Individual Contribution Summary

ECE Senior Design | Spring 2026 | Wyatt Gahm - CpE

Remote Control Remote Monitor Cat Toy

Project Overview

The remote monitor and remote control cat toy project primarily aims to allow a user who is anywhere to use the internet to connect to a camera in their home, and operate a small mouse robot. The purpose is to provide novel engagement for both the pet and operator. This idea is built on with the concept of autonomous play, mouse and cat tracking and mouse motion that entertains the cat as well. In summary, the desired experience for the user is a video featuring a high vantage point perspective of the room and controls to operate a robot that a cat will chase. The scope of the project encompasses live camera feed design, small robotics design, and object detection & tracking.

Individual Contributions

I worked primarily in the networking, CAD, editorial, and brainstorming sections of the project. Some of my largest contributions were the 3d printed robot enclosure, design expo poster, video streaming code, and core ideation surrounding the technologies and systems utilized. First, I developed and refined the robot enclosure to accommodate the motors, PCB, battery, wheels, and wires of the design [Appendix A]. I oversaw the assembly of the robot and the production of the PLA plastic parts. Quickly adapting and remanufacturing this critical component allowed other team members to focus on autonomous mode and refined electronics, enabling the robot to finally move for tracking and autonomous tests. Second, I designed the poster, which allowed us to attend the design expo and communicate our project [Appendix B]. I developed the video streaming code with the ability to send controls in the first two weeks of the project, serving as a proof of concept for the feasibility of the entire project, and later refined the networking capabilities to achieve lower latency and more entertaining control of the mouse overall [Appendix C]. Lastly in miscellaneous scope, I developed preliminary schematics utilizing the ESP32 SoC to drive H bridge connected motors[Appendix D], exploring the challenges of the embedded design. I also developed a Python library to interact with the mouse over BLE, as a plug and play component to aid autonomous development [Appendix E]. Lastly, I contributed to various editorial tasks like making graphics, slides, and assorted visual aids [Appendix F].

Knowledge Applied & Acquired

Throughout the project I leveraged my knowledge acquired in Computer Networks, Electronic Applications, and Design Fundamentals to produce highly functional video streaming, documentation, and designs. I gained new knowledge by researching needed context in academic journals and datasheets, in conjunction with hands-on experience messing around with CAD tools like OnShape. During the project I read lots of documentation and RFC documents [Appendix G] in order to understand the WebRTC protocol suite as well.

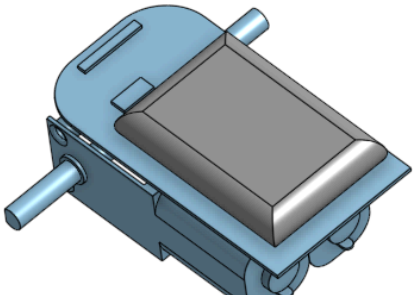
Team & Project Impact

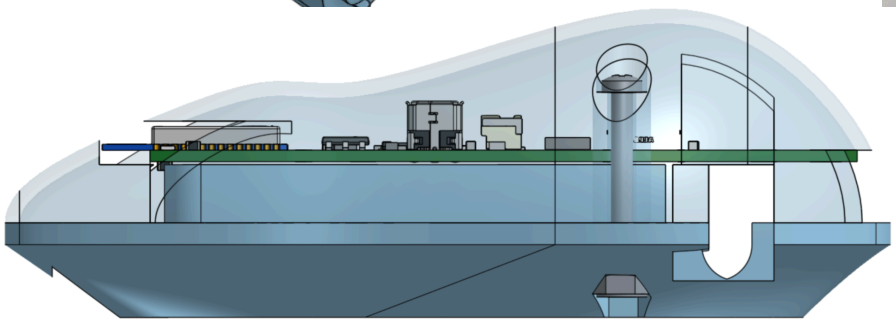
Me and my team were able to successfully demonstrate remote monitoring and control for cat entertainment. This is due in part to the contributions from the hardware design roles which allowed my work to be precise and accurate when modeling and developing communication libraries. I also initially proposed some of our design choices, such as using an ESP32 SoC, WebRTC, 20mm DC motors with gearboxes, and Python.

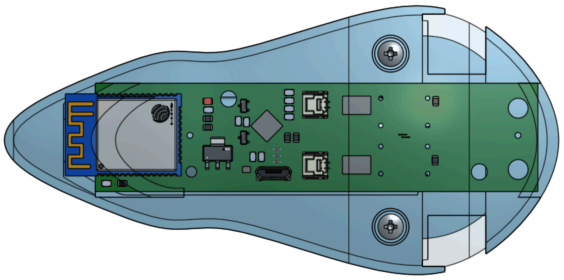
Artifact Summary

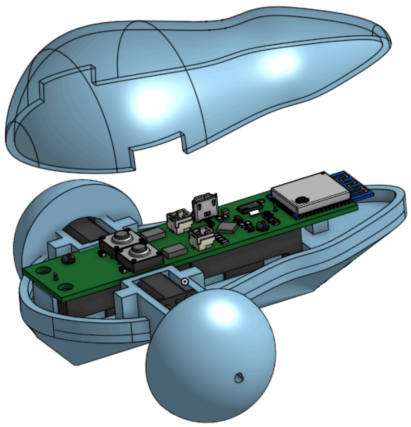
Page #	Appendix #	Summary
3	A	Various iterations of the mouse enclosure which I developed and 3d printed.
4	B	Poster with contributions made by other teammates omitted for clarity.
5	C	The code I developed for streaming video over the internet securely.
8	D	A preliminary schematic of the design of the robot hardware.
9	E	Mouse.py - A Python library for controlling and interacting with the mouse.
14	F	Assorted graphics and visual aids. Includes:
17	G	Various citations to RFC documents that explain the workings of WebRTC and underlying mechanisms.

