CPEN 291
Project 2 Report

A. Group Info, project title and contribution summary

Group #: ___G14 - L2B_____

Bob Bao	Tony Li
Glen Qiao	Tianrui (Sandy) Chu

Project title: Automated pet(cat) feeding and remote interaction system

Contribution summary:

Bob: Implemented the WebServer and help with the hardware component

Tony: Implemented the Backend Server and Pico code

Glen: Implemented the hardware components(Feeder, Cleaner, Scales, Teaser) and helped beautify the website

Sandy: developed and implemented the webpage, including the main, setting and history page. Incorporated streaming functionality to webpage

B. Introduction and Description

-Features and functionalities:

1. **Feeder:**

   can be remotely controlled through the web interface

   has a hopper that can store a certain amount of pet food and

   equipped with a pressure sensor to detect the amount of food available

2. **Water feeder:**

   has a reservoir capable of containing a specific quantity of water

   can automatically refill itself

3. **Cat teaser**

   Provides interactive play with pets

   Can be controlled remotely through the web interface with a switch on the webpage

4. **Cat litter**

   Cleans pet waste from a designated area

   Can be controlled remotely through the web interface

5. **interactive WebPage**

   Include a streaming camera remotely monitors your pet

   Buttons to manually control the feeding, cleaning, playing actions of the hardware

   Setting page allows automatic feeding on scheduled feeding time with specific feeding amount

   A history page to show the latest five feeding cleaning records

-motivations and project evolution

The motivation behind selecting the topic for our project was to create a solution to help pet owners remotely monitor and interact with their pets. Our design objectives included creating a cost-effective system that could be easily assembled and customized by the user. We also wanted to create a system that could handle a range of functionalities, such as feeding and playing with pets, as well as monitoring their health and behavior.

In order to achieve our design objectives, we had to make trade-offs between different factors such as cost, complexity, and functionality. For example, we chose to use recycled materials to reduce costs, but this meant that we had to carefully design and test our equipment to ensure it was safe and effective. Additionally, we had to consider the technical constraints of the Raspberry Pi Pico, such as its limited memory and processing power, when designing our system.

Our project also had several open-ended aspects, such as the potential to expand its functionalities and to improve its usability through additional features. For example, we considered adding voice as well as email notifications, but ultimately rejected these features due to technical constraints and cost considerations. Overall, our project aimed to provide a low-cost and customizable solution for pet owners to remotely monitor and interact with their pets.

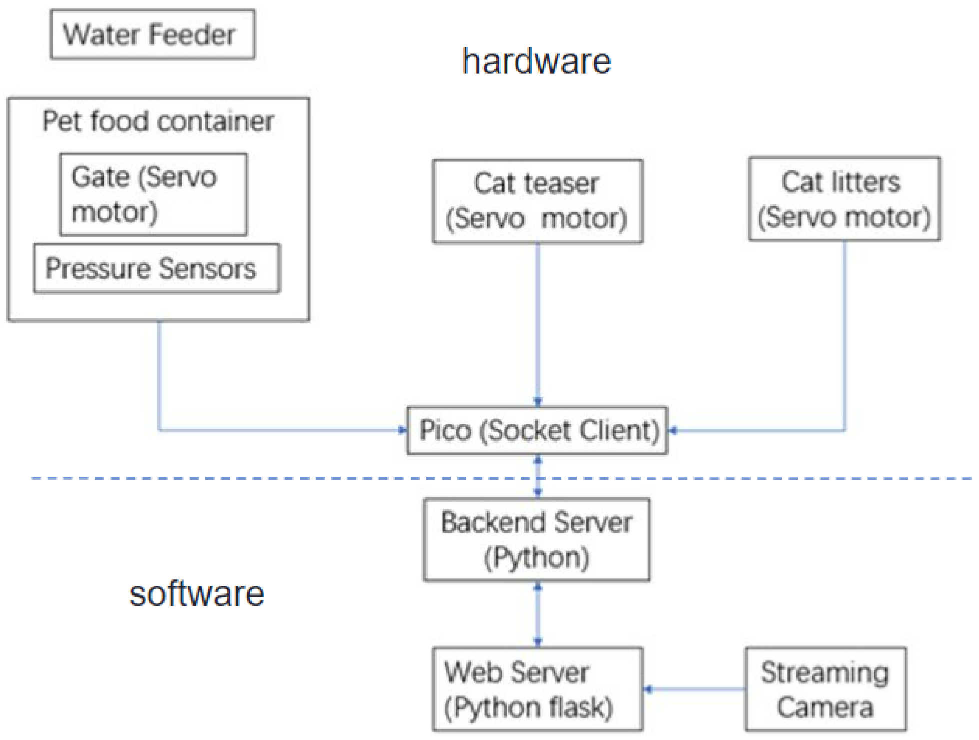
Our idea for the project came from our shared interest in pets and the desire to develop a system that could improve the pet ownership experience. We started with a general idea of creating a device that would allow pet owners to monitor and interact with their pets remotely. From there, we brainstormed different features and functionalities that we thought would be useful and feasible for us to develop with our available resources.

As we continued to develop the project, our ideas evolved and became more focused. We conducted research and consulted with some friends who are raising pets to determine the most important aspects of pet care that we could address with our system. This led us to prioritize features such as the ability to dispense food and water, monitor temperature and humidity, and provide a platform for remote play and interaction.

Throughout the development process, we also encountered challenges and limitations that forced us to adjust and refine our ideas. For example, we had to consider the cost and availability of materials, the technical capabilities of our hardware and software, and the complexity of integrating different functionalities into a cohesive system.

Overall, our idea for the project evolved from a general concept to a more focused and refined system as we gained a better understanding of the needs of pet owners and the technical constraints of the project.

C. Block diagrams



D. Technical documentation for the software [web app]

What technologies are used for the front-end
-HTML, CSS, javascript, Bootstrap framework, jquery

What technologies are used for the back-end
-python flask, socket

The front-end design and implementation

Our team has developed a system for cats that automates feeding, litter box, and remote interaction. Users can set the feeding times and amount, and view feeding and cleaning records. To create the user interface, we utilized HTML, CSS, and JavaScript to create multiple websites that allow users to navigate to their desired webpage. The main page has a header with buttons for settings and history that lead to subpages. A streaming camera window is also available for users to monitor their cleaning cat. Control buttons for scheduling, feeding, cleaning, and playing, with a switch next to the 'play' button, are located below the feeding schedule next to the streaming window. Users can enter their user information at the bottom of the main page to receive email notifications at specified times. However, Due to time limitations, we have not yet implemented the function to use a database for notifications. To set the feeding time and amount, users can click the 'setting' button on the upper-left corner of the main page and jump to the setting page, where they can add up to three-time points per day, delete them, and adjust the food amount before submitting. The setting page then jumps back to the main page. Additionally, users can click the 'history' button on the upper-right corner of the main page to view the recent five feeding and cleaning histories on the history page. The header buttons on the subpages allow users to navigate between the setting and main pages.

The back-end design and implementation

For the back-end part, we first used Apache and then switched to python flask because we think python flask can provide us with more functionality. We created a web server using python flask on the provided virtual machine to feed the HTML file to the IP address at port 80. We implemented a Flask web application that communicates with a socket server. The web application has three routes: "/" for the main page, "/setting" for the setting page and "/history" for the history page.

The main route allows the user to interact with the pet feeder by pressing buttons to feed, clean, or play with the pet. When the user presses the "schedule" button, a message is sent to the socket server requesting the schedule information, which is then displayed in a table on the web page.

The setting route allows the user to set a feeding schedule for the pet feeder. When the user submits the schedule information, it is sent to the socket server.

The history route allows the user to view the feeding and cleaning history of the pet feeder. A message is sent to the socket server requesting the history, which is then displayed in a table on the web page.

E. Technical documentation for the software [excluding the web app]

Communication between IoT and the back-end and any other device that uses the Internet for communication

To build a connection from hardware to the web app, we implemented two Python socket servers each connected with pico or web server. The server connecting to Pico listens on port 443 while the other one binds its address on port 777. As two separate threads running on a single process, these two backend servers shared the same data source, allowing it to function as a database with decision-making logic and most importantly an information transfer center.

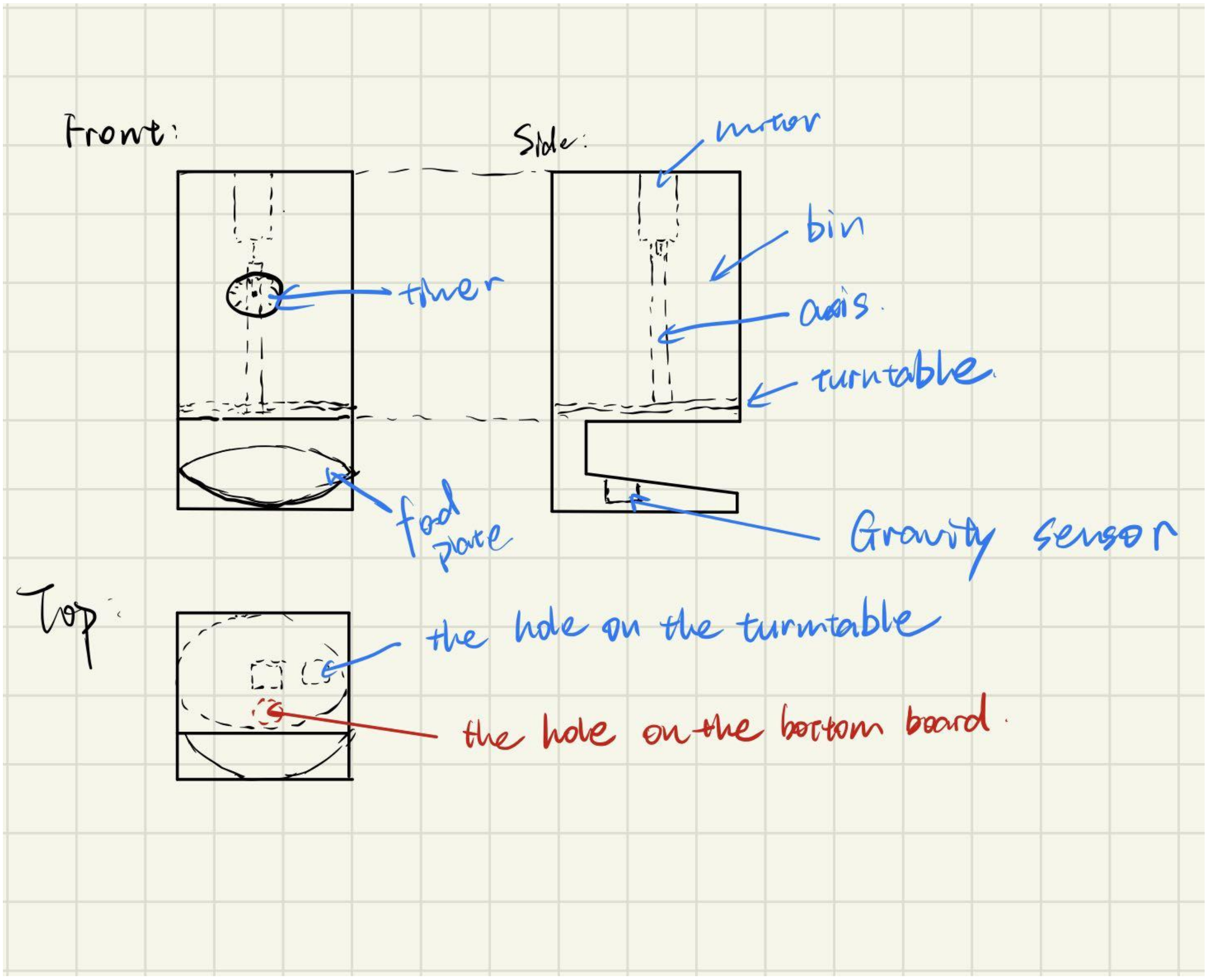
We used Raspberry Pi Pico W as the controller for all the hardware; it constantly sends messages to the backend server updating its sensor readings and waiting for new instructions. Then the backend server will determine the proper action to perform based on the data collected from both the hardware side and the web page input. When an actual action is sent and executed, the server also records the action with its action time for future usage.

