Project Ideas (Jack Couture)

Smart Irrigation System

Description: Develop a cost-effective, automated irrigation system that uses soil moisture sensors, weather forecasts, and IoT technology to optimize watering schedules for gardens or small farms.
Hardware: Soil moisture sensors, microcontroller (like Arduino or Raspberry Pi), water solenoids and valves.
Software: Application to monitor and control the system, integration with weather APIs to optimize watering based on forecasts.

Automated Recycling Sorter

Description: Build a machine that can sort recyclables (like plastics, metals, and glass) using sensors and basic robotic arms.
Hardware: Conveyor belt, sensors (like infrared for plastic or magnets for metals), robotic arms for sorting.
Software: Control software for the sorting system, possibly incorporating machine learning for material identification.

Smart Traffic Light Controller

Description: Develop a traffic light control system that adapts in real-time to traffic conditions, potentially integrating with emergency services to prioritize vehicle flow.
Hardware: Microcontrollers, cameras or sensors for traffic detection, model traffic lights for demonstration.
Software: Algorithm for dynamic traffic management, software to process input from traffic sensors and control lights accordingly.

Automated Public Library System

Description: Create an automated system to manage book check-ins and check-outs, reservations, and sorting at public libraries. This system could use RFID technology for tracking books and a robotic system for sorting and shelving returned items.
Hardware: RFID readers and tags, robotic arms for handling books, conveyor systems for moving books within the library.
Software: Library management software to track book locations, check-out and return statuses, and user reservations. Integration with existing library databases and user interface development for self-service kiosks.

Intelligent Document Organizer

Description: Build a device that can automatically scan, categorize, and file physical documents into digital formats using OCR and custom sorting algorithms.
Hardware: Document scanner, automated filing system with mechanical components for handling paper.
Software: OCR software to convert scanned images to text, categorization algorithms based on content analysis, user interface for accessing and searching documents.

Final Project Ideas (Entire Team Present)

Machine Intelligence Mimicking Industrial Choreography (MIMIC):
Inefficient factory environments
Design a tool to teach the robotic arm's hand movement
Build an arm or find a modifiable arm
Do research on the arm

Car To Car Collision Avoidance (C2C):
Around 3k deaths in the US due to distracted driving per year
Sensor on cars to provide alerts through GUIs
Tool to assist the driver
Concern with response time
Proof of concept
Easier to achieve then scale from there

Guidance Utility for Impaired Daily Experiences (GUIDE):
Over 340k Americans have no vision
Sense nearby organic or inorganic objects
Alert through an app using auditory cues
Sensors/Cameras/LIDAR

Accelerated Traffic Light Automation System (ATLAS):
28 million tons of CO_2 emitted idling
Lights are not optimized
Accelerate traffic light states
Computer vision
They are making a new system not using traffic lights

Schematics Ideas (Entire Team Present)

Hollow Walking Stick Collapsible with Wires Inside

A sensor at the front will detect the cane so a blind spot is needed

A sensor at the end will make it heavy

The sensor could be:
LIDAR
Depth Camera
Infrared Pulses
(Research affordability)

Audio Cues could impair hearing

Vibrations or pulse in the handle
The speed of the pulse increases as you get closer
Rumbler from a controller in the handle

Textured Rubber handle

Potentially extrude the top out to make room for electronics

Potentially expand the end of the stick's base for electronics

C++ or Python for communication

Light for at night to prevent injury and provide more visibility to others

Plug cane into the computer to change settings
- Rumble strength
- Sensitivity
- Turn the light on and off

24oz up to 38oz

Depending on weight we could use a smaller handheld device to limit the length allowing for more weight

* SLAM algorithm allows for mapping in a database (Location Data) *
Could be complex and a stretch goal

Configuration file with default values
- Ability to reset to default
- BATCH FILE :)

Storage through Arduino to prevent power issues as well as memory

* Bluetooth Functionality *
Research required

3D-printed casings for sensors

Tasks for 9/9 Week (Jack Couture)

Convert Python file to C++

Implement a script to take a CSV of data from the depth camera and create an image

Test this using a virtual machine connected to the depth camera or CSV files

Write the Introduction for the Project Proposal

CSV Conversion Code Ideas (Jack Couture)

VERSION #1

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <iomanip> // for hex

// Function to convert hex string to integer
int hexToInt(const std::string& hexStr) {
    int value;
    std::stringstream ss;
    ss << std::hex << hexStr;
    ss >> value;
    return value;
}

int main() {
    std::ifstream file("hex_log_20240910_173153.csv"); //Sample Image
    std::string line;

    // Store image data as a 2D array
    std::vector<std::vector<int>> image_data;

    while (std::getline(file, line)) {
        std::stringstream ss(line);
        std::string byte;
        std::vector<int> row_data;

        // Skip the first two columns (Elapsed Time and Offset)
        for (int i = 0; i < 2; i++) {
            std::getline(ss, byte, ',');
        }

        // Read the byte data (Byte0 to Byte15)
        for (int i = 0; i < 16; i++) {
            std::getline(ss, byte, ',');
            int value = hexToInt(byte); // Convert hex string to integer
            row_data.push_back(value);
        }

        image_data.push_back(row_data); // Store the row in image data
    }

    // Define the output PPM file
    std::ofstream imageFile("output_image.ppm");

    // PPM header
    int rows = image_data.size();
    int cols = image_data[0].size();
    imageFile << "P3\n" << cols << " " << rows << "\n255\n"; // PPM header (P3
format, max RGB value 255)

    // Write pixel data to PPM file
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int pixel_value = image_data[i][j];
            // For simplicity, we map the grayscale value to RGB by setting all
channels to the same value.
            imageFile << pixel_value << " " << pixel_value << " " << pixel_value
<< "  ";
        }
```