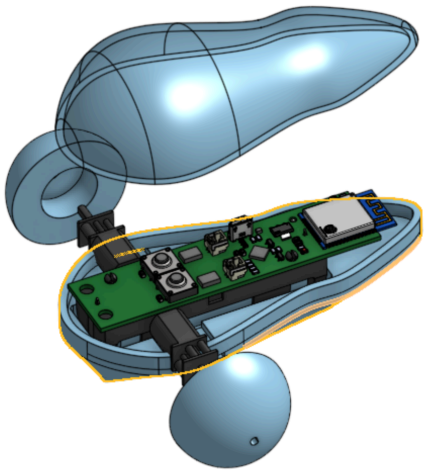
Appendix A



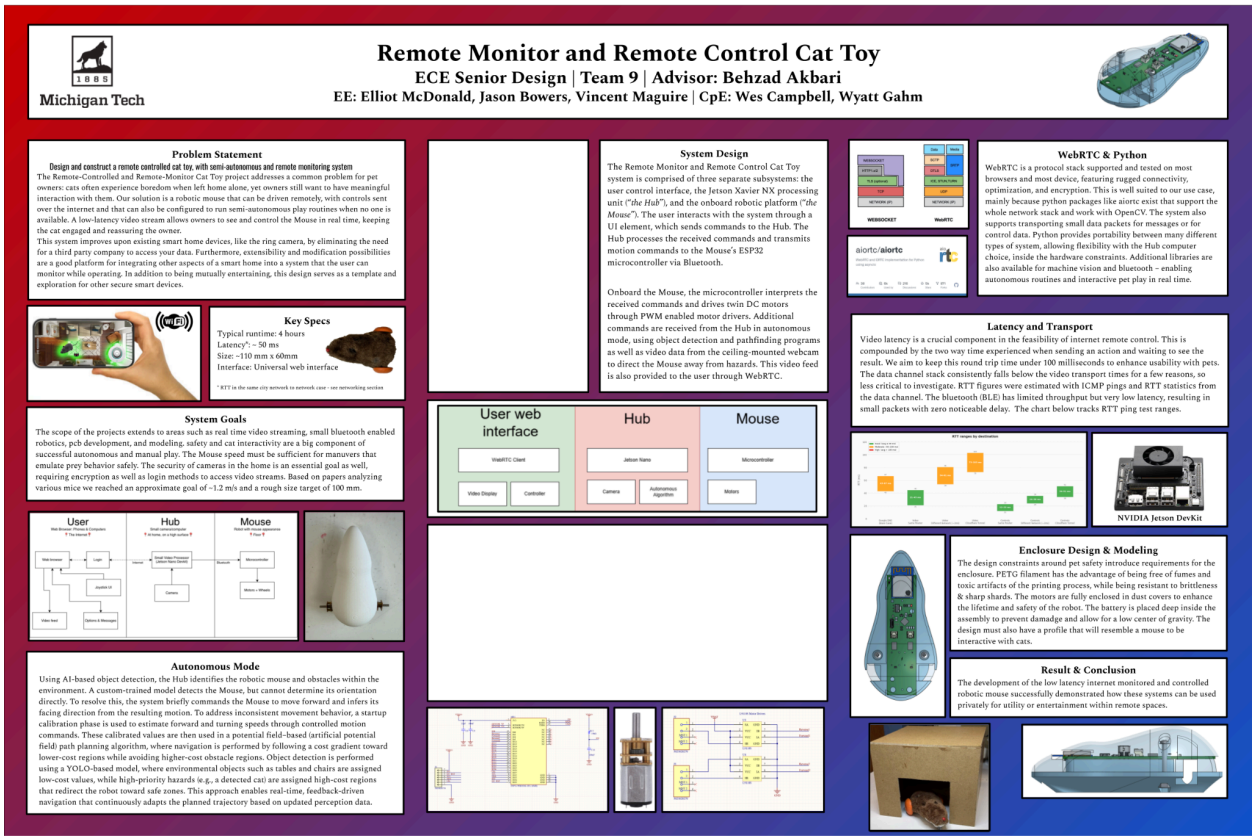








Appendix B


Michigan Tech
Remote Monitor and Remote Control Cat Toy
ECE Senior Design | Team 9 | Advisor: Behzad Akbari
EE: Elliot McDonald, Jason Bowers, Vincent Maguire | CpE: Wes Campbell, Wyatt Gahm
Problem Statement
Design and construct a remote controlled cat toy, with semi-autonomous and remote monitoring system
The Remote-Controlled and Remote-Monitor Cat Toy project addresses a common problem for pet owners: cats often experience boredom when left home alone, yet owners still want to have meaningful interaction with them. Our solution is a robotic mouse that can be driven remotely, with controls sent over the internet and that can also be configured to run semi-autonomous play routines when no one is available. A low-latency video stream allows owners to see and control the Mouse in real time, keeping the cat engaged and reassuring the owner.
This system improves upon existing smart home devices, like the ring camera, by eliminating the need for a third party company to access your data. Furthermore, extensibility and modification possibilities are a good platform for integrating other aspects of a smart home into a system that the user can monitor while operating. In addition to being mutually entertaining, this design serves as a template and exploration for other secure smart devices.
Key Specs
Typical runtime: 4 hours
Latency*: ~ 50 ms
Size: ~110 mm x 60mm
Interface: Universal web interface
* RTT in the same city network to network case - see networking section
System Goals
The scope of the projects extends to areas such as real time video streaming, small bluetooth enabled robotics, pcb development, and modeling, safety and cat interactivity are a big component of successful autonomous and manual play. The Mouse speed must be sufficient for manuvers that emulate prey behavior safely. The security of cameras in the home is an essential goal as well, requiring encryption as well as login methods to access video streams. Based on papers analyzing various mice we reached an approximate goal of ~1.2 m/s and a rough size target of 100 mm.
User
Hub
Mouse
Autonomous Mode
Using AI-based object detection, the Hub identifies the robotic mouse and obstacles within the environment. A custom-trained model detects the Mouse, but cannot determine its orientation directly. To resolve this, the system briefly commands the Mouse to move forward and infers its facing direction from the resulting motion. To address inconsistent movement behavior, a startup calibration phase is used to estimate forward and turning speeds through controlled motion commands. These calibrated values are then used in a potential field–based (artificial potential field) path planning algorithm, where navigation is performed by following a cost gradient toward lower-cost regions while avoiding higher-cost obstacle regions. Object detection is performed using a YOLO-based model, where environmental objects such as tables and chairs are assigned low-cost values, while high-priority hazards (e.g., a detected cat) are assigned high-cost regions that redirect the robot toward safe zones. This approach enables real-time, feedback-driven navigation that continuously adapts the planned trajectory based on updated perception data.