List all libraries used (the libraries for the hardware interface are listed in section F)

Python: socket, threading, time

F. Technical documentation for the hardware/IoT

List all components, sensors and actuators used in the product
Pico-w, Three servo motors, Three pressure sensors, Resistors, Plates, Plastic bottle, Paper board, Disposable chopsticks, Styrofoam, Wire, Yarn
List any libraries used for the added components
Circuit Python: wifi, socket pool, time, os, board, analogio, pwmio, servo
The design and implementation of the circuitry



G. **Test and evaluations**

Evaluation and Testing:

We conducted several rounds of testing to ensure that our project meets the requirements and functions as intended. This included both hardware and software testing.

Hardware Testing:

We tested the hardware components of our project, such as the Pico, sensors, and servo motors, to ensure they were working properly. We also tested the reliability and stability of our custom PCB design.

Software Testing:

We conducted extensive testing of our software components, such as the web server, backend server, and Pico firmware. We developed test cases and automated scripts to ensure the proper functionality of the system.

Challenges:

During the development and testing phase, we encountered several challenges that required us to find solutions and best practices.

Communication between server and client: We found that when our web server and the backend server communicate on the same port at the same time, one thread would block the other. We resolved this issue by building two-socket servers, one for communication between the web server and backend server on port 777, and the original one on port 443 for communication with the Pico.

Building the server: Initially, we tried to use Apache on the virtual machine to build the web server, but it was challenging to connect it directly to our Pico. We resolved this issue by building our socket server and webserver using Python and Flask.

Integration of sensors and Pico firmware: We faced challenges in integrating the sensors and Pico firmware into the backend server. We resolved this issue by developing custom firmware for the Pico and creating a message protocol to communicate between the backend server and Pico.

Test Cases and Results:

We developed several test cases to ensure the proper functioning of our system. This included testing the integration of sensors with the Pico, communication between the web server and backend server, and the stability and reliability of our custom PCB design. We also conducted stress testing to evaluate the system's performance under heavy load.

Overall, our testing confirmed that our system met the requirements and functioned as intended. We resolved all the challenges we encountered by finding solutions and best practices, which helped us improve the overall performance and reliability of our system.

H. Conclusions and Reflections

One of the most important lessons in project management is the need for effective communication. This includes both regular updates on project status and the ability to communicate effectively when issues arise. In addition, it's important for team members to have a clear understanding of their roles and responsibilities and to be willing to collaborate and share knowledge.

Time management is another key factor in project success. Team members should have a clear understanding of deadlines and priorities, and should be able to manage their time effectively to ensure that tasks are completed on schedule. It's also important for team members to be flexible and adaptable, and to be willing to adjust priorities and schedules as needed to ensure project success.

In conclusion, the tricks or interesting concepts that can be learned from successful projects include effective communication, time management, and teamwork. By implementing these concepts and lessons learned, teams can achieve their goals and deliver successful projects.

I. References and bibliography

Socket Programming in Python (Guide) – Real Python

[W3school.com]

[Create a Python Web Server - Python Tutorial (pythonbasics.org)]

[https://getbootstrap.com/]

Files submitted:

File Name	**Description**
VM backend server.py	the backend server on a virtual machine
webServer.py	the webserver file on a virtual machine
code.py	the control file on Pico
index.html	the main web page of our project
history.html	the subpage about checking the history
settingpage.html	the subpage about setting the feeding time and amount
Project2.fzz	the Fritzer file of our circuit

J. Answer the following questions:

Q1 - Design Process for some core functionalities: Clearly describe the process you have used for the following design aspects of some core component(s) of your project (for example, Web or embedded system, …).

Please spend time discussing and carefully answering each of them.

1. **Use of process**: Describe your approach to adapt and apply a general design process for the core features in your project. What was your approach?

   Our approach to adapting and applying a general design process for the core features in our project was based on the following steps:
   - Define the problem statement and project objectives.
   - Conduct research and analysis to identify potential solutions and constraints.
   - Brainstorm and generate ideas for the features and functionalities.
   - Evaluate and prioritize the ideas based on technical feasibility, resources, and impact on project objectives.

- Develop and prototype the selected features.
- Test and refine the features based on our actual use.

2. **Constraint identification**: Explain the constraints that you must consider in the design of those features/functionalities.

    In our project, we had to consider several constraints during the design process. One of the main constraints was cost. We had a limited budget to work with, so we had to be mindful of our spending and find ways to reduce costs wherever possible. To address this constraint, we decided to reuse materials from Project 1 where possible, such as scrap wood and metal, and we also used recycled bottles and boxes to make some of the equipment. This not only reduced costs but also helped to minimize waste.

    Another constraint we had to consider was the physical size of the equipment. We had limited space in the lab, so we had to design the equipment to be compact and space-efficient while still meeting the functional requirements.

3. **Solution generation**: Explain at least two possible alternative features that your group rejected due to technical reasons and explain why.

    Two possible alternative features that we rejected due to technical reasons were voice recognition and facial recognition. We initially considered incorporating voice recognition technology to allow users to get the situation of what their pet wants to do. However, we found that this technology required a significant amount of processing power and memory, which would be too much for the Raspberry Pi Pico to handle. Another feature that we rejected due to technical constraints was the ability to give notifications through email. Although it would be convenient for users, it would require the creation of a database to store user information and their email addresses, which would add significant complexity to the system. Additionally, it would require additional software and resources to send out notifications, which would increase the overall cost of the project. Therefore, we decided to focus on the core features of the project and prioritize simplicity and cost-effectiveness.

4. **Solution Assessment**: Explain how you tested and assessed the viability and then the correctness of your group's selected features.

    We tested and assessed the viability and correctness of our selected features through several methods, including manual testing, unit testing, integration testing, and user testing. We developed test cases for each feature and tested them thoroughly to ensure that they met the requirements and performed as expected. Based on the testing, we refined the features and made necessary adjustments to ensure that they were viable and correct.

Q2 – Teamwork: For this project, explain in detail the methods your group has used to communicate effectively among team members.

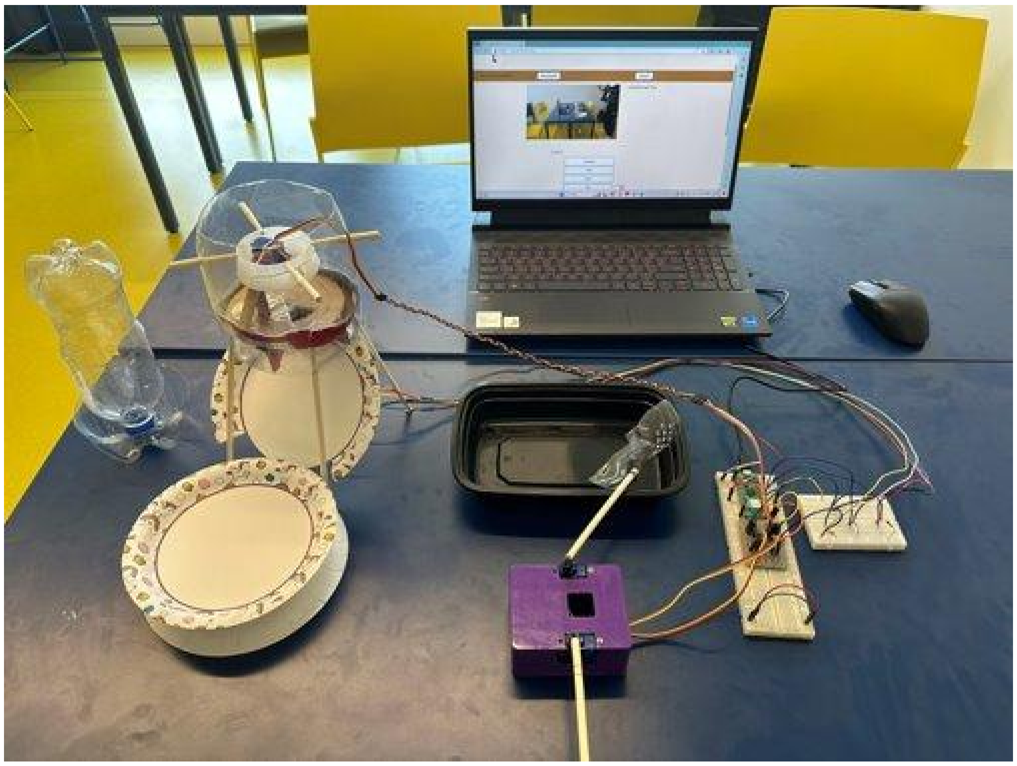
Regular Meetings: We hold regular meetings to discuss progress, set goals, and identify any issues that need to be addressed. These meetings are held in person. The meeting time for our group is every Wednesday in the Macleod building to check the progress of each member and then set a new target for the following week.