```
        imageFile << "\n";
    }

    imageFile.close();
    std::cout << "PPM image saved as output_image.ppm" << std::endl;
    return 0;
}
```

VERSION #2

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <iomanip> // for hex

// Function to convert hex string to integer
int hexToInt(const std::string& hexStr) {
    int value;
    std::stringstream ss;
    ss << std::hex << hexStr;
    ss >> value;
    return value;
}

int main() {
    std::ifstream file("hex_log_20240910_173153.csv"); //Sample CSV (NEED TO
EDIT FOR OTHER CSVs) *Should prob take input instead IDK
    std::string line;

    // Store image data as a 2D array
    std::vector<std::vector<int>> image_data;

    while (std::getline(file, line)) {
        std::stringstream ss(line);
        std::string byte;
        std::vector<int> row_data;

        // Skip the first two columns (Elapsed Time and Offset)
        for (int i = 0; i < 2; i++) {
            std::getline(ss, byte, ',');
        }

        // Read the byte data (Byte0 to Byte15)
        for (int i = 0; i < 16; i++) {
            std::getline(ss, byte, ',');
            int value = hexToInt(byte); // Convert hex string to integer
            row_data.push_back(value);
        }

        image_data.push_back(row_data); // Store the row in image data
    }
```

```
    // Define the output PPM file
    std::ofstream imageFile("output_image.ppm");

    // PPM header
    int rows = image_data.size();
    int cols = image_data[0].size();
    imageFile << "P3\n" << cols << " " << rows << "\n255\n"; // PPM header (P3
format, max RGB value 255)

    // Write pixel data to PPM file
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int pixel_value = image_data[i][j];
            // Map the grayscale value to RGB by setting all channels to the
same value
            imageFile << pixel_value << " " << pixel_value << " " << pixel_value
<< "  ";
        }
        imageFile << "\n";
    }

    imageFile.close();
    std::cout << "PPM image saved as output_image.ppm" << std::endl;
    return 0;
}
```

SHOULD PROBABLY:
Take input for CSV file
Not output as PPM because converting takes a while

ISSUE:
I made an image when it should be a collection of frames(images)

VERSION #3

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <iomanip> // for hex
#include <chrono>
#include <thread>

// Function to convert hex string to integer
int hexToInt(const std::string& hexStr) {
    int value;
    std::stringstream ss;
    ss << std::hex << hexStr;
    ss >> value;
    return value;
}

// Function to save a frame as a PPM image
void saveFrameAsPPM(const std::vector<std::vector<int>>& frame_data, int
frame_number) {
    std::string filename = "frame_" + std::to_string(frame_number) + ".ppm";
```

```
    std::cout << "Attempting to create file: " << filename << std::endl; //
Debug

    std::ofstream imageFile(filename);

    if (!imageFile) {
        std::cerr << "Error: Could not create PPM file " << filename <<
std::endl;
        return;
    }

    std::cout << "File " << filename << " created successfully." <<
std::endl; // Debug

    int rows = frame_data.size();
    int cols = frame_data[0].size();

    // Write PPM header
    imageFile << "P3\n" << cols << " " << rows << "\n255\n";

    // Write pixel data
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int pixel_value = frame_data[i][j];
            imageFile << pixel_value << " " << pixel_value << " " << pixel_value
<< "  ";
        }
        imageFile << "\n";
    }

    imageFile.close();
    std::cout << "Frame " << frame_number << " saved as " << filename <<
std::endl;
}

int main() {
    std::ifstream file("hex_log_20240910_173153.csv");
    if (!file.is_open()) {
        std::cerr << "Error: Could not open the CSV file!" << std::endl;
        return 1;
    }

    std::string line;

    // Skip the first line (header row)
    std::getline(file, line);

    std::vector<std::vector<int>> frame_data;
    int frame_number = 0;
    double previous_time = 0.0;

    // Process CSV data
    while (std::getline(file, line)) {
        std::stringstream ss(line);
        std::string byte;
        std::vector<int> row_data;
        double elapsed_time;

        // Read Elapsed Time and Offset
        std::getline(ss, byte, ',');
        elapsed_time = std::stod(byte);  // Convert Elapsed Time to double
        std::getline(ss, byte, ',');     // Skip Offset
```

```
        // Read Byte0 to Byte15
        for (int i = 0; i < 16; i++) {
            std::getline(ss, byte, ',');
            int value = hexToInt(byte); // Convert hex string to integer
            row_data.push_back(value);
        }

        // Add the row to the current frame data
        frame_data.push_back(row_data);

        // Check if we need to save the frame (if time difference is detected)
        if (elapsed_time != previous_time && !frame_data.empty()) {
            // Save the current frame as an image
            saveFrameAsPPM(frame_data, frame_number);
            frame_number++;

            // Sleep for the time difference between frames to simulate video
timing
            double time_diff = elapsed_time - previous_time;

std::this_thread::sleep_for(std::chrono::milliseconds(static_cast<int>(time_diff
* 1000)));

            // Clear the frame data for the next frame
            frame_data.clear();
        }

        previous_time = elapsed_time;
    }

    std::cout << "Frames saved." << std::endl;
    return 0;
}
```

Produces frames but I do not know what frame rate or sized was used to currently cannot be converted to a video using ffmpeg

VERSION #4

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <iomanip> // for hex

// Function to convert hex string to integer
int hexToInt(const std::string& hexStr) {
    int value;
    std::stringstream ss;
    ss << std::hex << hexStr;
```

```
    ss >> value;
    return value;
}

// Function to save a frame as a PPM image
void saveFrameAsPPM(const std::vector<std::vector<int>>& frame_data, int
frame_number) {
    std::string filename = "frame_" + std::to_string(frame_number) + ".ppm";

    std::ofstream imageFile(filename);

    if (!imageFile) {
        std::cerr << "Error: Could not create PPM file " << filename <<
std::endl;
        return;
    }

    const int rows = 16;  // Fixed number of rows
    const int cols = 16;  // Fixed number of columns

    // Write PPM header
    imageFile << "P3\n" << cols << " " << rows << "\n255\n";

    // Write pixel data
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int pixel_value = frame_data[i % frame_data.size()][j %
frame_data[0].size()];
            imageFile << pixel_value << " " << pixel_value << " " << pixel_value
<< "  ";
        }
        imageFile << "\n";
    }

    imageFile.close();
}

// Main function to read CSV and generate frames
int main() {
    std::ifstream file("hex_log_20240910_173153.csv");
    if (!file.is_open()) {
        std::cerr << "Error: Could not open the CSV file!" << std::endl;
        return 1;
    }

    std::string line;
    std::vector<std::vector<int>> frame_data;
    int frame_number = 0;

    // Skip the first line (header row)
    std::getline(file, line);

    // Process CSV data
    while (std::getline(file, line)) {
        std::stringstream ss(line);
        std::string byte;
        std::vector<int> row_data;

        // Read Elapsed Time and Offset (skip them for now)
        std::getline(ss, byte, ',');  // Elapsed Time
        std::getline(ss, byte, ',');  // Offset

        // Read Byte0 to Byte15
        for (int i = 0; i < 16; i++) {
```

```
            std::getline(ss, byte, ',');
            int value = hexToInt(byte); // Convert hex string to integer
            row_data.push_back(value);
        }

        // Add the row to the current frame data
        frame_data.push_back(row_data);

        // Save the current frame as an image every 16 rows
        if (frame_data.size() == 16) {
            saveFrameAsPPM(frame_data, frame_number);
            frame_number++;
            frame_data.clear();  // Clear for the next frame
        }
    }

    return 0;
}
```