System Design
The Remote Monitor and Remote Control Cat Toy system is comprised of three separate subsystems: the user control interface, the Jetson Xavier NX processing unit (*"the Hub"*), and the onboard robotic platform (*"the Mouse"*). The user interacts with the system through a UI element, which sends commands to the Hub. The Hub processes the received commands and transmits motion commands to the Mouse's ESP32 microcontroller via Bluetooth.
Onboard the Mouse, the microcontroller interprets the received commands and drives twin DC motors through PWM enabled motor drivers. Additional commands are received from the Hub in autonomous mode, using object detection and pathfinding programs as well as video data from the ceiling-mounted webcam to direct the Mouse away from hazards. This video feed is also provided to the user through WebRTC.
User web interface
Hub
Mouse
WebRTC & Python
WebRTC is a protocol stack supported and tested on most browsers and most device, featuring rugged connectivity, optimization, and encryption. This is well suited to our use case, mainly because python packages like aiortc exist that support the whole network stack and work with OpenCV. The system also supports transporting small data packets for messages or for control data. Python provides portability between many different types of system, allowing flexibility with the Hub computer choice, inside the hardware constraints. Additional libraries are also available for machine vision and bluetooth – enabling autonomous routines and interactive pet play in real time.
Latency and Transport
Video latency is a crucial component in the feasibility of internet remote control. This is compounded by the two way time experienced when sending an action and waiting to see the result. We aim to keep this round trip time under 100 milliseconds to enhance usability with pets. The data channel stack consistently falls below the video transport times for a few reasons, so less critical to investigate. RTT figures were estimated with ICMP pings and RTT statistics from the data channel. The bluetooth (BLE) has limited throughput but very low latency, resulting in small packets with zero noticeable delay. The chart below tracks RTT ping test ranges.
NVIDIA Jetson DevKit
Enclosure Design & Modeling
The design constraints around pet safety introduce requirements for the enclosure. PETG filament has the advantage of being free of fumes and toxic artifacts of the printing process, while being resistant to brittleness & sharp shards. The motors are fully enclosed in dust covers to enhance the lifetime and safety of the robot. The battery is placed deep inside the assembly to prevent damadge and allow for a low center of gravity. The design must also have a profile that will resemble a mouse to be interactive with cats.
Result & Conclusion
The development of the low latency internet monitored and controlled robotic mouse successfully demonstrated how these systems can be used privately for utility or entertainment within remote spaces.