Clear and Concise Communication: the messages should be clear and concise when discussing the details of how to build each component. Our team members are encouraged to ask questions if we don't understand something and to repeat back instructions to ensure clarity.

Feedback and Criticism: We encourage open and honest feedback and criticism, and our team members should be willing to receive constructive feedback without becoming defensive.

Respectful Communication: We always communicate respectfully. Disrespectful communication will create a toxic environment and lead to conflicts that can harm our productivity and morale.

Appendix A – Project pictures (hardware)



-

Figure 1 - Overview of all components

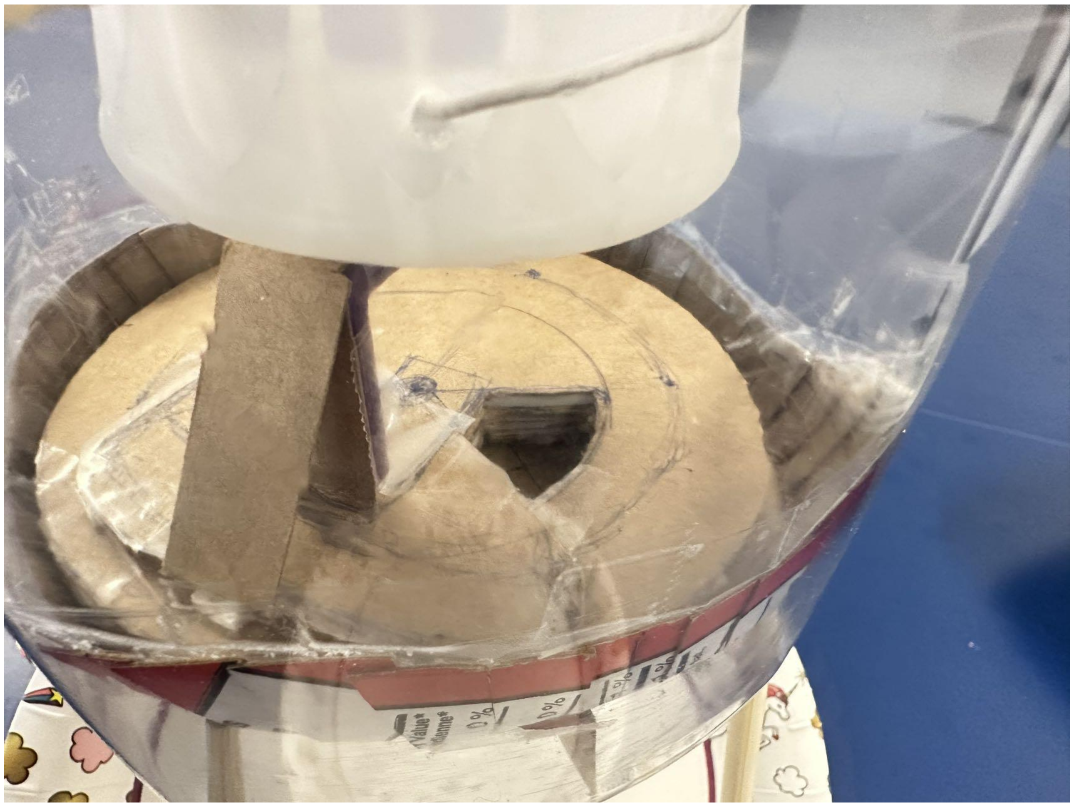


Figure 2: Feeder (1) — Status: Closed

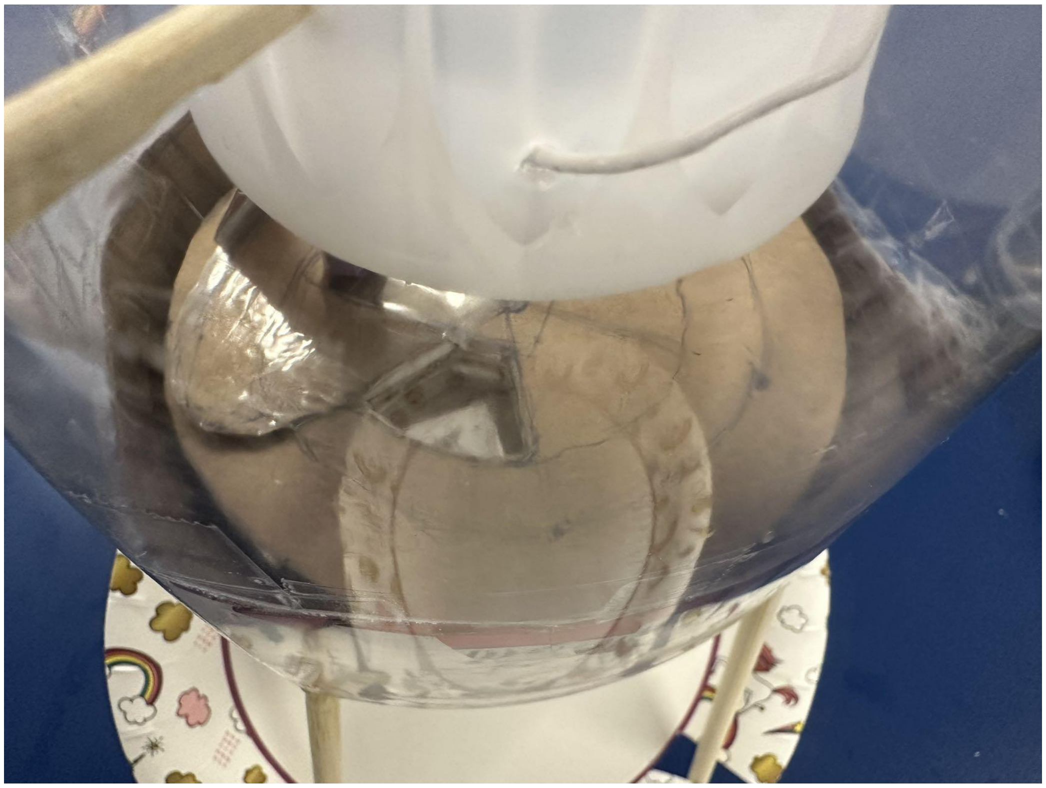


Figure 3: Feeder(2) — Status: Open

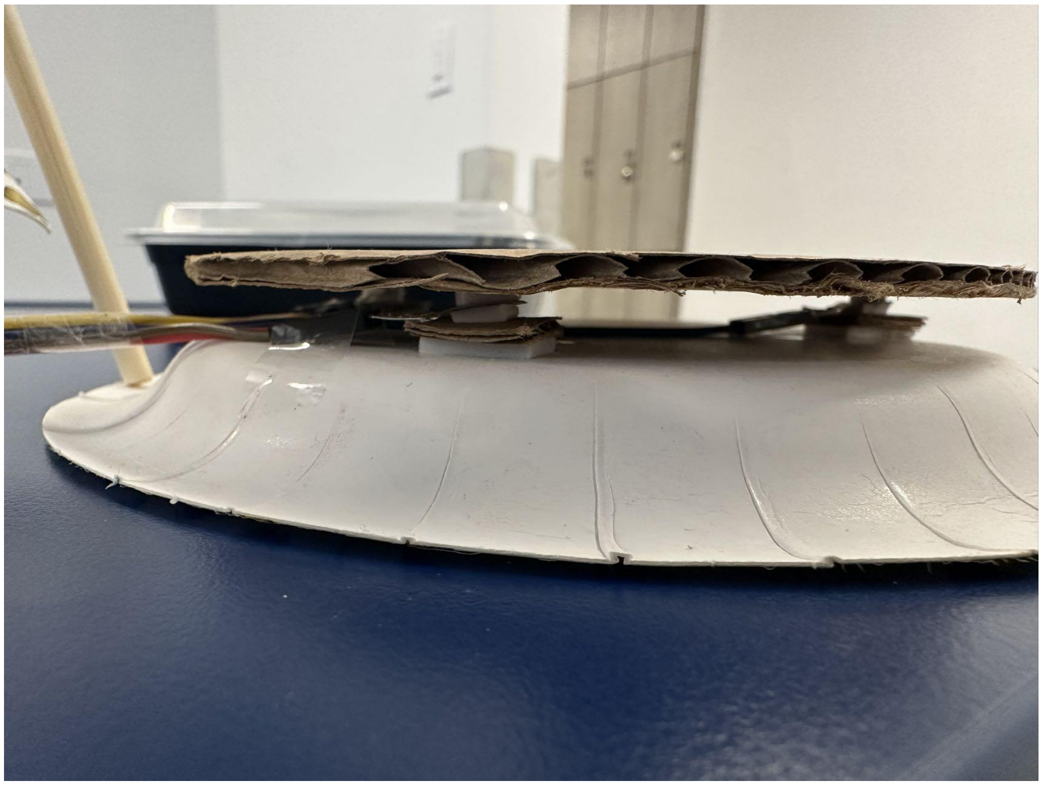


Figure4: Scale with three pressure sensors

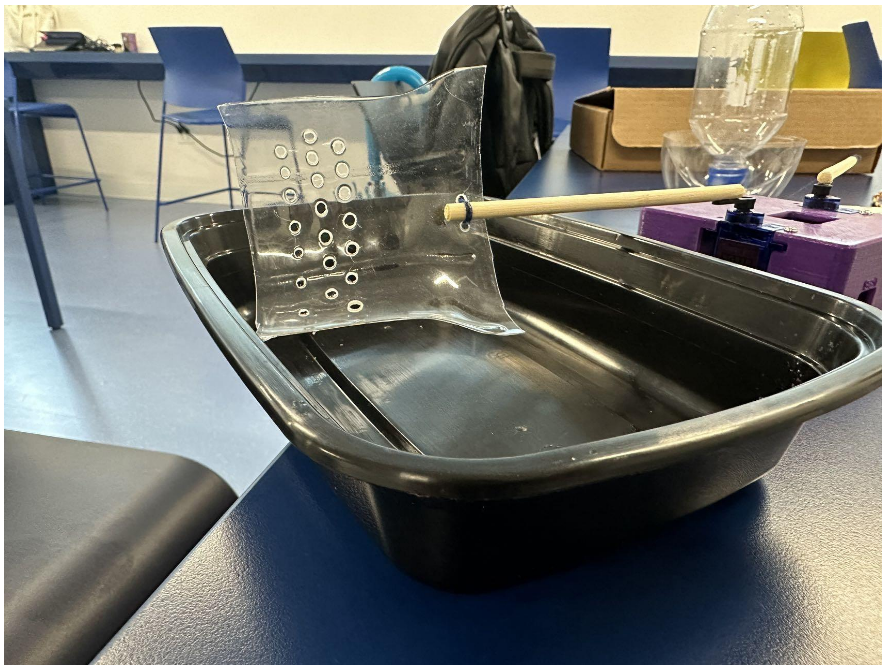


Figure5: Excrement Cleaner

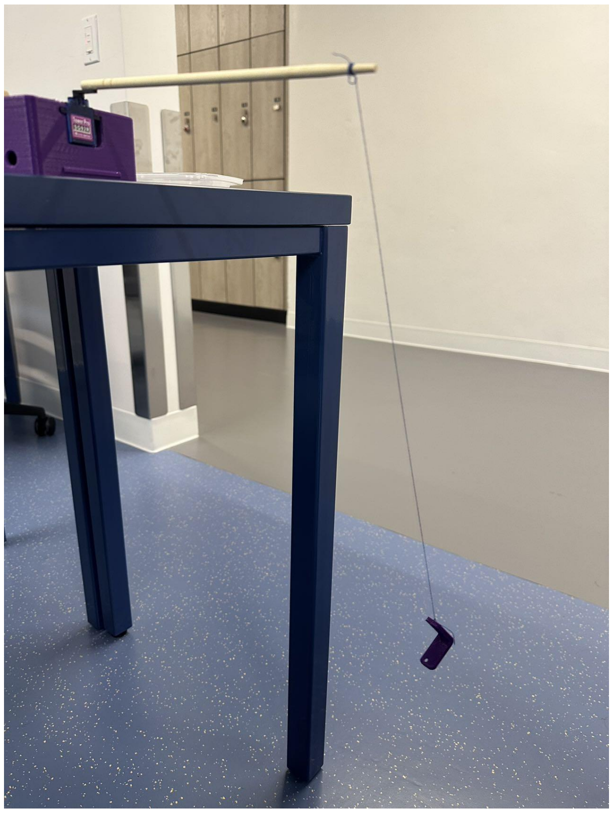
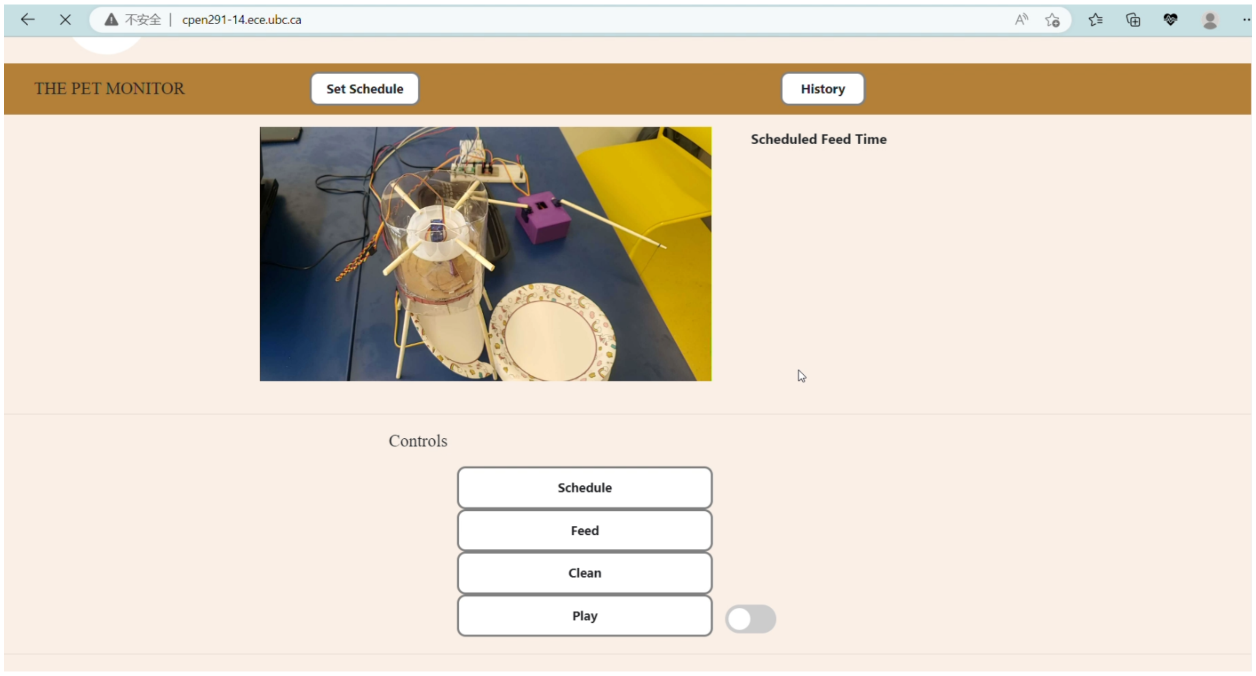
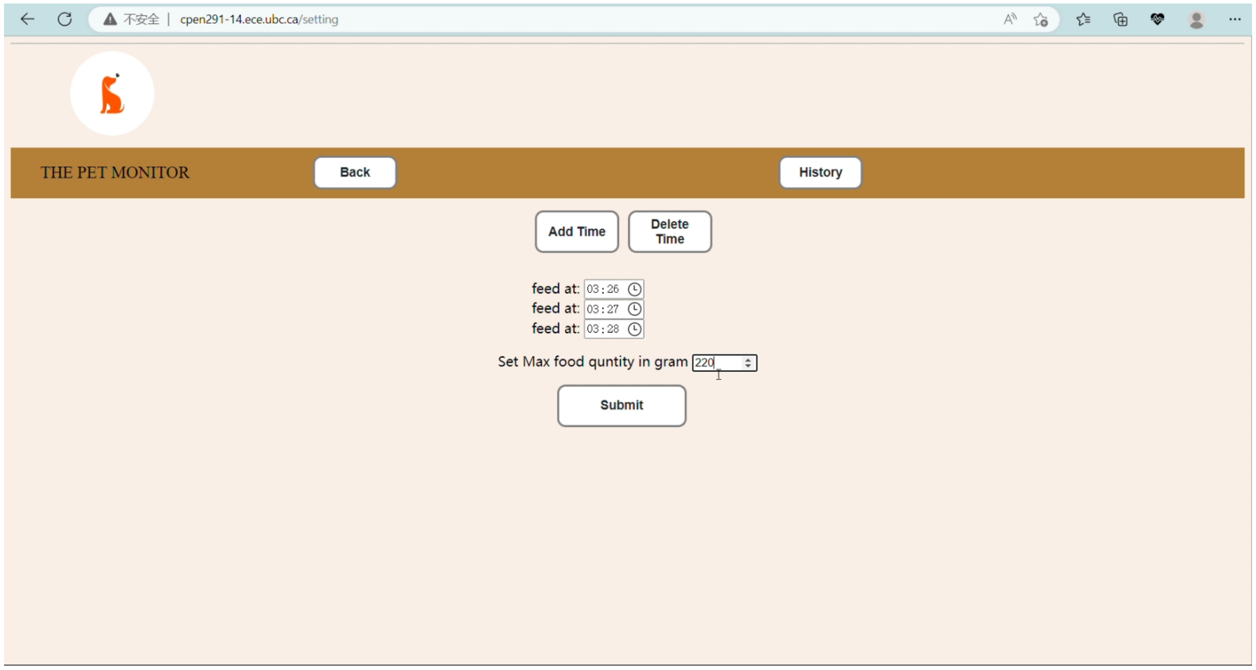


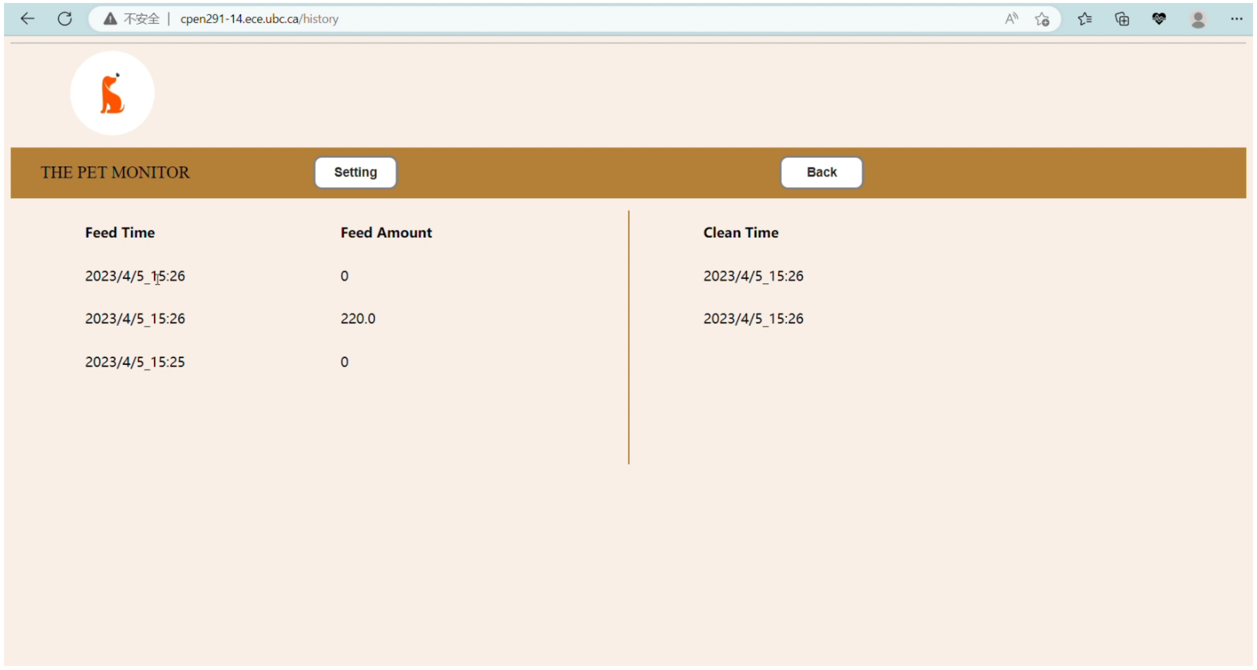
Figure6: Cat Teaser

Appendix B – Project pictures (software)