Prototype for Converting Code (Jack Couture)

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <iomanip> // for hex

// Function to convert hex string to integer
int hexToInt(const std::string& hexStr) {
    int value;
    std::stringstream ss;
    ss << std::hex << hexStr;
    ss >> value;
    return value;
}

// Function to save a frame as a PPM image
void saveFrameAsPPM(const std::vector<std::vector<int>>& frame_data, int
frame_number) {
    std::string filename = "frame_" + std::to_string(frame_number) + ".ppm";

    std::ofstream imageFile(filename);

    if (!imageFile) {
        std::cerr << "Error: Could not create PPM file " << filename <<
std::endl;
        return;
    }

    const int rows = 100;  // Fixed number of rows
    const int cols = 100;  // Fixed number of columns

    // Write PPM header
    imageFile << "P3\n" << cols << " " << rows << "\n255\n";

    // Write pixel data
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int pixel_value = frame_data[i % frame_data.size()][j %
frame_data[0].size()];
            imageFile << pixel_value << " " << pixel_value << " " << pixel_value
<< "  ";
        }
        imageFile << "\n";
    }

    imageFile.close();
}

// Main function to read CSV and generate frames
int main() {
    std::ifstream file("hex_log_20240910_173153.csv");
    if (!file.is_open()) {
        std::cerr << "Error: Could not open the CSV file!" << std::endl;
        return 1;
    }

    std::string line;
    std::vector<std::vector<int>> frame_data;
    int frame_number = 0;

    // Skip the first line (header row)
    std::getline(file, line);
```

```
    // Process CSV data
    while (std::getline(file, line)) {
        std::stringstream ss(line);
        std::string byte;
        std::vector<int> row_data;

        // Read Elapsed Time and Offset (skip them for now)
        std::getline(ss, byte, ',');  // Elapsed Time
        std::getline(ss, byte, ',');  // Offset

        // Read Byte0 to Byte15
        for (int i = 0; i < 16; i++) {
            std::getline(ss, byte, ',');
            int value = hexToInt(byte); // Convert hex string to integer
            row_data.push_back(value);
        }

        // Add the row to the current frame data
        frame_data.push_back(row_data);

        // Save the current frame as an image every 100
        if (frame_data.size() == 100) {
            saveFrameAsPPM(frame_data, frame_number);
            frame_number++;
            frame_data.clear();  // Clear for the next frame
        }
    }

    return 0;
}
```

```
cl /EHsc convertCSV.cpp
```

```
convertCSV.exe
```

```
ffmpeg -r 30 -i frame_%d.ppm -vcodec libx264 -pix_fmt yuv420p output.mp4
```

How To Benchmark Cameras (Jack L and Jack C)

Have a measured distance away object

Measure the distance with LiDAR

Measure the distance with depth camera

Repeat this at different angles and distances

Plot the results compared to the measured value

Prototype for Converting Code (Jack Couture)

#include <iostream> #include <fstream> #include <sstream> #include <vector> #include <iomanip> // for hex

// Function to convert hex string to integer int hexToInt(const std::string& hexStr) { int value; std::stringstream ss; ss « std::hex « hexStr; ss » value; return value; }

// Function to save a frame as a PPM image void saveFrameAsPPM(const std::vector<std::vector<int»& frame_data, int frame_number) { std::string filename = "frame_" + std::to_string(frame_number) + ".ppm";

std::ofstream imageFile(filename);

if (!imageFile) { std::cerr « "Error: Could not create PPM file " « filename « std::endl; return; }

const int rows = 100; // Fixed number of rows const int cols = 100; // Fixed number of columns

// Write PPM header imageFile « "P3\n" « cols « " " « rows « "\n255\n";

// Write pixel data for (int i = 0; i < rows; i++) { for (int j = 0; j < cols; j++) { int pixel_value = frame_data[i % frame_data.size()][j % frame_data[0].size()]; imageFile « pixel_value « " " « pixel_value « " " « pixel_value « " "; } imageFile « "\n"; }

imageFile.close(); }

// Main function to read CSV and generate frames int main() { std::ifstream file("hex_log_20240910_173153.csv"); if (!file.is_open()) { std::cerr « "Error: Could not open the CSV file!" « std::endl; return 1; }

std::string line; std::vector<std::vector<int» frame_data; int frame_number = 0;

// Skip the first line (header row) std::getline(file, line);

// Process CSV data while (std::getline(file, line)) { std::stringstream ss(line); std::string byte; std::vector<int> row_data;

// Read Elapsed Time and Offset (skip them for now) std::getline(ss, byte, ','); // Elapsed Time std::getline(ss, byte, ','); // Offset

// Read Byte0 to Byte15 for (int i = 0; i < 16; i++) { std::getline(ss, byte, ','); int value = hexToInt(byte); // Convert hex string to integer row_data.push_back(value); }

// Add the row to the current frame data frame_data.push_back(row_data);

// Save the current frame as an image every 100 if (frame_data.size() == 100) { saveFrameAsPPM(frame_data, frame_number); frame_number++; frame_data.clear(); // Clear for the next frame } }

