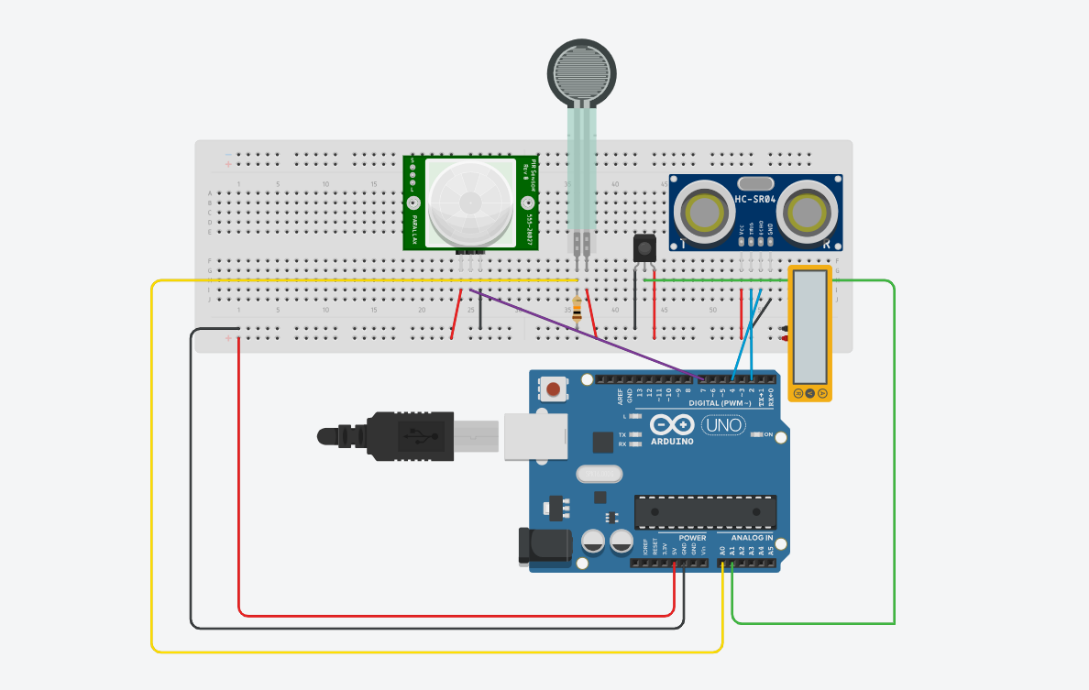
LINK OF THE PROJECT IN TINKERCAD: <https://www.tinkercad.com/things/eZrI7aOquat-lift-optimisation-and-time-estimation?sharecode=QCOUa4dtwrpbX6FUJhbhBxr7DjjDqprS8yHxc4c4ubE>

ARCHITECTURE:



FEATURES:

This code controls an elevator system that involves multiple sensors and decision-making logic. Below are the key features:

1. **Sensor Inputs:**
   * **Force Sensor (FSR)**: Reads weight data from a force sensor connected to analog pin A0. The sensor measures the weight of the passengers.
   * **IR Distance Sensor**: Connected to analog pin A1, this sensor measures the space availability inside the elevator based on proximity.
   * **Energy Harvesting Sensor**: Connected to analog pin A2, it measures the energy level (normalized to a value between 0 and 1).
   * **PIR Sensor**: Connected to digital pin 2, this sensor detects human presence at specific stop floors in the elevator.
2. **Data Collection & Variables:**
   * **Data Arrays**: Arrays (fsrValues, irValues, travelTimes, etc.) store various sensor data for each collected data point (up to 10 points).
   * **Max Floors**: The system supports up to 10 floors (maxFloors).
   * **Data Points**: The system collects a set number of data points (dataPoints = 10).
3. **Time and Stop Management:**
   * **Travel Time Calculation**: Travel time is calculated based on the number of floors and sensor values (e.g., IR distance and FSR weight). The time adjusts with acceleration, deceleration, and floor travel time.
   * **Stop Management**: The system simulates multiple stop floors between the starting and destination floors, ensuring no stop is repeated or at the same floor as the starting or destination floor.
   * **Stop Delays**: A delay (stopDelay) is added for each stop made to account for the elevator pausing.
4. **Decision Logic:**
   * **Weight Check**: If the FSR sensor detects a weight above a certain threshold, the elevator will not allow further movement, and a "Lift Overweight" decision is made.
   * **Space Availability**: If there is no space inside the elevator (based on the IR sensor), the system will display "Space Unavailable."
   * **PIR Sensor for Stops**: The system checks if the PIR sensor detects presence at a stop floor. If a person is detected, the elevator will stop.
   * **Energy Harvesting**: The system tracks energy usage and accumulates energy over time.
5. **Data Output:**
   * **CSV Output**: Data is printed to the serial monitor in CSV format with the following columns: Timestamp, Start Floor, End Floor, Stops, Stop Floors, FSR Value, IR Value, Travel Time, Energy Level, Accumulated Energy, and Decision.
   * **Export Data**: Once the number of data points is reached, the printData() function exports all collected data as a CSV format.
6. **Randomization and Simulations:**
   * **Simulated IR Sensor Values**: The IR sensor value is simulated with random values within a range.
   * **Random Floor Generation**: The destination and stop floors are chosen randomly, ensuring that the destination floor is different from the current floor.
7. **Energy Accumulation:**
   * **Energy Level Calculation**: Each elevator movement records the energy level (from the energy harvesting sensor), which is normalized and accumulated over time.
8. **Data Reset and Continuity:**
   * After collecting a set number of data points (dataPoints = 10), the system resets the data collection process to start anew.