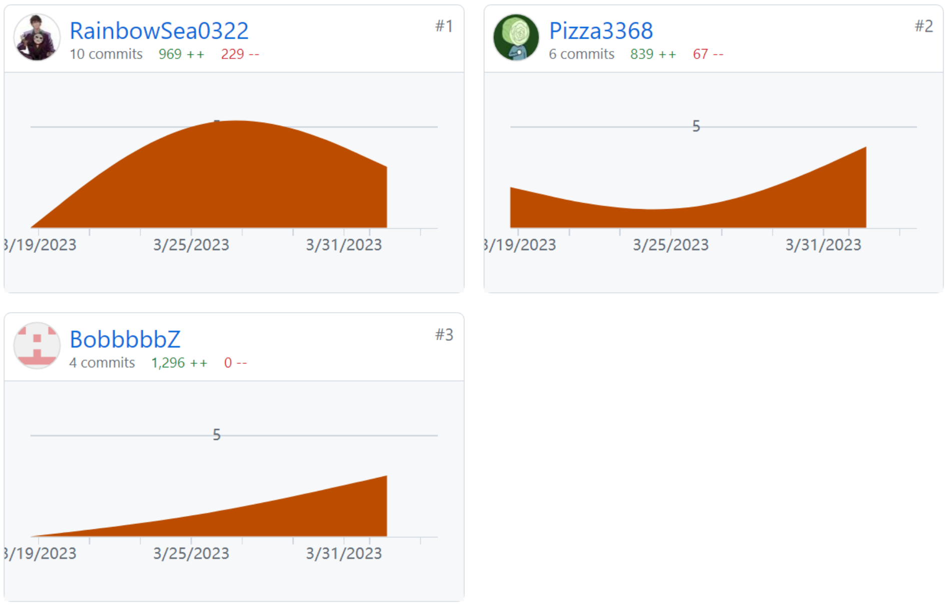
feed at: 03 : 26
feed at: 03 : 27
feed at: 03 : 28
Set Max food quntity in gram 220



Feed Time	Feed Amount	Clean Time
2023/4/5_15:26	0	2023/4/5_15:26
2023/4/5_15:26	220.0	2023/4/5_15:26
2023/4/5_15:25	0	

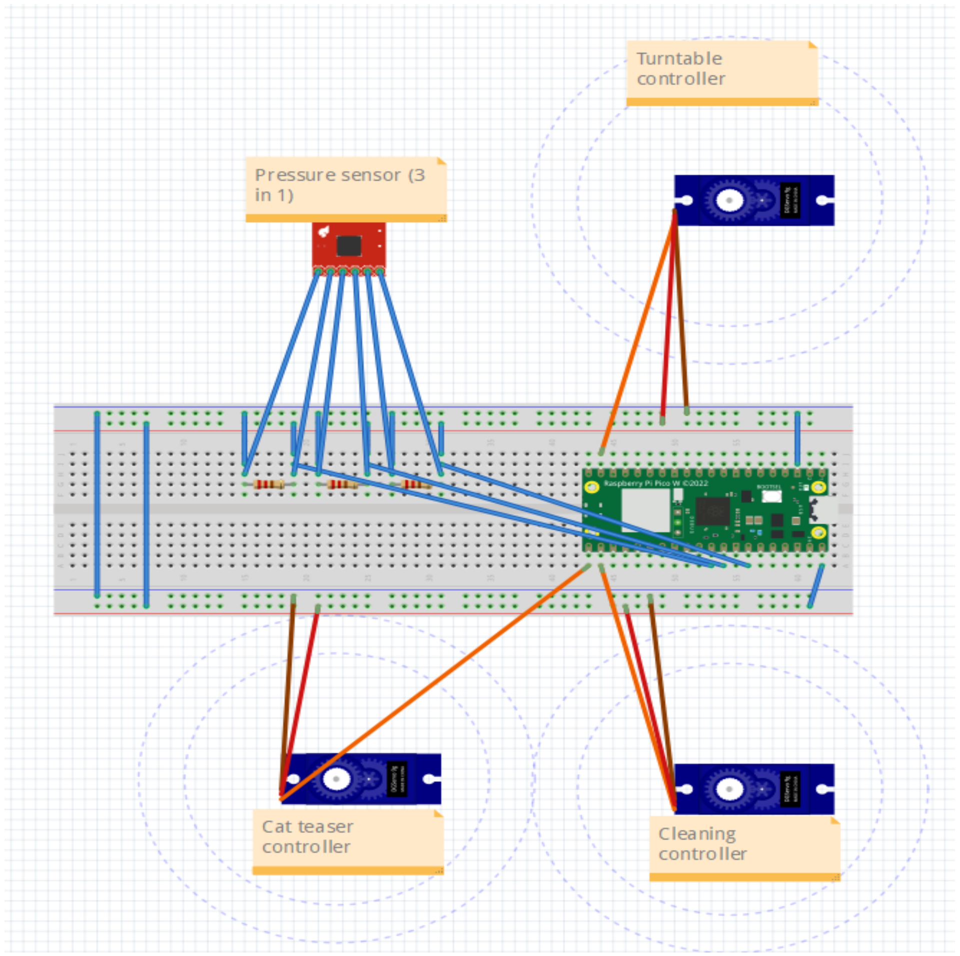
Appendix C - Fritzing

Appendix D - GitHub/Version Control


RainbowSea0322 #1
10 commits 969 ++ 229 --

Pizza3368 #2
6 commits 839 ++ 67 --

BobbbbbZ #3
4 commits 1,296 ++ 0 --



Despite being responsible for the hardware aspect of our project, Glen did not actively participate in the GitHub version control. However, he has provided valuable suggestions during the development of the front-end portion.

Appendix E – Component list

Include the complete list of the components/parts used for the project. For each component, including the part #, a short description and the quantity used in the project. As an example:

Part #	Description	Quantity used
Micro servo	Servo Motor	3
RP-C10-ST	Pressure Sensor Module	3
Pico-W (RASPBERRY PI)	Single Board Microcontroller	1
Breadboard	Circuit component	2

Appendix F – Hardware-related Code

```
# hardware Pico code

# code.py

import wifi

import socketpool

import time

import os

import board

import analogio

import pwmio

from adafruit_motor import servo


# edit host and port to match server

HEADER = 64

PORT = 443

HOST = "10.93.48.142"

ADDR = (HOST, PORT)
```

```
FORMAT = "utf-8"

INTERVAL = 5


print("Connecting to wifi")

wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
os.getenv('CIRCUITPY_WIFI_PASSWORD'))

print("Create TCP Client Socket")

pool = socketpool.SocketPool(wifi.radio)

socket = pool.socket(pool.AF_INET, pool.SOCK_STREAM)

print("Connecting to VM Server")

socket.connect(ADDR)


# sending message to server with appropriate protocol header

def send(msg):

    message = msg.encode(FORMAT)

    # fixed length header before actual message

    msg_length = len(message)

    send_length = str(msg_length).encode(FORMAT)

    send_length += b' ' * (HEADER - len(send_length))

    socket.send(send_length)

    socket.send(message)


# create PWMOut objects

pwmFD = pwmio.PWMOut(board.GP17, duty_cycle=2 ** 15, frequency=70) # feed servo

pwmPL = pwmio.PWMOut(board.GP16, duty_cycle=2 ** 15, frequency=70) # play servo

pwmCL = pwmio.PWMOut(board.GP15, duty_cycle=2 ** 15, frequency=70) # clean servo

# Create servo objects, my_servo.
```

```
servoFD = servo.Servo(pwmFD)

servoFD.angle = 60

servoPL = servo.Servo(pwmPL)

servoPL.angle = 60

servoCL = servo.Servo(pwmCL)

servoCL.angle = 60


# Initialize analog input connected to weight sensor

pressureSensor1 = analogio.AnalogIn(board.GP26)

pressureSensor2 = analogio.AnalogIn(board.GP27)

pressureSensor3 = analogio.AnalogIn(board.GP28)


# action array for tasks

cleanArr = [60, 105, 150, 60]

playArr = [60, 150, 90, 180, 90, 150, 60]


# signals for servo control

doFeed = False

doClean = False

doPlay = False


lastCleanTime = time.monotonic()

lastPlayTime = time.monotonic()


cleanIndex = 0

playIndex = 0
```

```
maxFeedAmount = 0.0


SERVO_INTERVAL = 1


# transfer analog input to actually meaningful weight in gram
def calc_weight(sensorIn1, sensorIn2, sensorIn3):
    w1 = sensorIn1.value / 1000 * sensorIn1.reference_voltage - 20
    w2 = sensorIn2.value / 1000 * sensorIn2.reference_voltage - 20
    w3 = sensorIn3.value / 1000 * sensorIn3.reference_voltage - 20
    w = w1 + w2 + w3
    if w < 0:
        w = 0
    return w


# open the gate for food container until weight reached the set value
def feed():
    global doFeed
    global maxFeedAmount
    if doFeed:
        w = calc_weight(pressureSensor1, pressureSensor2, pressureSensor3)
        print(f"in feed weight: {w}; max: {maxFeedAmount}")
        if  w < maxFeedAmount:
            servoFD.angle = 180
        else:
            servoFD.angle = 90
            doFeed = False
```

```
# swipe the feces of pet to one side
def clean():
    now = time.monotonic()
    global doClean
    global cleanIndex
    global lastCleanTime
    if doClean:
        if (now - lastCleanTime >= SERVO_INTERVAL):
            lastCleanTime = now
            servoCL.angle = cleanArr[cleanIndex]
            cleanIndex = cleanIndex + 1
            if cleanIndex >= len(cleanArr):
                cleanIndex = 0
                doClean = False
# keep moving the teaser until told to stop
def play():
    now = time.monotonic()
    global doPlay
    global playIndex
    global lastPlayTime
    global lastPlayTime
    if doPlay:
        if (now - lastPlayTime >= SERVO_INTERVAL):
            lastPlayTime = now
            servoPL.angle = playArr[playIndex]
            playIndex = playIndex + 1
            if playIndex >= len(playArr):
```

```
            playIndex = playIndex % len(playArr)

# differentiate different command from server and making appropriate variable change

def handle_instruction(inst_list):

    global doFeed

    global doPlay

    global doClean

    global maxFeedAmount

    command = inst_list[0]

    if command == "feed":

        doFeed = True

        maxFeedAmount = float(inst_list[1])

    elif command == "play":

        # switch doPlay according to input

        if inst_list[1] == "on":

            doPlay = True

        elif inst_list[1] == "off":

            doPlay = False

    elif command == "clean":

        doClean = True

    elif command == "None":

        pass


# time related variables

last_instr_time = time.monotonic()


while True:

    now = time.monotonic()
    try:
    #  every 5 seconds, ping server & update weight reading and ask for instruction
        if now > (last_instr_time + INTERVAL):
            last_instr_time = now
            weight = calc_weight(pressureSensor1, pressureSensor2, pressureSensor3)
            print(f"Weight: {weight}")
            send("pico weight " + str(weight))
            send("pico instruction")
            # get instructions
            instruction = receive_reply(socket)
            print(instruction)
            inst_list = instruction.split()
            handle_instruction(inst_list)



    except Exception as e:
        print(e)
        continue
    # try do all the tasks, actual servo movement controlled by signal variables
    feed()
    clean()
    play()
# end of the code.py
```