```
return 0; }
cl /EHsc convertCSV.cpp
convertCSV.exe
ffmpeg -r 30 -i frame_%d.ppm -vcodec libx264 -pix_fmt yuv420p output.mp4
```

Schematics Ideas (Entire Team Present)

Hollow Walking Stick Collapsible with Wires Inside

A sensor at the front will detect the cane so a blind spot is needed

A sensor at the end will make it heavy

The sensor could be: LIDAR Depth Camera Infrared Pulses (Research affordability)

Audio Cues could impair hearing

Vibrations or pulse in the handle The speed of the pulse increases as you get closer Rumbler from a controller in the handle

Textured Rubber handle

Potentially extrude the top out to make room for electronics

Potentially expand the end of the stick's base for electronics

C++ or Python for communication

Light for at night to prevent injury and provide more visibility to others

Plug cane into the computer to change settings Rumble strength Sensitivity Turn the light on and off

24oz up to 38oz

Depending on weight we could use a smaller handheld device to limit the length allowing for more weight

* SLAM algorithm allows for mapping in a database (Location Data) * Could be complex and a stretch goal

Configuration file with default values Ability to reset to default BATCH FILE :)

Storage through Arduino to prevent power issues as well as memory

* Bluetooth Functionality * Research required

3D-printed casings for sensors

Final Project Ideas (Entire Team Present)

Machine Intelligence Mimicking Industrial Choreography (MIMIC): Inefficient factory environments Design a tool to teach the robotic arm's hand movement Build an arm or find a modifiable arm Do research on the arm

Car To Car Collision Avoidance (C2C): Around 3k deaths in the US due to distracted driving per year Sensor on cars to provide alerts through GUIs Tool to assist the driver Concern with response time Proof of concept Easier to achieve then scale from there

Guidance Utility for Impaired Daily Experiences (GUIDE): Over 340k Americans have no vision Sense nearby organic or inorganic objects Alert through an app using auditory cues Sensors/Cameras/LIDAR

Accelerated Traffic Light Automation System (ATLAS): 28 million tons of CO2 emitted idling Lights are not optimized Accelerate traffic light states Computer vision They are making a new system not using traffic lights

From C++ to Python (Others Present)

Python has shown it can keep up with C++ performance

The Python libraries would make coding a lot easier

More comfort with Python

Hardware changes make Python the better option

CSV Conversion Code Ideas (Jack Couture)

VERSION #1

#include <iostream> #include <fstream> #include <sstream> #include <vector> #include <iomanip> // for hex

// Function to convert hex string to integer int hexToInt(const std::string& hexStr) { int value; std::stringstream ss; ss « std::hex « hexStr; ss » value; return value; }

int main() { std::ifstream file("hex_log_20240910_173153.csv"); //Sample Image std::string line;

// Store image data as a 2D array std::vector<std::vector<int» image_data;

while (std::getline(file, line)) { std::stringstream ss(line); std::string byte; std::vector<int> row_data;

// Skip the first two columns (Elapsed Time and Offset) for (int i = 0; i < 2; i++) { std::getline(ss, byte, ','); }

// Read the byte data (Byte0 to Byte15) for (int i = 0; i < 16; i++) { std::getline(ss, byte, ','); int value = hexToInt(byte); // Convert hex string to integer row_data.push_back(value); }

image_data.push_back(row_data); // Store the row in image data }

// Define the output PPM file std::ofstream imageFile("output_image.ppm");

// PPM header int rows = image_data.size(); int cols = image_data[0].size(); imageFile « "P3\n" « cols « " " « rows « "\n255\n"; // PPM header (P3 format, max RGB value 255)

// Write pixel data to PPM file for (int i = 0; i < rows; i++) { for (int j = 0; j < cols; j++) { int pixel_value = image_data[i][j]; // For simplicity, we map the grayscale value to RGB by setting all channels to the same value. imageFile « pixel_value « " " « pixel_value « " " « pixel_value « " "; } imageFile « "\n"; }

imageFile.close(); std::cout « "PPM image saved as output_image.ppm" « std::endl; return 0; }

VERSION #2

#include <iostream> #include <fstream> #include <sstream> #include <vector> #include <iomanip> // for hex

// Function to convert hex string to integer int hexToInt(const std::string& hexStr) { int value; std::stringstream ss; ss « std::hex « hexStr; ss » value; return value; }

int main() { std::ifstream file(”hex_log_20240910_173153.csv”); //Sample CSV (NEED TO EDIT FOR OTHER CSVs) *Should prob take input instead IDK std::string line;

// Store image data as a 2D array std::vector<std::vector<int» image_data;

while (std::getline(file, line)) { std::stringstream ss(line); std::string byte; std::vector<int> row_data;

// Skip the first two columns (Elapsed Time and Offset) for (int i = 0; i < 2; i++) { std::getline(ss, byte, ’,’); }

// Read the byte data (Byte0 to Byte15) for (int i = 0; i < 16; i++) { std::getline(ss, byte, ’,’); int value = hexToInt(byte); // Convert hex string to integer row_data.push_back(value); }

image_data.push_back(row_data); // Store the row in image data }

// Define the output PPM file std::ofstream imageFile(”output_image.ppm”);

// PPM header int rows = image_data.size(); int cols = image_data[0].size(); imageFile « ”P3\n” « cols « ” ” « rows « ”\n255\n”; // PPM header (P3 format, max RGB value 255)

// Write pixel data to PPM file for (int i = 0; i < rows; i++) { for (int j = 0; j < cols; j++) { int pixel_value = image_data[i][j]; // Map the grayscale value to RGB by setting all channels to the same value imageFile « pixel_value « ” ” « pixel_value « ” ” « pixel_value « ” ”; } imageFile « ”\n”; }

imageFile.close(); std::cout « ”PPM image saved as output_image.ppm” « std::endl; return 0; }

SHOULD PROBABLY: Take input for CSV file Not output as PPM because converting takes a while

ISSUE: I made an image when it should be a collection of frames(images)

VERSION #3

#include <iostream> #include <fstream> #include <sstream> #include <vector> #include <iomanip> // for hex #include <chrono> #include <thread>

// Function to convert hex string to integer int hexToInt(const std::string& hexStr) { int value; std::stringstream ss; ss « std::hex « hexStr; ss » value; return value; }