Appendix C

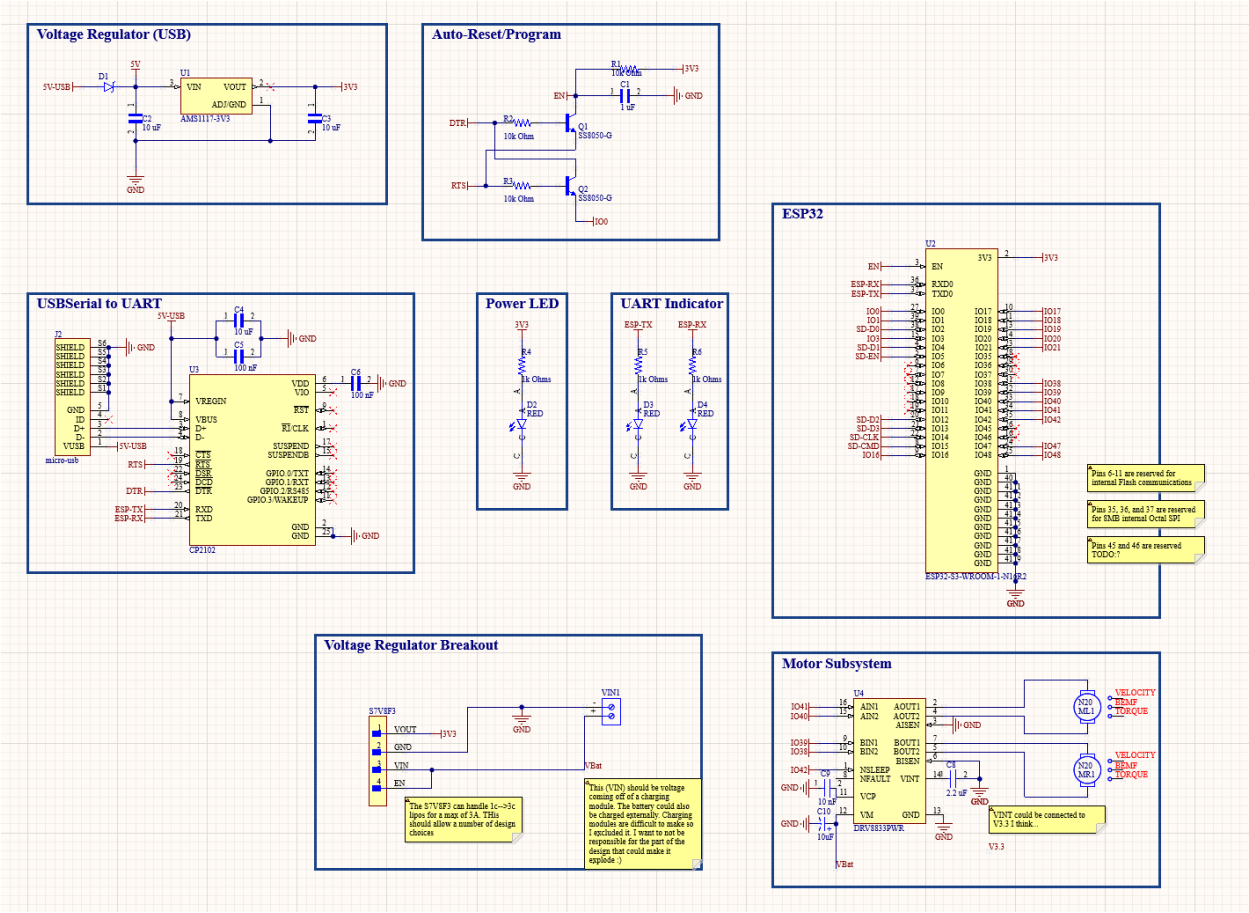
Python

```python
import asyncio
import json
import logging
import numpy as np
import cv2
import av
from aiohttp import web
from aiortc import RTCPeerConnection, RTCSessionDescription, VideoStreamTrack, RTCStatsReport

logging.basicConfig(level=logging.INFO)

ROOMS = {}

#resolution = (720, 480)

resolution = (1280, 720)

class CameraVideoTrack(VideoStreamTrack):
    """
    A VideoStreamTrack that pulls frames from a local OpenCV capture device.
    """
    def __init__(self, device=0):
        super().__init__()
        self.cap = cv2.VideoCapture(0)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, resolution[0])
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, resolution[1])


    async def recv(self):
        pts, time_base = await self.next_timestamp()

        ret, frame = self.cap.read()
        if not ret:
            # if capture fails, return a black frame
            print("failed to read camera")
            w,h  = resolution
            frame = (0 * np.ones((h, w, 3), dtype='uint8'))

        # convert BGR (OpenCV) to RGB for av
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        video_frame = av.VideoFrame.from_ndarray(frame_rgb, format="rgb24")
        video_frame.pts = pts
        video_frame.time_base = time_base
        return video_frame


async def offer(request):
```

```
    try:
        params = await request.json()
    except Exception:
        body = await request.text()
        print("Failed to parse JSON body:", body)
        raise

    if "sdp" not in params or "type" not in params:
        return web.json_response({"error": "Invalid offer"}, status=400)

    offer = RTCSessionDescription(sdp=params["sdp"], type=params["type"])
    pc = RTCPeerConnection()
    print("Created PeerConnection")

    pc.addTrack(CameraVideoTrack())

    @pc.on("datachannel")
    async def on_datachannel(channel):
        print("DataChannel created:", channel.label)

        @channel.on("message")
        async def on_message(message):
            stats = await pc.getStats()
            for stat in stats.values():
                if stat.type == "remote-inbound-rtp":
                    # Only care about roundTripTime if it exists
                    rtt = getattr(stat, "roundTripTime", None)
                    if rtt is not None:
                        print(f"Round Trip Time: {rtt} seconds")
            # message will be JSON like {"x":123,"y":456}
            print("Touch data:", message)


    @pc.on("connectionstatechange")
    async def on_state_change():
        print("Connection state:", pc.connectionState)
        if pc.connectionState == "failed":
            await pc.close()

    await pc.setRemoteDescription(offer)
    answer = await pc.createAnswer()
    await pc.setLocalDescription(answer)

    return web.json_response(
        {"sdp": pc.localDescription.sdp, "type": pc.localDescription.type}
    )
async def index(request):
    return web.FileResponse("public/index.html")

def run_server():
```

```
    app = web.Application()
    app.router.add_get("/", index)
    app.router.add_post("/offer", offer)
    app.router.add_static("/static/", path="public", name="static")

    web.run_app(app, host="0.0.0.0", port=8000)



if __name__ == "__main__":

    #run_server()
    import random
    import time
    import threading
    from Simulation import *

    simulation = Simulation("livingroom.jpg", (1280, 720))

    def random_inputs():
        """Run in background thread, feed random positions every 0.5 s."""
        while True:
            x = random.randint(0, 1280)
            y = random.randint(0, 720)
            simulation.update(x, y)
            time.sleep(0.5)

    # Start input thread
    threading.Thread(target=random_inputs, daemon=True).start()

    # Start Tkinter safely in main thread
    simulation.start()
```