Appendix G – Web app code

```
# webServer of the website

# webServer.py
```

```
from flask import Flask, render_template, request, redirect

import socket


# setting of socket webClient

app = Flask(__name__)

PORT = 777

HOST = "10.93.48.142"

ADDR = (HOST, PORT)

HEADER = 64

FORMAT = "utf-8"


# connecting to the socket server

webClient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

webClient.connect(ADDR)



# define the method of send message and receive message

def send(msg):

    message = msg.encode(FORMAT)

    msg_length = len(message)

    send_length = str(msg_length).encode(FORMAT)

    send_length += b' ' * (HEADER - len(send_length))

    webClient.sendall(send_length)

    webClient.sendall(message)



def receive_reply(conn):
    msg_length = conn.recv(HEADER).decode(FORMAT)

    msg = ""

    if msg_length:

        msg_length = int(msg_length)

        msg = conn.recv(msg_length).decode(FORMAT)

    return msg



# define the main route of the webpage

@app.route("/", methods=["GET", "POST"])

def main():

    # we set the message from web start with 'web'

    message = "web"

    schefeed = ["", "", ""]

    if request.method == 'POST':

          # when the schedule button pressed we send a message to request the schedule information
from the socket server

        # then we receive the message and arrange them into the schedule table

        if request.form.get('schedule') == 'schedule':

            message += " schedule"

            send(message)

            data = receive_reply(webClient)

            print(data)

            sche = data.split(" ")

            for n in range(len(sche)):

                schefeed[n] += sche[n]

            print(schefeed)
```

```
                return render_template("index.html", sfeed1time=schefeed[0], sfeed2time=schefeed[1],
sfeed3time=schefeed[2])

        # when the feed button pressed we append 'feed' to the message

        if request.form.get('feed') == 'feed':

            message += " feed"

        # when the clean button pressed we append 'clean' to the message

            message += " clean"

        # when the play button pressed we append 'play on' to the message

        if request.form.get('play') == 'play on':

            message += " play on"

        # when the play button pressed we append 'play off' to the message

        if request.form.get('play') == 'play off':

            message += " play off"

        # send the message to backend server

        send(message)

    return render_template("index.html")



# the route for setting page

@app.route("/setting", methods=["GET", "POST"])

def setting():

    # this for the starter of the update message

    message = "web update"

    if request.method == 'POST':

        # we get the information from the input on the website

        if request.form.get('submit_button') == 'Submit':

            message += " "
            message += request.form.get('numberOfTimes')
            message += " "
            if request.form.get('numberOfTimes') == '1':
                message += str(request.form.get('time1'))
            elif request.form.get('numberOfTimes') == '2':
                message += str(request.form.get('time1'))
                message += " "
                message += str(request.form.get('time2'))
            elif request.form.get('numberOfTimes') == '3':
                message += str(request.form.get('time1'))
                message += " "
                message += str(request.form.get('time2'))
                message += " "
                message += str(request.form.get('time3'))
            message += " "
            message += request.form.get('feedAmount')
            # arrange the setting information and send it to the backend server
            send(message)
            # after submit, the webpage jump back to the main page
            return redirect("/")
    return render_template("settingpage.html")



# the route for history page
@app.route("/history", methods=["GET", "POST"])
def history():
    # the starter of the history request message
```

```
message = "web history"

send(message)

data = receive_reply(webClient)

print(data)

# the list to save information table

feedhistime = [" ", " ", " ", " ", " "]

feedhisamount = [" ", " ", " ", " ", " "]

cleanhistime = [" ", " ", " ", " ", " ", " "]

text = data.split(" clean ")

feedlist = text[0].split(" ")

print(feedlist)

cleanlist = text[1].split(" ")

print(cleanlist)

lenfeed = len(feedlist)

lenclean = len(cleanlist)

numfeed = int(feedlist[1])

numclean = int(cleanlist[0])

# rendering the message into the format to fit in the website

if numfeed > 5:

    for n in range(5):

        feedhistime[5 - n] += feedlist[lenfeed - 2 * n - 2]

        feedhisamount[5 - n] += feedlist[lenfeed - 2 * n - 1]

elif numfeed > 0:

    for n in range(numfeed):

        feedhistime[n] += feedlist[lenfeed - 2 * n - 2]

        feedhisamount[n] += feedlist[lenfeed - 2 * n - 1]
    if numclean > 5:
        for n in range(5):
            cleanhistime[n] += cleanlist[lenclean - n]
    elif numclean > 0:
        for n in range(numclean):
            cleanhistime[n] += cleanlist[lenclean - n - 1]

        return  render_template("history.html",  feed1time=feedhistime[0],  feed2time=feedhistime[1],
feed3time=feedhistime[2],
                                                 feed4time=feedhistime[3],  feed5time=feedhistime[4],
feed1amount=feedhisamount[0],
                                           feed2amount=feedhisamount[1],  feed3amount=feedhisamount[2],
feed4amount=feedhisamount[3],
                                              feed5amount=feedhisamount[4],  clean1time=cleanhistime[0],
clean2time=cleanhistime[1],
                                                 clean3time=cleanhistime[2],  clean4time=cleanhistime[3],
clean5time=cleanhistime[4])



if __name__ == "__main__":
    app.run(debug=True, host="10.93.48.142", port=80)
```

# end of the webServer.py

# start of index.html (main page)

```
<!DOCTYPE html>
<html>
<body style="background-color:linen;">
<head>
    <meta  charset="utf-8">
```

```
<title>

  Project 2 G14

</title>

<!--this is the quick start CSS link of bootstrap-->

                                                  <link                rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2M
Zw1T" crossorigin="anonymous">

<style>


    .headericon{

      margin-bottom: 10px;

    }




    .overallheader{

      display: flex;

      background-color: #b58037;

      height: 60px;

      margin-bottom: 15px

    }


    .flex-container > div {

     background-color: green;

     width: 100px;

     margin: 10px;

     text-align: center;

     line-height: 75px;
     font-size: 30px;
    }

    .headertext {
      vertical-align: middle;
      font-size: 20px;
      font-family: "Times New Roman", Times, serif;
      margin-left: 35px;
    }

    img {
      width:100px;
      height: 100px;
      border-radius: 50px;
      float: left
      margin-right: 100px;
      margin-left: 70px;
    }

    .button{
      display: inline-block;
    }

    button {
      background-color: white;
      border-color: grey;
      border-style: solid;
      border-width: 3px;
      height: 50px;
      width: 300px;

      color:  #212427;
      cursor: pointer;
      margin-right: 10px;
      border-radius: 10px;
      font-weight: bold;
      font-size: 15px;

      transition: opacity 0.1s; background-color 0.3s;
    }

    button:hover{
      background-color: grey;
      color: white;
      box-shadow: 2px 2px 2px rgba(0,0,0,0.15);
    }

    button:active{
      opacity: 0.7;
      box-shadow: 2px 2px 2px rgba(0,0,0,0.15);
    }

    .controls {
      width: 400px;
```

```
}

.form-group{
    margin: auto;
    width: 50%;
    padding: 10px;
}
/* The switch - the box around the slider */
.switch {
  position: relative;
  display: inline-block;
  width: 60px;
  height: 34px;
}

/* Hide default HTML checkbox */
.switch input {
  opacity: 0;
  width: 0;
  height: 0;
}

/* The slider */
.slider {
  position: absolute;
  cursor: pointer;
  top: 0;
  left: 0;

  right: 0;

  bottom: 0;

  background-color: #ccc;

  -webkit-transition: .4s;

  transition: .4s;

}


.slider:before {

  position: absolute;

  content: "";

  height: 26px;

  width: 26px;

  left: 4px;

  bottom: 4px;

  background-color: white;

  -webkit-transition: .4s;

  transition: .4s;

}


input:checked + .slider {

  background-color: #2196F3;

}


input:focus + .slider {

  box-shadow: 0 0 1px #2196F3;

}
```

```
input:checked + .slider:before {
  -webkit-transform: translateX(26px);
  -ms-transform: translateX(26px);
  transform: translateX(26px);
}

.videostream{
    width: 50;
    margin-left: 300px;
}

table{
    margin-left: 15px;
}
/* Rounded sliders */
.slider.round {
  border-radius: 34px;
}

.slider.round:before {
  border-radius: 50%;
}

.videocontainer{
    display: flex;
}
```

```
   </style>
</head>
<body>
<hr>
<div class="headericon">
                                                                                              <img
src="https://media.istockphoto.com/id/1324471626/vector/dog-love-simple-logo.jpg?s=612x6
12&w=0&k=20&c=U7PzRbOpk9MVCVIfT3ONvnFbcOnpzmQM7eIAWGNy1ok=">


</div>


<div class="overallheader">
   <div style = "align-self: center;font-size: 20px; margin-left: 35px; " class="headertext">
       THE PET MONITOR
   </div>
   <div style= "align-self: center; flex: 30%; margin-left: 10%" class="headerbutton">
      <a href="/setting"><button style="width:130px; height : 40px">Set Schedule</button>
   </a>
   </div>
   <div style= "align-self: center; flex: 30%" class="headerbuttontwo">
            <a href="/history"><button style="width:100px; height : 40px"  name="history"
value="history">History</button></a>
   </div>
</div>
<div class="videocontainer">
   <div class ="videostream">
```

```
        <iframe width="560" height="315" src="ipcameraaddress" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>
    </div>
    <div>
        <table>
            <tr>
                <th>Scheduled Feed Time</th>
            </tr>
            <tr>
                <td>{{sfeed1time}}</td>
            </tr>
            <tr>
                <td>{{sfeed2time}}</td>
            </tr>
            <tr>
                <td>{{sfeed3time}}</td>
            </tr>
        </table>
    </div>
</div>
<hr>
<div style = "font-size: 20px;width: 560px; margin: auto; margin-bottom: 15px" class="headertext">
    Controls
</div>
<form method="POST" action="/">
<div style="margin: auto" class="controls">
```

```
  <button name="schedule" type="submit" value="schedule">Schedule</button>

  <button name="feed" type="submit" value="feed">Feed</button>

  <button name="clean" value="clean" type="submit">Clean</button>

          <button onclick="myFunction()" name="play" id="play" type="submit"
value="play">Play</button>

 <label class="switch">

  <input id="myCheckbox" type="checkbox">

  <span class="slider round"></span>

 </label>

</div>

</form>

<hr>

<form>

  <div class = "form-group">   <p>Sign up for Email notification:</p>

  </div>

  <div class="form-group">

    <label for="exampleInputEmail1">Email address</label>

              <input type="email" class="form-control" id="exampleInputEmail1"
aria-describedby="emailHelp" placeholder="Enter email">

     <small id="emailHelp" class="form-text text-muted">We'll never share your email with
anyone else.</small>

  </div>

  <div class="form-group">

    <label for="exampleInputPassword1">Confirm Email address</label>

              <input type="email" class="form-control" id="exampleInputEmail1"
aria-describedby="emailHelp" placeholder="Enter email">

  </div>
  <div class="form-group">
              <button class="btn btn-secondary dropdown-toggle" type="button"
id="dropdownMenuButton" data-toggle="dropdown" aria-haspopup="true"
aria-expanded="false">
      Notification Frequency
    </button>
    <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
      <a class="dropdown-item" href="#">Real Time</a>
      <a class="dropdown-item" href="#">Hourly</a>
      <a class="dropdown-item" href="#">Daily</a>
    </div>
  </div>
  <div class="form-group">
    <button type="submit" class="form-group">Submit</button>
  </div>
</form>
<script>
  function myFunction() {
    let checkBox = document.getElementById("myCheckbox");
    if (checkBox.checked === true) {
      document.getElementById("play").value = "play on"
    }
    else {
      document.getElementById("play").value = "play off"
    }
  }
```

```
</script>

<hr>

<p> "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est
laborum."

</p>

   <!--this is the quick start javascript link of bootstrap-->

                                   <script          src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jiz
o" crossorigin="anonymous"></script>

            <script   src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/dist/umd/popper.min.js"
integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIH
NDz0W1" crossorigin="anonymous"></script>

             <script   src="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM
" crossorigin="anonymous"></script>

</body>

</html>

# end of the index.html
```