// Function to save a frame as a PPM image void saveFrameAsPPM(const std::vector<std::vector<int»& frame_data, int frame_number) { std::string filename = ”frame_” + std::to_string(frame_number) + ”.ppm”;

std::cout « ”Attempting to create file: ” « filename « std::endl; // Debug

std::ofstream imageFile(filename);

if (!imageFile) { std::cerr « ”Error: Could not create PPM file ” « filename « std::endl; return; }

std::cout « ”File ” « filename « ” created successfully.” « std::endl; // Debug

int rows = frame_data.size(); int cols = frame_data[0].size();

// Write PPM header imageFile « ”P3\n” « cols « ” ” « rows « ”\n255\n”;

// Write pixel data for (int i = 0; i < rows; i++) { for (int j = 0; j < cols; j++) { int pixel_value = frame_data[i][j]; imageFile « pixel_value « ” ” « pixel_value « ” ” « pixel_value « ” ”; } imageFile « ”\n”; }

imageFile.close(); std::cout « ”Frame ” « frame_number « ” saved as ” « filename « std::endl; }

int main() { std::ifstream file(”hex_log_20240910_173153.csv”); if (!file.is_open()) { std::cerr « ”Error: Could not open the CSV file!” « std::endl; return 1; }

std::string line;

// Skip the first line (header row) std::getline(file, line);

std::vector<std::vector<int» frame_data; int frame_number = 0; double previous_time = 0.0;

// Process CSV data while (std::getline(file, line)) { std::stringstream ss(line); std::string byte; std::vector<int> row_data; double elapsed_time;

// Read Elapsed Time and Offset std::getline(ss, byte, ’,’); elapsed_time = std::stod(byte); // Convert Elapsed Time to double std::getline(ss, byte, ’,’); // Skip Offset

// Read Byte0 to Byte15 for (int i = 0; i < 16; i++) { std::getline(ss, byte, ’,’); int value = hexToInt(byte); // Convert hex string to integer row_data.push_back(value); }

// Add the row to the current frame data frame_data.push_back(row_data);

// Check if we need to save the frame (if time difference is detected) if (elapsed_time != previous_time && !frame_data.empty()) { // Save the current frame as an image saveFrameAsPPM(frame_data, frame_number); frame_number++;

// Sleep for the time difference between frames to simulate video timing double time_diff = elapsed_time - previous_time; std::this_thread::sleep_for(std::chrono::milliseconds(static_cast<i
* 1000)));

// Clear the frame data for the next frame frame_data.clear(); }

previous_time = elapsed_time; }

std::cout « ”Frames saved.” « std::endl; return 0; }

Produces frames but I do not know what frame rate or sized was used to currently cannot be converted to a video using ffmpeg

VERSION #4

#include <iostream> #include <fstream> #include <sstream> #include <vector> #include <iomanip> // for hex

// Function to convert hex string to integer int hexToInt(const std::string& hexStr) { int value; std::stringstream ss; ss « std::hex « hexStr; ss » value; return value; }

// Function to save a frame as a PPM image void saveFrameAsPPM(const std::vector<std::vector<int»& frame_data, int frame_number) { std::string filename = "frame_" + std::to_string(frame_number) + ".ppm";

std::ofstream imageFile(filename);

if (!imageFile) { std::cerr « "Error: Could not create PPM file " « filename « std::endl; return; }

const int rows = 16; // Fixed number of rows const int cols = 16; // Fixed number of columns

// Write PPM header imageFile « "P3\n" « cols « " " « rows « "\n255\n";

// Write pixel data for (int i = 0; i < rows; i++) { for (int j = 0; j < cols; j++) { int pixel_value = frame_data[i % frame_data.size()][j % frame_data[0].size()]; imageFile « pixel_value « " " « pixel_value « " " « pixel_value « " "; } imageFile « "\n"; }

imageFile.close(); }

// Main function to read CSV and generate frames int main() { std::ifstream file("hex_log_20240910_173153.csv"); if (!file.is_open()) { std::cerr « "Error: Could not open the CSV file!" « std::endl; return 1; }

std::string line; std::vector<std::vector<int» frame_data; int frame_number = 0;

// Skip the first line (header row) std::getline(file, line);

// Process CSV data while (std::getline(file, line)) { std::stringstream ss(line); std::string byte; std::vector<int> row_data;

// Read Elapsed Time and Offset (skip them for now) std::getline(ss, byte, ','); // Elapsed Time std::getline(ss, byte, ','); // Offset

// Read Byte0 to Byte15 for (int i = 0; i < 16; i++) { std::getline(ss, byte, ','); int value = hexToInt(byte); // Convert hex string to integer row_data.push_back(value); }

// Add the row to the current frame data frame_data.push_back(row_data);

// Save the current frame as an image every 16 rows if (frame_data.size() == 16) { saveFrameAsPPM(frame_data, frame_number); frame_number++; frame_data.clear(); // Clear for the next frame } }

return 0; }

Tasks for 9/9 Week (Jack Couture)

Convert Python file to C++

Implement a script to take a CSV of data from the depth camera and create an image

Test this using a virtual machine connected to the depth camera or CSV files

Write the Introduction for the Project Proposal

Project Ideas (Jack Couture)

Smart Irrigation System

Description: Develop a cost-effective, automated irrigation system that uses soil moisture sensors, weather forecasts, and IoT technology to optimize watering schedules for gardens or small farms. Hardware: Soil moisture sensors, microcontroller (like Arduino or Raspberry Pi), water solenoids and valves. Software: Application to monitor and control the system, integration with weather APIs to optimize watering based on forecasts.

Automated Recycling Sorter

Description: Build a machine that can sort recyclables (like plastics, metals, and glass) using sensors and basic robotic arms. Hardware: Conveyor belt, sensors (like infrared for plastic or magnets for metals), robotic arms for sorting. Software: Control software for the sorting system, possibly incorporating machine learning for material identification.

Smart Traffic Light Controller

Description: Develop a traffic light control system that adapts in real-time to traffic conditions, potentially integrating with emergency services to prioritize vehicle flow. Hardware: Microcontrollers, cameras or sensors for traffic detection, model traffic lights for demonstration. Software: Algorithm for dynamic traffic management, software to process input from traffic sensors and control lights accordingly.