Appendix D


Voltage Regulator (USB)
Auto-Reset/Program
ESP32
USBSerial to UART
Power LED
UART Indicator
Voltage Regulator Breakout
Motor Subsystem
Pins 6-11 are reserved for internal Flash communications
Pins 35, 36, and 37 are reserved for 8MB internal Octal SPI
Pins 45 and 46 are reserved TODO:?
The S7V8F3 can handle 1c->3c lipos for a max of 3A. THis should allow a number of design choices
This (VIN) should be voltage coming off of a charging module. The battery could also be charged externally. Charging modules are difficult to make so I excluded it. I want to not be responsible for the part of the design that could make it explode :)
VINT could be connected to V3.3 I think.

Appendix E

Python

```python
import asyncio
import threading
import time
import math
from bleak import BleakClient, BleakScanner

# usage:
# global variable: mouse = Mouse()
# when data channel created: mouse.connect() - this will block the thread and return bool
# when valid data is recieved: mouse.drive(joystick coords)
# send joystick data even when user not touching it so the mouse naturally slows down
# this is important for accidental disconnects
# addtionally we need to add a button to connect and disconnect from the mouse on the site
# automatically disconnect when the datachannel is closed




# ~40% transmission rate boost
USE_ACK = False
OFF_STATE = b'\x00'

#class to smooth out controller inputs and prevent device damage
class SlewLimiter2D:
    def __init__(self, max_rate):
        """
        max_rate: max change per second in vector magnitude
        """
        self.max_rate = max_rate
        self.x = 0.0
        self.y = 0.0
        self.last_time = time.time()

    def update(self, target_x, target_y):
        now = time.time()
        dt = now - self.last_time
        self.last_time = now

        # desired change
        dx = target_x - self.x
        dy = target_y - self.y

        dist = math.sqrt(dx*dx + dy*dy)
        max_step = self.max_rate * dt
```

```
        if dist > max_step and dist > 0:
            scale = max_step / dist
            dx *= scale
            dy *= scale

        self.x += dx
        self.y += dy

        return self.x, self.y

#important to stop moving if we disconnect
class MotionFallback:
    def __init__(self, frequency, threshold):
        self.last_time = time.time()
        self.f = frequency
        self.t = threshold
        self.started = False

    def start(self):
        self.started = True
        self.last_time = time.time()

    def should_turn_off_mouse(self):
        if self.started:
            if time.time() > self.last_time + self.t:
                print("Control data halted, falling back into off state")
                return True
            else:
                return False




# mouse class
class Mouse:
    #hardcoded parameters
    def __init__(self):
        self.max_roc = .2
        self.device_name = "ToyESP32"
        self.watchdog_frequency = 5 # Hz
        self.uart_uuid = "6e400002-b5a3-f393-e0a9-e50e24dcca9e"
        self.fallback_timeout = 2 # S

        self.limiter = SlewLimiter2D(self.max_roc)

        self.fallback = MotionFallback(self.watchdog_frequency, self.fallback_timeout)

        self.loop = asyncio.new_event_loop()
        self.thread = threading.Thread(target=self._run_loop, daemon=True)
        self.thread.start()
```

```
        self.client = None
        self.address = None

    def _run_loop(self):
        asyncio.set_event_loop(self.loop)
        self.loop.run_forever()

    # ----------- ASYNC -----------

    async def _find_device(self):
        devices = await BleakScanner.discover()
        for d in devices:
            if d.name and self.device_name in d.name:
                return d.address
        return None

    async def _connect_async(self, retry = True):
        addr = await self._find_device()

        if self.client and self.client.is_connected:
            print("Already connected")
            return

        if addr is None:
