  // Tracks if the speed is above threshold

// New addition: Pin 10 for buzzer

const int buzzerPin = 10;  // Pin connected to the buzzer

void setup() {

  Serial.begin(9600);

  pinMode(2, INPUT\_PULLUP);

  pinMode(4, INPUT\_PULLUP);

  pinMode(12, OUTPUT);

  pinMode(11, OUTPUT);

  pinMode(valvePin, OUTPUT);

  pinMode(ephysPin, OUTPUT);

  pinMode(indicatorPin, OUTPUT);

  pinMode(buzzerPin, OUTPUT);  // Set buzzer pin as output

  digitalWrite(valvePin, LOW);

  digitalWrite(ephysPin, LOW);

  digitalWrite(indicatorPin, LOW);

  digitalWrite(buzzerPin, LOW);  // Ensure buzzer is off initially

  Serial.println("Initial Counts:");

  Serial.print("Clockwise Revolutions: ");

  Serial.println(revCountClockwise);

  Serial.print("Counterclockwise Revolutions: ");

  Serial.println(revCountCounterClockwise);

}

void loop() {

  int inputState2 = digitalRead(2);

  int inputState4 = digitalRead(4);

  if (inputState2 != prevInputState2 || inputState4 != prevInputState4) {

    if (prevInputState2 == HIGH && inputState2 == LOW) {

      if (inputState4 == HIGH) {

        stepCount++;

        totalStepCount++;

      } else {

        stepCount--;

        totalStepCount--;

      }

    }

    if (prevInputState4 == HIGH && inputState4 == LOW) {

      if (inputState2 == HIGH) {

        stepCount--;

        totalStepCount--;

      } else {

        stepCount++;

        totalStepCount++;

      }

    }

    if (stepCount >= stepsPerRevolution) {

      stepCount = 0;

      revCountClockwise++;

      totalRevolutions++;

    } else if (stepCount <= -stepsPerRevolution) {

      stepCount = 0;

      revCountCounterClockwise++;

      totalRevolutions++;

    }

  }

  prevInputState2 = inputState2;

  prevInputState4 = inputState4;

  // Calculate speed every second

  if (millis() - lastTime >= interval) {

    speedRPM = (abs(totalStepCount) / (float)stepsPerRevolution) \* 60;

    speedKmh = (speedRPM \* wheelCircumference) / 100000 \* 60;

    Serial.print("Speed (RPM): ");

    Serial.println(speedRPM);

    Serial.print("Speed (km/h): ");

    Serial.println(speedKmh);

    totalStepCount = 0;

    lastTime = millis();

    // Check if speed is above the threshold

    if (speedKmh >= speedThresholdKmh) {

      digitalWrite(ephysPin, HIGH);

      Serial.println("Speed threshold reached, Pin 8 HIGH.");

      // Start or maintain the timer if speed is above threshold

      if (!speedAboveThreshold) {

        speedAboveThresholdStartTime = millis();

        speedAboveThreshold = true;

      }

    } else {

      digitalWrite(ephysPin, LOW);

      Serial.println("Speed below threshold, Pin 8 LOW.");

      // Reset the timer if speed falls below threshold

      speedAboveThreshold = false;

      speedAboveThresholdStartTime = 0;

    }

  }

  // Check if the speed has been consistently above threshold for the required time

  if (speedAboveThreshold &&

      (millis() - speedAboveThresholdStartTime >= requiredAboveThresholdTime) &&

      !valveOpen &&

      (millis() - lastRewardTime >= rewardCooldown)) {

    digitalWrite(valvePin, HIGH);

    digitalWrite(indicatorPin, HIGH);

    digitalWrite(buzzerPin, HIGH);  // Turn on the buzzer

    valveOpenTime = millis();

    indicatorEndTime = millis() + indicatorMinDuration;

    valveOpen = true;

    lastRewardTime = millis();  // Update the last reward time

    Serial.println("Valve Opened! Reward given after sustained speed.");

    // Reset the threshold timer so next reward also requires 10 seconds

    speedAboveThresholdStartTime = millis();

  }

  // Close the valve after the duration has passed

  if (valveOpen && millis() - valveOpenTime >= valveDuration) {

    digitalWrite(valvePin, LOW);

    digitalWrite(buzzerPin, LOW);  // Turn off the buzzer

    valveOpen = false;

    Serial.println("Valve Closed.");

  }

  // Keep indicatorPin HIGH for at least 1 second after the valve opens

  if (!valveOpen && millis() >= indicatorEndTime) {

    digitalWrite(indicatorPin, LOW);

  }

  // Write to pin 12 based on the state of pin 2

  if (inputState2 == HIGH) {

    digitalWrite(12, HIGH);

  } else {

    digitalWrite(12, LOW);

  }

  // Write to pin 11 based on the state of pin 4

  if (inputState4 == HIGH) {

    digitalWrite(11, HIGH);

  } else {

    digitalWrite(11, LOW);

  }

}