Automated Public Library System

Description: Create an automated system to manage book check-ins and check-outs, reservations, and sorting at public libraries. This system could use RFID technology for tracking books and a robotic system for sorting and shelving returned items. Hardware: RFID readers and tags, robotic arms for handling books, conveyor systems for moving books within the library. Software: Library management software to track book locations, check-out and return statuses, and user reservations. Integration with existing library databases and user interface development for self-service kiosks.

Intelligent Document Organizer

Description: Build a device that can automatically scan, categorize, and file physical documents into digital formats using OCR and custom sorting algorithms. Hardware: Document scanner, automated filing system with mechanical components for handling paper. Software: OCR software to convert scanned images to text, categorization algorithms based on content analysis, user interface for accessing and searching documents.

ML and AI Ideas

Have an algorithm take benchmarking data and determine algorithms such as a weighted average between the two capture devices based on angle and estimated distance

Put these algorithms on the Pico to avoid flooding it with the AI as a whole

Tell a machine learning program what a person's shape is using the depth camera's features as well as other objects

Attach machine learning program to database to store what is what

Have custom vibrations depending on the object (person, chair, etc)

TESTING PHASE OF CODE

VERSION COMPLETE PROD FINAL TEST

from machine import UART, Pin import utime import struct

led = machine.Pin(25, machine.Pin.OUT) buzzer_one = machine.Pin(10, machine.Pin.OUT) buzzer_two = machine.Pin(14, machine.Pin.OUT) UNIT = 0

Initialize UART on GP0 (TX) and GP1 (RX) uart0 = UART(0, baudrate=115200, tx=Pin(0), rx=Pin(1)) # LiDAR uart1 = UART(1, baudrate=115200, tx=Pin(4), rx=Pin(5)) # Depth Camera

def send_at_command(command): # Send configuration settings to the depth camera uart1.write(command + '\r') utime.sleep_ms(100) # Wait for response response = b" if uart1.any(): response += uart1.read() print(f"Command: {command}, Response: {response}") return response

def initialize_camera(): utime.sleep_ms(100) send_at_command("AT+DISP=5") # Enable UART display utime.sleep_ms(100) send_at_command("AT+UNIT=9") utime.sleep_ms(100) send_at_command("AT+FPS=15")

last_frame_time = None # Global variable to store the last frame time (for tracking latency) last_frame_time_camera = None

def read_lidar(): global last_frame_time current_time = utime.ticks_ms() try: if uart0.any(): data = uart0.read() if len(data) >= 7 and data.startswith(b'YY'): distance = struct.unpack('<H', data[2:4])[0] last_frame_time = current_time return distance else: return None except Exception as e: print(f"Error: {e}") return None return None

def get_distance_from_packet(data_bytes): """ Extract distance from packet using formulas from section 1.8 """ try: # Make sure we have enough bytes if len(data_bytes) < 32: return None

Look for packet header (0x00, 0xFF) for i in range(len(data_bytes) - 1): if data_bytes[i] == 0x00 and data_bytes[i+1] == 0xFF: # Found packet start packet = data_bytes[i:] # Make sure we have enough bytes after header if len(packet) < 22: # 2(header) + 2(length) + 16(other) + 1(check) + 1(end) return None # Image frame starts after header(2) + length(2) + other(16) = 20 bytes frame_start = i + 20 # Now process pixel data, starting from frame_start pixel_values = [] for idx in range(frame_start, len(data_bytes), 3): p = data_bytes[idx] distance = int((p / 5.1) ** 2) pixel_values.append(distance) if pixel_values: avg_distance = sum(pixel_values) / len(pixel_values) print(f"Packet found at {i}, Average Distance: {avg_distance}") return avg_distance else: return None return None except Exception as e: print(f"Error in get_distance_from_packet: {e}") return None

Global variables for non-blocking camera read camera_data_buffer = bytearray() camera_reading_in_progress = False

def read_camera_non_blocking(): """Non-blocking read from the depth camera.""" global camera_data_buffer, camera_reading_in_progress try: if uart1.any(): data = uart1.read(uart1.any()) if data is not None: camera_data_buffer.extend(data) # Check if we have a complete packet if len(camera_data_buffer) >= 32: # Attempt to extract distance from packet distance = get_distance_from_packet(camera_data_buffer) if distance is not None: # Reset buffer and reading flag camera_data_buffer = bytearray() camera_reading_in_progress = False return distance else: # Remove processed bytes to avoid buffer growing indefinitely camera_data_buffer = camera_data_buffer[-64:] # Keep last 64 bytes except Exception as e: print(f"Camera Error: {e}") camera_data_buffer = bytearray() camera_reading_in_progress = False return None

print("Initializing camera...") send_at_command("AT") initialize_camera() print("Camera initialized.")

Initialize variables for the main loop n = 1 # The 'n' in 'n x 100' LiDAR scans start_time = utime.ticks_ms() lidar_scan_count = 0 last_depth_camera_time = start_time lidar_readings = []

Main loop for 10 seconds while True: try: # Read LiDAR data distance = read_lidar() if distance is not None: lidar_scan_count += 1 # Append the LiDAR reading to the list lidar_readings.append(distance) # Check if it's time to initiate depth camera reading current_time = utime.ticks_ms() if not camera_reading_in_progress and (lidar_scan_count >= n * 100 or utime.ticks_diff(current_time, last_depth_camera_time) >= 10000): # Start depth camera reading camera_reading_in_progress = True last_depth_camera_time = current_time lidar_scan_count = 0 # If depth camera reading is in progress, read data if camera_reading_in_progress: distance2 = read_camera_non_blocking() if distance2 is not None: # Depth camera reading is available total_readings = lidar_readings + [distance2] avg_distance = sum(total_readings) / len(total_readings) print(f"Average Distance combining depth camera and LiDAR readings: {avg_distance}") # Decide whether to turn on the buzzer and LED if (distance <= 200 and distance >= 100) or (distance2 <= 2000 and distance2 > 1000): led.value(1) buzzer_one.value(1) buzzer_two.value(0) elif (distance < 100 and distance >= 50) or (distance2 <= 1000 and distance2 > 500): led.value(1) buzzer_one.value(0) buzzer_two.value(1) elif (distance < 50) or (distance2 <= 500): led.value(1) buzzer_one.value(1) buzzer_two.value(1) else: led.value(0) buzzer_one.value(0) buzzer_two.value(0) # Reset LiDAR readings and flags for the next cycle lidar_readings = [] camera_reading_in_progress = False elif not uart1.any(): # Depth camera reading is not available after attempting if lidar_readings: # Use LiDAR readings alone avg_distance = sum(lidar_readings) / len(lidar_readings) print(f"Average Distance from LiDAR readings: {avg_distance}") # Decide whether to turn on the buzzer and LED if (distance <= 200 and distance >= 100) or (distance2 <= 2000 and distance2 > 1000): led.value(1) buzzer_one.value(1) buzzer_two.value(0) elif (dis-