            print("Device not found, sending fallback disconnect packet")
            await self._send_disconnect_packet()
            if retry:
                self._connect_async(False)
            return False

        self.address = addr
        self.client = BleakClient(self.address)
        await self.client.connect()
        # turn mouse off
        await self.client.write_gatt_char(self.uart_uuid, OFF_STATE)
        return True

    async def _send_disconnect_packet(self):
        try:
            if self.client and self.client.is_connected:
                await self.client.write_gatt_char(self.uart_uuid, OFF_STATE)
                await self.client.disconnect()
        except Exception as e:
            print("Fallback disconnect failed:", e)

    async def _write_async(self, data: bytes):
        await self.client.write_gatt_char(self.uart_uuid, data, response = USE_ACK)

    async def _disconnect_async(self):
```

```
    if self.client:
        await self.client.disconnect()


# ----------- BLOCKING -----------
# call these from other threads

def connect(self):
    future = asyncio.run_coroutine_threadsafe(
        self._connect_async(), self.loop
    )
    return future.result()

def write(self, data: bytes):
    if self.client and self.client.is_connected:
        future = asyncio.run_coroutine_threadsafe(
            self._write_async(data), self.loop
        )
        try:
            result = future.result()
        except OSError as e:
            print("The ble write timed out! Reconnecting...")
            self.connect()
        return
    else:
        print("Mouse is not connected")


def disconnect(self):
    future = asyncio.run_coroutine_threadsafe(
        self._disconnect_async(), self.loop
    )
    return future.result()

def drive(self, joy_x, joy_y):
    # slew limiter
    x, y = self.limiter.update(joy_x, joy_y)

    # curve
    y = y
    x = x **3

    # arcade drive
    l = y + x
    r = y - x

    # normalize
    m = max(abs(l), abs(r), 1.0)
    l = l / m
    r = r / m
```

```
        # convert to command
        cmd = 0
        cmd |= (int(l * 7) & 0b111)
        cmd |= (int(r * 7) & 0b111) << 4
        cmd |= (1 if l < 0 else 0) << 3
        cmd |= (1 if r < 0 else 0) << 7

        # send over ble
        print(f"[=={bin(cmd)}==]")
        self.write(bytes([cmd]))

# demo program - for led toggling firmware
if __name__ == "__main__":
    ble = Mouse()
    f = 15 # Hz
    ble.connect()
    for i in range(300):
        time.sleep(1/f)
        if i % 2 == 0:
            ble.write(b'1')
        else:
            ble.write(b'0')
    ble.disconnect()
```

Appendix F

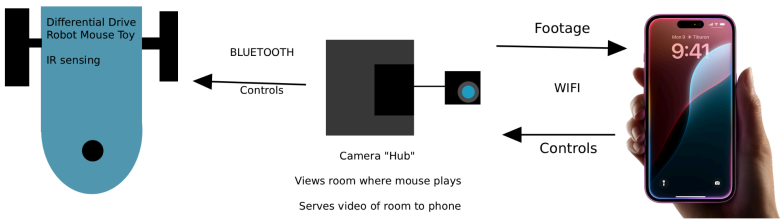
Meeting Notes 9/5/25

Taken by Wyatt

Discussion

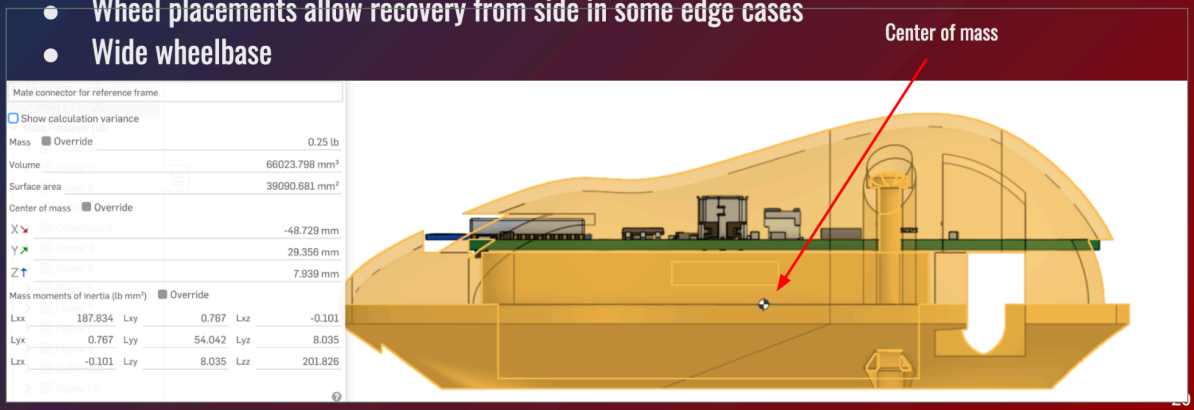
- How should the video stream be performed
  - Jami, UDP, VNC
- What kind of device should the camera be?
  - NVDIA Jetson
  - Raspberry Pi
  - Not ESP32-p4
  - Could start with just a computer
  - Need high fidelity
  - Possible USB amazon devkits IMX219
- Mounting device to ceiling
  - Onboard on the mouse toy? Prolly not
- Controls integrated into app?
  - Should controls be separate from video stream?
- Mouse design
  - Possible RC car base, upgrade to custom hardware
- Good devices
  - Jetson Nano
    - High resolution, processing, supported
- Behzad arrived 12:52
  - Narrowing Scope
    - Cat toy and camera are interactive and remotely operated
    - Ideally a toy mouse
    - Phone as the device to control everything
    - Additional questions
      - How can the toy be fully operated when obstructed from camera view
    - Advisor notes on networking
      - TCP/IP standard protocol
      - Control over network
      - Advising against p2p
      - There will be latency
  - Buy the car toy, disguise as a mouse, dissect controller, connect to microcontroller
- Wyatt's note: WebRTC is an amazing option for sending controls + video, minimal client infrastructure, supports HTTPS, encryption needed for port forwarding or can use VPN

