Smart Office in a Shoebox

Group 10: David Bruns, Jo Hollenbeck, and Max Warkentin

Service Computing Department, IAAS, University of Stuttgart
st167120@stud.uni-stuttgart.de,
st167104@stud.uni-stuttgart.de,
st187876@stud.uni-stuttgart.de

Abstract. The "Smart Office in a Shoebox" project aims to demonstrate how Internet of Things (IoT) devices and Artificial Intelligence (AI) can enhance the energy efficiency and comfort of office environments. This system integrates various sensors, including those for brightness, temperature, humidity, and water, to control actuators like lights, fans, and a cooling pad. The architecture includes an Arduino for low-level processing and a Raspberry Pi for higher-level functions, with AI planning facilitated by a connected laptop. User interfaces allow for real-time monitoring and manual interventions, ensuring flexibility and responsiveness. Prioritizing user comfort, energy savings, and system reliability, the project highlights how smart technologies can create healthier, more sustainable office spaces. Through the implementation and testing of this scaled-down model, the smart office in a shoebox demonstrates the potential for broader application of IoT and AI in actual office settings, aiming to reduce the substantial energy consumption and enhance the wellbeing of office workers.

Keywords: Smart office · IoT · Sensors · Actuators · AI planning · Energy efficiency · Office comfort.

1 System Introduction

A very central structure in our society is the office building; millions of office workers spend hours upon hours in here, using many electronic devices and requiring heat and light. This heavy usage of these buildings poses some major challenges.
According to the International Energy Agency (IEA), the usage of buildings in general made up 30% of global final energy consumption [1].
Among buildings, offices are one of the biggest energy consumers: the U.S. Energy Information Administration published statistics, that make out office buildings as the main culprit. Figure 2 shows the electricity and natural gas consumption of commercial buildings in the U.S. in 2018. It can be observed that for office buildings the electricity consumption is nearly double that of the next biggest consumer, education buildings [5].
With hybrid home-office models becoming more and more popular post COVID, office buildings need to be flexible. This means for example, that heating must

be turned down or lighting and office equipment must be switched off when no one is on site.
Not only do office buildings have a massive economic footprint, they are also not very beneficial to the health of its inhabitants. For example, almost half of employees working in office buildings suffer from back pain, and one-fifth of eye pain [11].

However, office buildings are a necessity for humanity and indispensable for out society. This leads us to the main motivation of this work: How can we use Internet of Things (IoT) devices and Artificial Intelligence (AI) planning to improve the energy efficiency of office buildings, make them more pleasant for the employees to work in and be less detrimental for their inhabitants health while not decreasing productivity levels.
To this end, we model a smart office room inside a shoebox, using brightness, temperature, humidity, water sensors to control actuators like a servo, lights, fan and cooling pad.

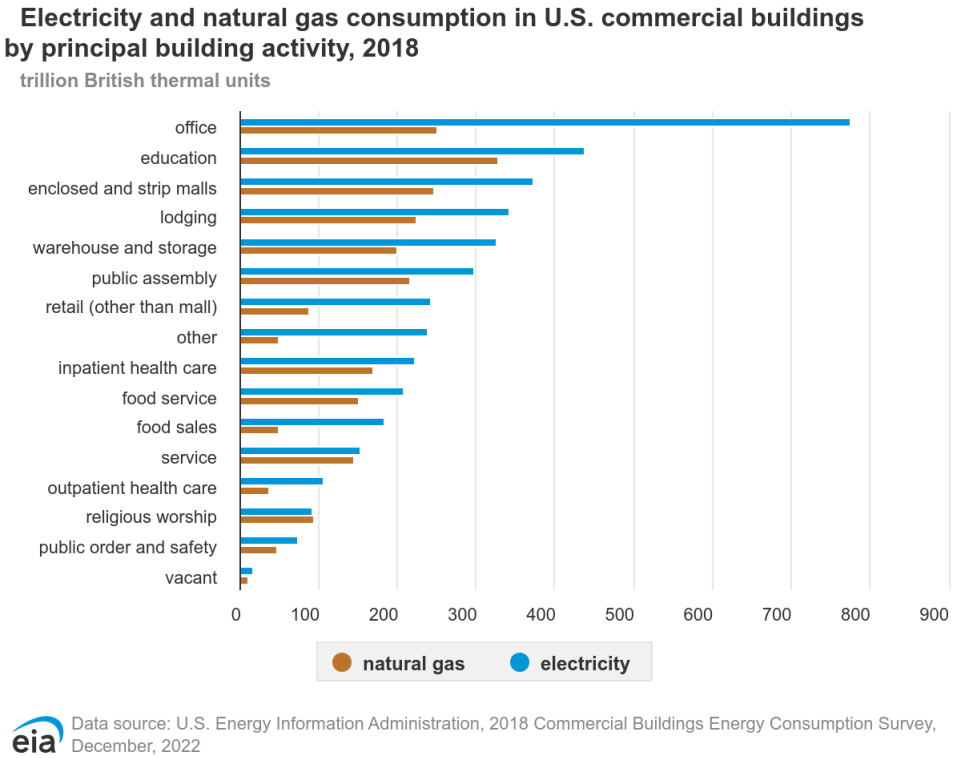

Fig. 1. U.S. Energy Information Administration, 2022

2 System Analysis

The "Smart Office in a Shoebox" project aims to create a comfortable, energy-efficient and healthy environment using various sensors and actuators. To get a better idea what exactly "comfortable, energy-efficient and healthy" means, the system requirements are detailed through use cases and user stories in the following. Our focus is on maintaining optimal environmental conditions, saving energy, and ensuring system reliability. Prioritization is based on the impact on user comfort, safety, and sustainability goals.