tance < 100 and distance >= 50) or (distance2 <= 1000 and distance2 > 500): led.value(1) buzzer_one.value(0) buzzer_two.value(1) elif (distance < 50) or (distance2 <= 500): led.value(1) buzzer_one.value(1) buzzer_two.value(1) else: led.value(0) buzzer_one.value(0) buzzer_two.value(0) # Reset LiDAR readings and flags for the next cycle lidar_readings = [] camera_reading_in_progress = False else: # No readings available print("No readings available to compute average distance.") led.value(0) buzzer_one.value(0) buzzer_two.value(0) camera_reading_in_progress = False else: # No depth camera reading in progress; continue collecting LiDAR data pass except Exception as e: print(f"Error: {e}")

ORIGINAL ATTEMPT

from machine import UART, Pin, PWM import utime import struct

Initialize PWM outputs on GP21, GP10, GP17, GP14 pwm_pins = [21, 10, 17, 14] pwms = [] pwm_freq = 1000 # 1 kHz frequency

for pin_num in pwm_pins: pwm = PWM(Pin(pin_num)) pwm.freq(pwm_freq) pwm.duty_u16(0) # Start with 0% duty cycle (buzzer off) pwms.append(pwm)

Initialize UART on GP0 (TX) and GP1 (RX) uart0 = UART(0, baudrate=115200, tx=Pin(0), rx=Pin(1)) # LiDAR uart1 = UART(1, baudrate=115200, tx=Pin(4), rx=Pin(5)) # Depth Camera

def send_at_command(command): # Send configuration settings to the depth camera uart1.write(command + '\r') utime.sleep_ms(100) # Wait for response response = b'' if uart1.any(): response += uart1.read() print(f"Command: {command}, Response: {response}") return response

def initialize_camera(): utime.sleep_ms(1000) send_at_command("AT+DISP=5") # Enable UART display utime.sleep_ms(1000) send_at_command("AT+UNIT=9") utime.sleep_ms(1000) send_at_command("AT+FPS=15")

def read_lidar(): try: if uart0.any(): data = uart0.read() if len(data) >= 7 and data.startswith(b'YY'): distance = struct.unpack('<H', data[2:4])[0] return distance else: return None except Exception as e: print(f"Error: {e}") return None return None

def get_distance_from_packet(data_bytes): try: # Ensure we have enough bytes if len(data_bytes) < 32: return None # Look for packet header (0x00, 0xFF) for i in range(len(data_bytes) - 1): if data_bytes[i] == 0x00 and data_bytes[i+1] == 0xFF: # Found packet start packet = data_bytes[i:] # Ensure we have enough bytes after header if len(packet) < 22: return None # Image frame starts after header(2) + length(2) + other(16) = 20 bytes frame_start = i + 20 # Process pixel data, starting from frame_start pixel_values = [] for idx in range(frame_start, len(data_bytes), 10): p = data_bytes[idx] distance = int((p / 5.1) ** 2) pixel_values.append(distance) if pixel_values: avg_distance = sum(pixel_values) / len(pixel_values) print(f"Packet found at {i}, Average Distance: {avg_distance}") return avg_distance else: return None return

None except Exception as e: print(f"Error in get_distance_from_packet: {e}") return None

Global variables for non-blocking camera read camera_data_buffer = bytearray() camera_reading_in_progress = False

def read_camera_non_blocking(): """Non-blocking read from the depth camera.""" global camera_data_buffer, camera_reading_in_progress try: if uart1.any(): data = uart1.read(uart1.any()) if data is not None: camera_data_buffer.extend(data) # Check if we have a complete packet if len(camera_data_buffer) >= 32: # Attempt to extract distance from packet distance = get_distance_from_packet(camera_data_buffer) if distance is not None: # Reset buffer and reading flag camera_data_buffer = bytearray() camera_reading_in_progress = False return distance else: # Remove processed bytes to avoid buffer growing indefinitely camera_data_buffer = camera_data_buffer[-64:] # Keep last 64 bytes else: # No data available, continue pass else: # No data available, continue pass except Exception as e: print(f"Camera Error: {e}") camera_data_buffer = bytearray() camera_reading_in_progress = False return None

def calculate_averages(): global lidar_average_calculate, lidar_average_count, depth_average_calculate, depth_average_count global lidar_average, depth_average if lidar_average_count > 0: lidar_average = lidar_average_calculate / lidar_average_count else: lidar_average = -1 if depth_average_count > 0: depth_average = depth_average_calculate / depth_average_count else: depth_average = -1 # Reset counts and sums lidar_average_count = 0 lidar_average_calculate = 0 depth_average_count = 0 depth_average_calculate = 0

class BuzzerController: def __init__(self, pwm_list): self.pwms = pwm_list # list of PWM objects self.state = 'off' # current state: 'on' or 'off' self.on_duration = 0 # duration in ms self.off_duration = 0 self.last_change_time = utime.ticks_ms()

def set_pattern(self, on_duration, off_duration): self.on_duration = on_duration self.off_duration = off_duration self.last_change_time = utime.ticks_ms() self.state = 'on' if on_duration > 0 else 'off' self.update_pwm()

def update(self): current_time = utime.ticks_ms() elapsed_time = utime.ticks_diff(current_time, self.last_change_time) if self.state == 'on' and self.on_duration > 0: if elapsed_time >= self.on_duration: self.state = 'off' self.last_change_time = current_time self.update_pwm() elif self.state == 'off' and self.off_duration > 0: if elapsed_time >= self.off_duration: self.state = 'on' self.last_change_time = current_time self.update_pwm()

def update_pwm(self): if self.state == 'on': for pwm in self.pwms: pwm.duty_u16(32768) # 50% duty cycle (adjust as needed) else: for pwm in self.pwms: pwm.duty_u16(0) # duty cycle 0% (off)