Task Name	Wk 1	Wk 2	Wk 3	Wk 4	Wk 5	Wk 6	Wk 7	Wk 8	Wk 9	Wk 10	Wk 11	Wk 12	Wk 13	Wk 14	Wk 15	Wk 16	Wk 17	Wk 18	Wk 19	Wk 20	Wk 21	Wk 22	Wk 23	Wk 24	Wk 25
Refine objectives, Clarify design and goals																									
Determine robot purpose + function																									
Create Project Proposal																									
Establish Video Streaming																									
Finalize mouse decisions																									
Simple mouse simulation																									
Add Controls to interface																									
Design and order parts for mouse																									
Complete hub implimentation																									
Develop mouse appearance and body																									
Polish user interface																									
Create apps for iOS/Android																									
Test and refine features																									



Self-Righting

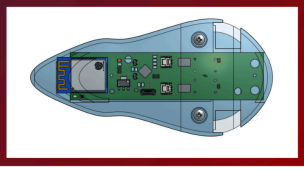
- The mouse toy should be resistant to being turtled (flipped over)
- Battery placement is a big factor - as low as possible
- Wheel placements allow recovery from side in some edge cases
- Wide wheelbase



Wyatt

Safety and durability

- Fully enclosed, impact-resistant design with no exposed electronics or small detachable parts
- Low-voltage, regulated power system with redesigned supply to prevent brownout and overheating
- Safe behavior (auto stop) upon connection loss
- Non-toxic and pet resistant material (PETG)



Wyatt

18

Enclosure - Material

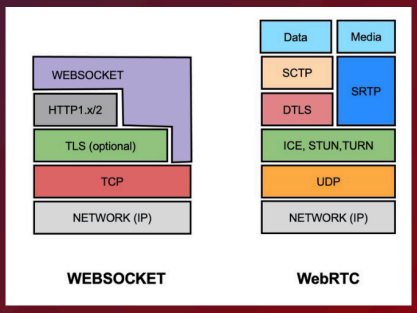
Material	Durability	Non-Toxicity	Ease	Overall Safety
Weight	5	3	2	--
PLA	6	9	9	75%
TPU	8	9	5	77%
PETG	9	9	9	90%
Nylon	8	7	2	65%
ABA	7	1	5	48%
ABS	6	1	5	43%