# Start of the settingpage.html

```
<!DOCTYPE html>

<html lang="en">

<body style="background-color:linen;">

<head>

   <meta charset="UTF-8">

   <title>Title</title>

   <style>
```

```
.headericon{
    margin-bottom: 10px;
}

.overallheader{
    display: flex;
    background-color: #b58037;
    height: 60px;
    margin-bottom: 15px
}

.flex-container > div {
  background-color: green;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}

.headertext {
    vertical-align: middle;
    font-size: 20px;
    font-family: "Times New Roman", Times, serif;
    margin-left: 35px;
}
```

```
img {

    width:100px;

    height: 100px;

    border-radius: 50px;

    float: left

    margin-right: 100px;

    margin-left: 70px;

}


.button{

    display: inline-block;

}


button {

    background-color: white;

    border-color: grey;

    border-style: solid;

    border-width: 3px;

    height: 50px;

    width: 100px;

    color:  #212427;

    cursor: pointer;

    margin-right: 10px;

    border-radius: 10px;

    font-weight: bold;

    font-size: 15px;

    transition: opacity 0.1s; background-color 0.3s;
```

```
}


button:hover{
    background-color: grey;
    color: white;
    box-shadow: 2px 2px 2px rgba(0,0,0,0.15);
}


button:active{
    opacity: 0.7;
    box-shadow: 2px 2px 2px rgba(0,0,0,0.15);
}


.controls {
    width: 400px;
}




.form-group{
    margin: auto;
    width: 50%;
    padding: 10px;
}


.flex-container {
  display: flex;
```

```
          justify-content: center;
          margin-bottom: 15px;
        }

        .timeselect {
          display: flex;
          justify-content: center;
        }

        ul, ol {
          list-style-type: none;
        }
    </style>
</head>
<body>
<hr>
<div class="headericon">
                                                                                                      <img
src="https://media.istockphoto.com/id/1324471626/vector/dog-love-simple-logo.jpg?s=612x6
12&w=0&k=20&c=U7PzRbOpk9MVCVIfT3ONvnFbcOnpzmQM7eIAWGNy1ok=">
</div>

<div class="overallheader">
    <div style = "align-self: center;font-size: 20px; margin-left: 35px; " class="headertext">
        THE PET MONITOR
    </div>
    <div style= "align-self: center; flex: 30%; margin-left: 10%" class="headerbutton">
```

```
        <a href="/"><button style="width:100px; height : 40px">  Back </button><a/>
    </div>
    <div style= "align-self: center; flex: 30%" class="headerbuttontwo">
        <a href="/history"><button style="width:100px; height : 40px">  History </button><a/>
    </div>
</div>
<div class="flex-container">
    <button onclick="addTime()">Add Time</button>
    <button onclick="deleteTime()">Delete Time</button>
</div>
<div class = "timeselect">
    <form id="myForm" name="myForm" action="" method="post">
        <ul id="timeList">
            <li>
                <label for="time1">feed at: </label>
                <input type="time" id="time1" name="time1" min="00:00" max="24:00" required>
            </li>
        </ul>
        <label for="feedAmount">Set Max food quntity in gram</label>
            <input  type="number"  name="feedAmount"  id="feedAmount"  min="0"  max="2000"
required>
                <input  type="hidden"  id="numberOfTimes"  name="numberOfTimes"  value="1"
required>
        <br>
         <button style="margin-left:70px; margin-top:15px" type="submit" name='submit_button'
value='Submit' class="form-group">Submit</button>
    </form>
</div>
```

```
</body>

<script>

  function addTime() {

    let number = document.myForm.numberOfTimes.value

    if(number >= 3) {

      alert("You can't click me anymore");

      return;

    }

    document.myForm.numberOfTimes.value = (parseInt(number) + 1).toString()

    number = document.myForm.numberOfTimes.value

    document.getElementById("numberOfTimes").innerHTML = (number + 1).toString();

    var list = document.getElementById("timeList");

    var li = document.createElement("li");

    var lable = document.createElement("lable");

    var input = document.createElement("input");


    lable.appendChild(document.createTextNode("feed at: "));

    lable.setAttribute("for", "time"+ number.toString())

    input.setAttribute("name", "time"+ number.toString());

    input.setAttribute("type", "time");

    input.setAttribute("id", "time"+ number.toString());

    input.setAttribute("min", "00:00");

    input.setAttribute("max", "24:00");

    input.setAttribute("required", "required");


    li.append(lable);

    li.append(input);
```

```
        list.append(li);
    }
    function deleteTime() {
        let number = document.myForm.numberOfTimes.value
        if(number <= 1) {
            alert("You can't click me anymore");
            return;
        }
        document.myForm.numberOfTimes.value = (parseInt(number) - 1).toString()
        let list = document.getElementById("timeList");
        if (list.firstChild) {
            let child = document.getElementById("timeList").lastChild;
            list.removeChild(child)
        }
    }
</script>
</html>
```

# End of the settingpage.html

# Start of the history.html

```
<!DOCTYPE html>
<html lang="en">
<body style="background-color:linen;">
<head>
    <meta charset="UTF-8">
    <title>History</title>
    <style>
```

```
.headericon{
      margin-bottom: 10px;
    }

    .overallheader{
      display: flex;
      background-color: #b58037;
      height: 60px;
      margin-bottom: 15px
    }

    .flex-container > div {
     background-color: green;
     width: 100px;
     margin: 10px;
     text-align: center;
     line-height: 75px;
     font-size: 30px;
    }

    .headertext {
      vertical-align: middle;
      font-size: 20px;
      font-family: "Times New Roman", Times, serif;
      margin-left: 35px;
    }
```

```
img {
    width:100px;
    height: 100px;
    border-radius: 50px;
    float: left
    margin-right: 100px;
    margin-left: 70px;
}

.button{
    display: inline-block;
}

button {
    background-color: white;
    border-color: grey;
    border-style: solid;
    border-width: 3px;
    height: 50px;
    width: 100px;
    color:  #212427;
    cursor: pointer;
    margin-right: 10px;
    border-radius: 10px;
    font-weight: bold;
    font-size: 15px;
    transition: opacity 0.1s; background-color 0.3s;
```

```
}

button:hover{

    background-color: grey;

    color: white;

    box-shadow: 2px 2px 2px rgba(0,0,0,0.15);

}

button:active{

    opacity: 0.7;

    box-shadow: 2px 2px 2px rgba(0,0,0,0.15);

}

.controls {

    width: 400px;

}

/* Style the table */

table {

    width: 80%;

    margin: 0 auto;

    border-collapse: collapse;

}


/* Style the table cells */

td, th {

    padding: 15px;

    text-align: left;

}

.outer {

    display: flex;
    width: 100%; /* Set the outer div's width to 100% of the parent container */
}
.inner {
    width: 50%;
}
.vertical-line {
    border-left: 2px solid #b58037;
    height: 300px; /* Set the height of the line */
}
.form-group{
    margin: auto;
    width: 50%;
    padding: 10px;
}
.flex-container {
  display: flex;
  justify-content: center;
  margin-bottom: 15px;
}
.timeselect {
  display: flex;
  justify-content: center;
}

ul, ol {
  list-style-type: none;
}
```

```
  </style>
</head>
<body>
<hr>
<div class="headericon">
                                                                                      <img
src="https://media.istockphoto.com/id/1324471626/vector/dog-love-simple-logo.jpg?s=612x6
12&w=0&k=20&c=U7PzRbOpk9MVCVIfT3ONvnFbcOnpzmQM7eIAWGNy1ok=">

</div>
<div class="overallheader">
  <div style = "align-self: center;font-size: 20px; margin-left: 35px; " class="headertext">
    THE PET MONITOR
  </div>

  <div style= "align-self: center; flex: 30%; margin-left: 10%" class="headerbutton">
    <a href="/setting"> <button style="width:100px; height : 40px">  Setting </button><a/>
  </div>

  <div style= "align-self: center; flex: 30%" class="headerbuttontwo">
    <a href="/"><button style="width:100px; height : 40px">  Back </button></a>
  </div>
</div>

<div class="outer">
  <div class="inner">
```

```
<table>

  <tr>

    <th>Feed Time</th>

    <th>Feed Amount</th>

  </tr>

  <tr>

    <td>{{feed1time}}</td>

    <td>{{feed1amount}}</td>

  </tr>

  <tr>

    <td>{{feed2time}}</td>

    <td>{{feed2amount}}</td>

  </tr>

  <tr>

    <td>{{feed3time}}</td>

    <td>{{feed3amount}}</td>

  </tr>

  <tr>

    <td>{{feed4time}}</td>

    <td>{{feed4amount}}</td>

  </tr>

  <tr>

    <td>{{feed5time}}</td>

    <td>{{feed5amount}}</td>

  </tr>


</table>
```

```
    </div>
    <div class="vertical-line"></div>
    <div class="inner">
        <table>
            <tr>
                <th>Clean Time</th>
            </tr>
            <tr>
                <td>{{clean1time}}</td>
            </tr>
            <tr>
                <td>{{clean2time}}</td>
            </tr>
            <tr>
                <td>{{clean3time}}</td>
            </tr>
            <tr>
                <td>{{clean4time}}</td>
            </tr>
            <tr>
                <td>{{clean5time}}</td>
            </tr>
        </table>
    </div>
</div>
</div>
</body>
```

```
</html>
```