print(”Initializing camera...”) send_at_command(”AT”) initialize_camera() print(”Camera initialized.”)

Initialize variables for the main loop calibration_start_time = utime.ticks_ms() calibration_duration = 10000 # Calibration period in milliseconds (10 seconds) calibration = True # Flag to indicate calibration period

lidar_average_calculate = 0 lidar_average_count = 0 lidar_average = -1

depth_average_calculate = 0 depth_average_count = 0 depth_average = -1

Initialize the BuzzerController buzzer_controller = BuzzerController(pwms)

Main loop while True: try: current_time = utime.ticks_ms() elapsed_calibration_time = utime.ticks_diff(current_time, calibration_start_time)

if calibration: if elapsed_calibration_time <= calibration_duration: # Collect data for calibration distance = read_lidar() if distance is not None: lidar_average_calculate += distance lidar_average_count += 1 distance2 = read_camera_non_blocking() if distance2 is not None: depth_average_calculate += distance2 depth_average_count += 1 else: # Calibration over, calculate averages calculate_averages() print(f”Calibration completed. LiDAR average: {lidar_average} mm, Depth camera average: {depth_average} mm”) calibration = False else: # Process new readings distance = read_lidar() if distance is not None: # Categorize the LiDAR distance and set buzzer pattern accordingly if distance <= 500: category = ”0.5m or less” buzzer_controller.set_pattern(on_duration=0, off_duration=0) # Buzzer on continuously elif distance <= 1000: category = ”0.5m to 1m” buzzer_controller.set_pattern(on_duration=500, off_duration=500) # On 0.5s, Off 0.5s elif distance <= 2000: category = ”1m to 2m” buzzer_controller.set_pattern(on_duration=200, off_duration=800) # On 0.2s, Off 0.8s else: category = ”beyond 2m” buzzer_controller.set_pattern(on_duration=0, off_duration=0) # Buzzer off continuously print(f”LiDAR distance: {distance} mm, Category: {category}”) else: pass

Update the buzzer controller buzzer_controller.update()

Process depth camera readings if needed # distance2 = read_camera_non_blocking() # Add similar logic for the depth camera if need be idk :)

except Exception as e: print(f”Error: {e}”)

Attempts to average and modify buzzing pulse using computed averages

PRE-PROD VERSION

from machine import UART, Pin import utime import struct

led = machine.Pin(21, machine.Pin.OUT) buzzer_one = machine.Pin(10, machine.Pin.OUT) buzzer_two = machine.Pin(14, machine.Pin.OUT) UNIT = 0

Initialize UART on GP0 (TX) and GP1 (RX) uart0 = UART(0, baudrate=115200, tx=Pin(0), rx=Pin(1)) # LiDAR uart1 = UART(1, baudrate=115200, tx=Pin(4), rx=Pin(5)) # Depth Camera

def send_at_command(command): # Send configuration settings to the depth camera uart1.write(command + '\r') utime.sleep_ms(100) # Wait for response response = b" if uart1.any(): response += uart1.read() print(f"Command: {command}, Response: {response}") return response

def initialize_camera(): utime.sleep_ms(1000) send_at_command("AT+DISP=4") # Enable UART display utime.sleep_ms(1000) send_at_command("AT+UNIT=9") utime.sleep_ms(1000) send_at_command("AT+FPS=15")

last_frame_time = None # Global variable to store the last frame time (for tracking latency) last_frame_time_camera = None

def read_lidar(): global last_frame_time current_time = utime.ticks_ms() try: if uart0.any(): data = uart0.read() if len(data) >= 7 and data.startswith(b'YY'): distance = struct.unpack('<H', data[2:4])[0] last_frame_time = current_time return distance else: return None except Exception as e: print(f"Error: {e}") return None return None

def get_distance_from_packet(data_bytes): """ Extract distance from packet using formulas from section 1.8 """ try: # Make sure we have enough bytes if len(data_bytes) < 32: return None

Look for packet header (0x00, 0xFF) for i in range(len(data_bytes) - 1): if data_bytes[i] == 0x00 and data_bytes[i+1] == 0xFF: # Found packet start packet = data_bytes[i:] # Make sure we have enough bytes after header if len(packet) < 22: # 2(header) + 2(length) + 16(other) + 1(check) + 1(end) return None # Image frame starts after header(2) + length(2) + other(16) = 20 bytes frame_start = i + 20 # Now process pixel data, starting from frame_start pixel_values = [] for idx in range(frame_start, len(data_bytes), 3): p = data_bytes[idx] distance = int((p / 5.1) ** 2) pixel_values.append(distance) if pixel_values: avg_distance = sum(pixel_values) / len(pixel_values) print(f"Packet found at {i}, Average Distance: {avg_distance}") return avg_distance else: return None return None except Exception as e: print(f"Error in get_distance_from_packet: {e}") return None

Global variables for non-blocking camera read camera_data_buffer = bytearray() camera_reading_in_progress = False

def read_camera_non_blocking(): """Non-blocking read from the depth camera.""" global camera_data_buffer, camera_reading_in_progress try: if uart1.any(): data = uart1.read(uart1.any()) if data is not None: camera_data_buffer.extend(data) # Check if we have a complete packet if len(camera_data_buffer) >= 32: # Attempt to extract distance from packet distance = get_distance_from_packet(camera_data_buffer) if distance is

not None: # Reset buffer and reading flag camera_data_buffer = bytearray() camera_reading_in_progress = False return distance else: # Remove processed bytes to avoid buffer growing indefinitely camera_data_buffer = camera_data_buffer[-64:] # Keep last 64 bytes except Exception as e: print(f"Camera Error: {e}") camera_data_buffer = bytearray() camera_reading_in_progress = False return None

print("Initializing camera...") send_at_command("AT") initialize_camera() print("Camera initialized.")

while True: try: # Read LiDAR data distance = read_lidar() if distance is not None: # Decide whether to turn on the buzzer and LED if (distance <= 200 and distance >= 100): led.value(1) buzzer_one.value(1) buzzer_two.value(0) elif (distance < 100 and distance >= 50): led.value(1) buzzer_one.value(0) buzzer_two.value(1) elif (distance < 50): led.value(1) buzzer_one.value(1) buzzer_two.value(1) else: led.value(0) buzzer_one.value(0) buzzer_two.value(0) except Exception as e: print(f"Error: {e}")

Focuses on LiDAR to ensure it functions as the depth camera will