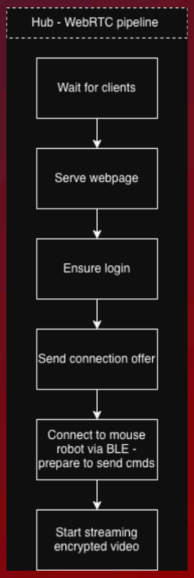
Justification

19

Serving Video

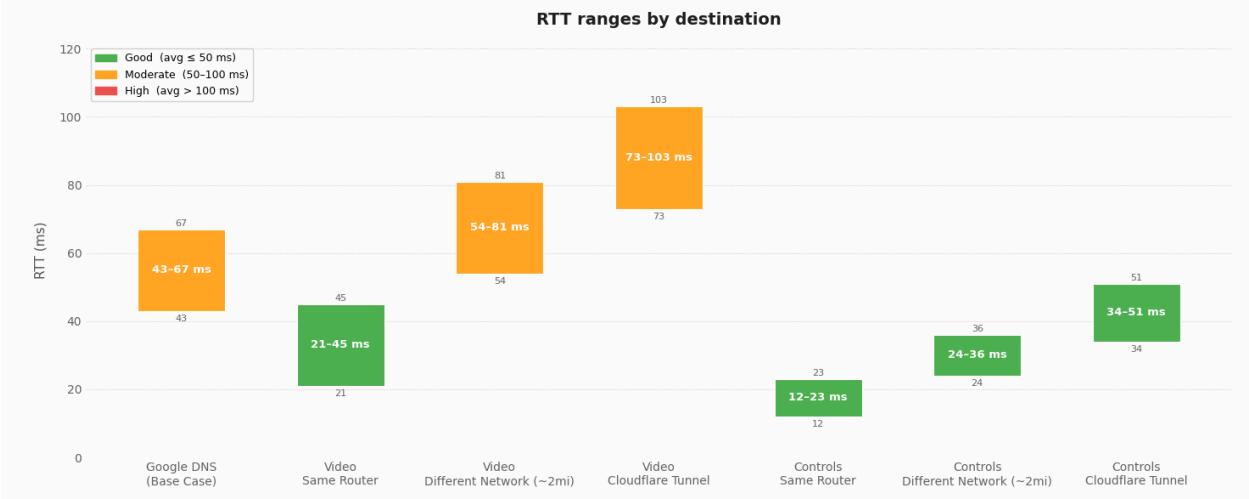
- The NVIDIA Xavier development kit
  - Single board computer with USB
  - Hardware acceleration support for H.264 video encoding
  - High bus bandwidth ~30GB/S
  - Available linux distributions and packages
- AIORTP Library - WebRTC stack
  - Encrypted video feed (SRTP)
  - Offers
  - Datachannels
  - ICE - STUN - TURN
  - DTLS - custom transport layer security
  - Browser compatibility
- Python
  - Portable and consistent
  - Many libraries





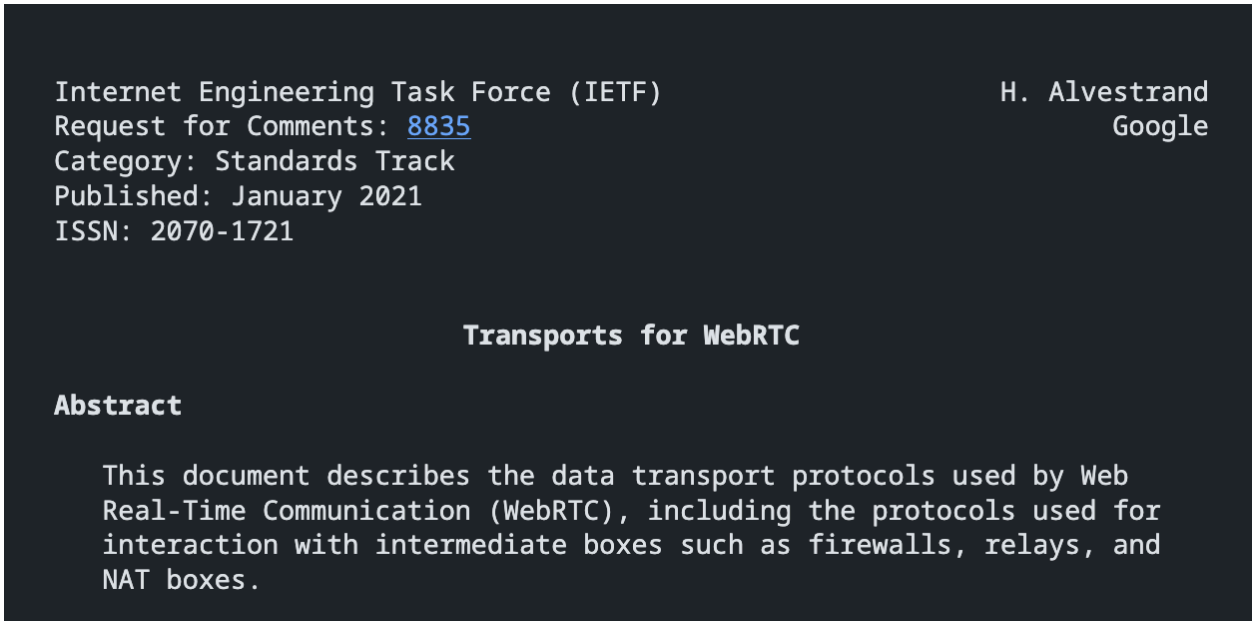
Wyatt

34



Appendix G

IETF, “Overview: Real-Time Protocols for Browser-Based Applications,” RFC 8825, Jan. 2021.
IETF, “JavaScript Session Establishment Protocol (JSEP),” RFC 8829, Jan. 2021.
IETF, “WebRTC Security Architecture,” RFC 8827, Jan. 2021.
IETF, “RTP: A Transport Protocol for Real-Time Applications,” RFC 3550, Jul. 2003.
IETF, “The Secure Real-time Transport Protocol (SRTP),” RFC 3711, Mar. 2004.
IETF, “DTLS Extension to Establish Keys for SRTP,” RFC 5764, May 2010.
IETF, “Media Transport and Use of RTP in WebRTC,” RFC 8834, Jan. 2021.



Internet Engineering Task Force (IETF) H. Alvestrand
Request for Comments: 8835 Google
Category: Standards Track
Published: January 2021
ISSN: 2070-1721

Transports for WebRTC

Abstract

This document describes the data transport protocols used by Web Real-Time Communication (WebRTC), including the protocols used for interaction with intermediate boxes such as firewalls, relays, and NAT boxes.