CODE:

#define FSR\_PIN A0 // Force Sensor (FSR) connected to A0

#define IR\_PIN A1 // IR Distance Sensor connected to A1

#define ENERGY\_PIN A2 // Energy harvesting sensor connected to A2

#define PIR\_PIN 2 // PIR sensor connected to digital pin 2

const int dataPoints = 10; // Number of data points to collect

const int maxFloors = 10; // Maximum number of floors

const float baseTravelTimePerFloor = 2.0; // Base travel time per floor in seconds

const float stopDelay = 3.0; // Delay per stop in seconds

const int weightThreshold = 700; // Threshold for FSR sensor to determine overweight

const int spaceThreshold = 500; // Threshold for IR sensor to determine space

int fsrValues[dataPoints];

int irValues[dataPoints];

float travelTimes[dataPoints];

unsigned long timestamps[dataPoints];

int startFloors[dataPoints];

int endFloors[dataPoints];

int stopCounts[dataPoints];

int stopFloors[dataPoints][5]; // Array to store stop floors for each journey

float energyLevels[dataPoints]; // Array to store energy levels

float accumulatedEnergy = 0; // Variable to store accumulated energy

int index = 0;

int currentFloor = 0; // Starting floor

void setup() {

Serial.begin(9600); // Start Serial Monitor

pinMode(PIR\_PIN, INPUT); // Set PIR sensor pin as input

// CSV header aligned with SEIS schema

Serial.println("Timestamp,StartFloor,EndFloor,Stops,StopFloors,FSRValue,IRValue,TravelTime,EnergyLevel,AccumulatedEnergy,Decision");

}

void loop() {

int fsrValue = analogRead(FSR\_PIN); // Read FSR value

int irValue = analogRead(IR\_PIN); // Read IR sensor value

float energyLevel = analogRead(ENERGY\_PIN) / 1023.0; // Read and normalize energy level

// Simulate IR sensor values for Tinkercad

irValue = random(100, 900); // Simulated IR values

int destinationFloor;

do {

destinationFloor = random(1, maxFloors + 1); // Random destination floor between 1 and maxFloors

} while (destinationFloor == currentFloor); // Ensure destination floor is different from the current floor

// Simulate the number of stops

int stops = random(1, 4); // Random number of stops between 1 and 3

for (int i = 0; i < stops; i++) {

do {

stopFloors[index][i] = random(1, maxFloors + 1); // Generate random stop floors

} while (stopFloors[index][i] == currentFloor || stopFloors[index][i] == destinationFloor || (i > 0 && stopFloors[index][i] == stopFloors[index][i - 1]));

}

float loadFactor = fsrValue / 1023.0; // Normalize FSR value

// Calculate acceleration and deceleration times based on IR sensor value

float accelerationTime = min(irValue / 1000.0, 2.0); // Example calculation, scale based on IR value

float decelerationTime = min(irValue / 1000.0, 2.0); // Example calculation, scale based on IR value

// Calculate travel time with realistic factors

int floorsTravelled = abs(destinationFloor - currentFloor);

float travelTime = floorsTravelled \* baseTravelTimePerFloor \* (1 + loadFactor) + accelerationTime + decelerationTime;