# end of the history.html

Appendix H – Machine Learning code and data

No ML was used/implemented

Appendix I – Other code

# backend python socket server

```
# VM backend server.py

import socket

import threading

import time


# server info

HEADER = 64

PORT_PICO = 443

PORT_WEB = 777

SERVER = socket.gethostbyname(socket.gethostname())

ADDR_PICO = (SERVER, PORT_PICO)

ADDR_WEB = (SERVER, PORT_WEB)

FORMAT = "utf-8"

INSTRUCTION_INTERVAL = 5

MAX_HISTORY_RECORD = 5


# saved data

currInst: str = "None"
```

```
currFoodWeight: float = 0.0

maxFoodWeight: float = 100.0

feedSchedule: list = list() # (str: time, bool: done) time in format: "hour minute"

feedHistory: list = list() # (str: time, str: amount) time in format: "year mon day hour minute"

cleanHistory: list = list() # (str: time) time in format: "year mon day hour minute"

play: bool = False

play_status : str = "off"

manualFeed = False

manualClean = False


server_pico = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_pico.bind(ADDR_PICO)


server_web = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_web.bind(ADDR_WEB)


previousTime:list = list()


def refreshSchedule():

    global feedSchedule

    for i in range(len(feedSchedule)):

        feedSchedule[i] = (feedSchedule[i][0],False)

# struct_time to string for time comparison : hh mm ss

def time_transfer_day(localTime: time.struct_time):

    hour = localTime.tm_hour

    minute = localTime.tm_min

    second = localTime.tm_sec
    result = str(hour) + " " + str(minute) + " " + str(second)

    return result

# struct_time to string for data storage: yyyy/mm/dd_hh:mm

def time_transfer_date(localTime: time.struct_time):

    year = localTime.tm_year

    month = localTime.tm_mon

    day = localTime.tm_mday

    hour = localTime.tm_hour

    minute = localTime.tm_min

    result = str(year) + "/" + str(month) + "/" + str(day) + "_" + str(hour) + ":" + str(minute)

    return result


# check if the difference between two time_string is within certain range

def time_in_range_day(time1: str, time2: str):

    time1_list = time1.split()

    time2_list = time2.split()


    time_1 = int(time1_list[0])*3600 + int(time1_list[1])*60 + int(time1_list[2])

    time_2 = int(time1_list[0])*3600 + int(time2_list[1])*60 + int(time2_list[2])

    return abs(time_1 - time_2) <= INSTRUCTION_INTERVAL * 2


# sending message to certain connection with appropriate protocal header

def reply(conn, msg):

    message = msg.encode(FORMAT)

    # fixed length header before actual message

    msg_length = len(message)

    send_length = str(msg_length).encode(FORMAT)
    send_length += b' ' * (HEADER - len(send_length))


    conn.send(send_length)

    conn.send(message)


# preparation before sending instruction to pico

def update_inst():

    now = time.localtime()

    now_str: str = time_transfer_day(now)

    # for schedule refresh

    instruct = "None"

    global manualFeed

    global manualClean

    global feedHistory

    global cleanHistory

    # manual command has higher priority than timed event

    manual: bool = False

    if manualFeed:

        # for history record

        amount = maxFoodWeight - currFoodWeight

        if amount < 0:

            amount = 0

        # record the action in a list

        feedHistory.append((time_transfer_date(now), str(amount)))

        # only keeps the last n records, remove the earliest record if it exceeds the limit

        if len(feedHistory) > MAX_HISTORY_RECORD:

            feedHistory = feedHistory[-MAX_HISTORY_RECORD:]
        # actual message
        instruct = "feed" + " " + str(maxFoodWeight)
        manual = True
        manualFeed = False
    elif manualClean:
        # record the action in a list
        cleanHistory.append(time_transfer_date(now))
        # only keeps the last n records, remove the earliest record if it exceeds the limit
        if len(cleanHistory) > MAX_HISTORY_RECORD:
            cleanHistory = cleanHistory[-MAX_HISTORY_RECORD:]
        # actual message
        instruct = "clean"
        manual = True
        manualClean = False

    if not manual: # scheduled feed
        global feedSchedule
        for i in range(len(feedSchedule)):
            # check if time has reached the setting feeding time
            if time_in_range_day(feedSchedule[i][0], now_str) and not feedSchedule[i][1]:
                # for history record
                amount = maxFoodWeight - currFoodWeight
                if amount >= 0:
                    amount = 0
                # record the action in a list
                feedSchedule[i] = (feedSchedule[i][0], True)
                feedHistory.append((time_transfer_date(now), str(amount)))
            # only keeps the last n records, remove the earliest record if it exceeds the limit
            if len(feedHistory) > MAX_HISTORY_RECORD:
                feedHistory = feedHistory[-MAX_HISTORY_RECORD:]
            # construct actual message
            instruct = "feed" + " " + str(maxFoodWeight)
            break

    global play
    if play:
        instruct = "play" + " " + play_status
        play = False

    global currInst
    currInst = instruct
# send instruction to pico
def sendInst(conn):
    update_inst()
    print(f"To pico instruction: {currInst}")
    reply(conn, currInst)
# change local variables for instruction sending
def update_play(status):
    global play
    play = True
    global play_status
    play_status = status
# change local variables for instruction sending
def update_max_food(amount):
    global maxFoodWeight
    if float(amount) > 0:
        maxFoodWeight = float(amount)
# current food weight on hardware, getting from pico server connection
def updateWeight(weight):
    global currFoodWeight
    currFoodWeight = weight
# change the feeding schedule
def updateSchedule(new_schedule: list, number: int):
    global feedSchedule
    feedSchedule = list()


    for i in range(number):
        time_list = new_schedule[i].split(":")
        hour = time_list[0]
        minute = time_list[1]
        second = "0"
        time = str(hour) + " " + str(minute) + " " + second
        feedSchedule.append((time, False))
# send history info to web server in the form of: "feed [feedNum] {[feedTime] [feedAmount]} clean [cleanNum] {[cleanTime]}
def sendHistory(conn):
    msg = ""
    feedNum = len(feedHistory)
    msg += "feed "
    msg += str(feedNum)
    for i in range(feedNum):
        t_str = feedHistory[i][0]
        amount: str = feedHistory[i][1]
        msg += " "
        msg += t_str
        msg += " "
        msg += amount

    cleanNum = len(cleanHistory)
    msg += " clean "
    msg += str(cleanNum)
    for i in range(cleanNum):
        t_str = cleanHistory[i]
        msg += " "
        msg += t_str
    print(f"sendHistory {msg}")
    reply(conn, msg)

# send current schedule info to web server in the form of: "[scheduleNum] {[feedTime]}
def sendSchedule(conn):
    msg = ""
    pairNum = len(feedSchedule)
    for i in range(pairNum):
        time_str_list = feedSchedule[i][0].split()
        msg += time_str_list[0]
        msg += ":"
        msg += time_str_list[1]
        if i != pairNum - 1 :
            msg += " "

    print(f"sendSchedule {msg}")
    reply(conn, msg)

# interaction with pico W through socket + logic control
def handle_pico(conn, msg_list):
    command = msg_list[1]
    if command == "instruction":
        sendInst(conn)
    elif command == "weight":
        w = float(msg_list[2])
        updateWeight(w)

# interaction with web server
def handle_web(conn,msg_list):
    command = msg_list[1]
    if command == "feed":
        global manualFeed
        manualFeed = True
    elif command == "clean":
        global manualClean
        manualClean = True
    elif command == "update":
        number = int(msg_list[2])
        new_schedule: list = msg_list[3:-1]
        amount = msg_list[-1]

        updateSchedule(new_schedule, number)

        update_max_food(amount)

    elif command == "history":

        sendHistory(conn)

    elif command == "schedule":

        sendSchedule(conn)

    elif command == "play":

        status = msg_list[2]

        update_play(status)


# general message receive from client

def handle_client(conn, addr):

    print(f"New connection {addr} connected")

    connected  = True

    while connected:

        msg_length = conn.recv(HEADER).decode(FORMAT)

        if msg_length:

            msg_length = int(msg_length)

            msg = conn.recv(msg_length).decode(FORMAT)

            print(*msg)

            msg_list = msg.split()

            client_id = msg_list[0]

            # recognize different devices by message first word: part of protocol

            if client_id == "pico":

                handle_pico(conn, msg_list)

            elif client_id == "web":
```

```
            handle_web(conn, msg_list)
        else:
            print("unknown connection device")
    conn.close()


# server action
def start():
    # start both servers listening to their ports
    server_pico.listen()
    server_web.listen()
    pico_connected = False
    while True:
        # can only have one pico connection
        if not pico_connected:
            conn, addr = server_pico.accept()
            pico_connected = True
            # communicate with client on a new thread
            thread = threading.Thread(target=handle_client, args=(conn, addr))
            thread.start()
            print(f"Active connection: {threading.activeCount() - 1}" )
        # could have multiple web connections
        conn1, addr1 = server_web.accept()
        # communicate with client on a new thread
        thread1 = threading.Thread(target=handle_client, args=(conn1, addr1))
        thread1.start()
        print(f"Active connection: {threading.activeCount() - 1}" )
print("Server is starting...")
```

```
start()

# end of the webServer.py
```

Appendix J - Other

Feeder: We build it with a big plastic bottle and open a hole at the bottom, then we design a turntable switch to control the opening. The turntable is controlled by one of the three servos.

Cat Teaser: The teaser is built with yarn and a little plastic toy, which can be played with by the cat.

Cleaner: Built with a disposable chopstick and a shovel made of plastic and connected with one servo.

Scale: We use an upside-down plate as the base and stick three small pressure sensors on it, then cut the same size paper board to cover all the sensors.