// Store the data

fsrValues[index] = fsrValue;

irValues[index] = irValue;

travelTimes[index] = travelTime;

timestamps[index] = millis(); // Store the timestamp

startFloors[index] = currentFloor;

endFloors[index] = destinationFloor;

stopCounts[index] = 0; // Initialize stop count

energyLevels[index] = energyLevel;

// Accumulate the energy consumption

accumulatedEnergy += energyLevel;

// Decision logic based on IR, FSR, and PIR values

String decision;

if (fsrValue >= weightThreshold) {

decision = "Lift Overweight";

} else {

decision = "Space Unavailable";

for (int i = 0; i < stops; i++) {

// Check if PIR sensor detects a person at stop floors

currentFloor = stopFloors[index][i];

int pirValue = digitalRead(PIR\_PIN); // Read PIR sensor value at each stop

if (pirValue == HIGH) {

stopCounts[index]++;

travelTime += stopDelay;

decision = "Stop at Floor";

}

}

}

// Print data in CSV format for copying and saving

Serial.print(timestamps[index]);

Serial.print(",");

Serial.print(startFloors[index]);

Serial.print(",");

Serial.print(endFloors[index]);

Serial.print(",");

Serial.print(stopCounts[index]);

Serial.print(",");

// Print stop floors

for (int i = 0; i < stopCounts[index]; i++) {

Serial.print(stopFloors[index][i]);

if (i < stopCounts[index] - 1) Serial.print("-");

}

Serial.print(",");

Serial.print(fsrValue);

Serial.print(",");

Serial.print(irValue);

Serial.print(",");

Serial.print(travelTime);

Serial.print(",");

Serial.print(energyLevel);

Serial.print(",");

Serial.print(accumulatedEnergy); // Print the accumulated energy value

Serial.print(",");

Serial.println(decision);

// If the lift is overweight, prevent it from stopping

if (decision == "Lift Overweight") {

// Add code here to prevent the lift from stopping

Serial.println("Lift will not stop due to overload.");

}

index++;

// If we have collected enough data points, print all data

if (index >= dataPoints) {

printData();

index = 0; // Reset the index to collect new data

}

delay(1000); // Delay for readability

}

void printData() {

Serial.println("Exporting data as CSV:");

// CSV header aligned with SEIS schema

Serial.println("Timestamp,StartFloor,EndFloor,Stops,StopFloors,FSRValue,IRValue,TravelTime,EnergyLevel,AccumulatedEnergy,Decision");

for (int i = 0; i < dataPoints; i++) {

Serial.print(timestamps[i]);

Serial.print(",");

Serial.print(startFloors[i]);

Serial.print(",");

Serial.print(endFloors[i]);

Serial.print(",");

Serial.print(stopCounts[i]);

Serial.print(",");

// Print stop floors

for (int j = 0; j < stopCounts[i]; j++) {

Serial.print(stopFloors[i][j]);

if (j < stopCounts[i] - 1) Serial.print("-");

}

Serial.print(",");

Serial.print(fsrValues[i]);

Serial.print(",");

Serial.print(irValues[i]);

Serial.print(",");

Serial.print(travelTimes[i]);

Serial.print(",");

Serial.print(energyLevels[i]);

Serial.print(",");

Serial.print(accumulatedEnergy); // Print the accumulated energy value

Serial.print(",");

String decision;

if (fsrValues[i] >= weightThreshold) {

decision = "Lift Overweight";

} else {

int pirValue = digitalRead(PIR\_PIN); // Read PIR sensor value

if (pirValue == HIGH) {

decision = "Stop at Floor";

} else if (irValues[i] > spaceThreshold && fsrValues[i] < weightThreshold) {

decision = "Wait for Lift";

} else {

decision = "Space Unavailable";

}

}

Serial.println(decision);

}

}

OUTPUT:

Timestamp,StartFloor,EndFloor,Stops,StopFloors,FSRValue,IRValue,TravelTime,EnergyLevel,AccumulaedEnergy,Decision

10831,7,4,2,10-5,0,830,7.66,0.53,10.38,Stop at Floor

11838,5,4,1,9,0,301,2.60,0.53,10.38,Stop at Floor

12844,9,4,2,2-6,0,328,10.66,0.83,10.38,Stop at Floor

13850,6,5,3,7-10-2,0,113,2.23,0.78,10.38,Stop at Floor

14857,2,6,1,5,0,217,8.43,0.41,10.38,Stop at