2.1 Identification of System Requirements

Use Case: Monitor Light Levels

- **Name**: Monitor light levels
- **Actors**: light sensor, light switch
- **Preconditions**: The light sensor and switch are active and connected to the system.
- **Basic Flow**:
  1. Light sensor measures the current light level in the room.
  2. Sensor sends light level data to the central system.
  3. System processes data and determines if the light level is optimal.
  4. If the light level is too low or too high, the system sends a signal to turn on the lights.
- **Postconditions**: Light levels are maintained at optimal levels for comfort and productivity. The room is never too dark, resulting in a more healthy environment for the eyes and a more comfortable workspace.
- **Exceptions**: Sensor failure, actuator failure, communication errors, planning errors (false decisions).

Use Case: Monitor Temperature and Humidity

- **Name**: Monitor temperature and humidity
- **Actors**: temperature and humidity sensors, window, fan and cooler actuators
- **Preconditions**: Sensors and actuators are active and connected to the system.
- **Basic Flow**:
  1. Temperature and humidity sensors measure the current environmental conditions.
  2. Sensors send data to the central system.
  3. System processes data and determines if conditions are within the comfort range.
  4. If conditions are outside the comfort range, the system sends a signal to adjust the cooler, fan or windows.
- **Postconditions**: Temperature and humidity are maintained within optimal ranges for comfort and health.
- **Exceptions**: Sensor failure, actuator failure, communication errors, planning errors (false decisions).

Use Case: Monitor Precipitation

- **Name**: Monitor Precipitation
- **Actors**: water sensor, weather API, window actuators
- **Preconditions**: The water sensor is active and connected to the system. The weather API is callable. The window actuators are active and connected to the system.
- **Basic Flow**:
  1. Water sensor measures the current precipitation outside the office.
  2. Sensor sends water level data to the central system.
  3. System processes data and double-checks with the weather API whether it is raining or not.
  4. If water level is too high and the weather API agrees that it is indeed raining, the windows are closed.
- **Postconditions**: Precipitation is monitored to automatically close the windows and ensure comfort.
- **Exceptions**: Sensor failure, actuator failure, communication errors, planning errors (false decisions), false weather API data.

User Story: Adjust Lights for Comfort

- **As an employee**, I want the lights to automatically adjust so that I always have optimal lighting conditions without manual intervention.
- **Priority**: High

User Story: Maintain Comfortable Temperature and Humidity

- **As an employee**, I want the temperature and humidity levels to be automatically controlled to stay within a comfortable range so that I can work efficiently.
- **Priority**: Very High

User Story: Close Windows during Rain

- **As an employee**, I want the smart office to close the windows based on the weather forecast and the precipitation measured outside of the office so that I can stay comfortable regardless of external weather conditions.
- **Priority**: Medium

User Story: Manual Interventions

- **As an employee**, I do not want to be disturbed by the system whenever I want to close or open the windows or turn the lights or the cooler on or off. I want the system to adapt to my manual interventions.
- **Priority**: Very High

User Story: Check Sensor Data

- **As an employer**, I want to be able to check the current sensor data to verify whether my employees can work under the best condition.
- **Priority**: Medium

User Story: Save Energy

- **As an environmental sustainability advocate**, I want the system to optimize energy usage so that we can reduce our carbon footprint. I do not want the system to e.g. open the window when its hot outside or the cooler to be on when the windows are open.
- **Priority**: High

User Story: Save Costs

- **As the owner of the office building**, I want the system to save energy whenever possible so that we can reduce the energy costs.
- **Priority**: High

User Story: Ensure System Reliability

- **As an IT administrator**, I want the system to have minimal downtime and quick recovery so that it can consistently provide a comfortable and safe environment.
- **Priority**: Medium

2.2 Ranking/Prioritization of System Requirements

In Table 1, the system requirements for our smart office in a shoebox are listed and prioritized, beginning with the most important requirement. The foremost priority, identified as manual interventions, underscores the necessity for employees to intervene directly in creating a healthy and comfortable environment during system failures. This prioritization is grounded in the significant impact on employee well-being and stress reduction, highlighting the system's responsiveness to human needs as paramount.
Maintaining comfortable temperature and humidity emerges as another high-priority requirement due to its direct influence on employee comfort and productivity. The regulation of these environmental factors not only supports physical well-being but also fosters an optimal working environment essential for sustained performance [10]. Similarly, adjusting lights for comfort is prioritized highly owing to its role in enhancing the work environment and mitigating eye strain, thus promoting employee satisfaction and operational efficiency [8].
Saving energy and costs is positioned as a high-priority requirement aligned with broader sustainability objectives. This criterion aims to optimize resource

Priority	System Requirement
Very High	**Manual Interventions** – *Lets employees create a healthy and comfortable environment whenever the system fails.* – *Directly impacts health and well-being, reduces stress potentially introduced by the system.*
Very High	**Maintain Comfortable Temperature and Humidity** – *Ensures employee comfort and productivity.* – *Directly impacts health and well-being.*
High	**Adjust Lights for Comfort** – *Enhances work environment and reduces eye strain.* – *Essential for daily operations and comfort.*
High	**Save Energy and Costs** – *Aligns with sustainability goals.* – *Reduces operational costs and environmental impact.*
High	**Ensure System Reliability** – *Critical for maintaining continuous operation and user trust.* – *Prevents disruptions in a smart office environment.*
Medium	**Anticipate Precipitation** – *Adds proactive adjustments for enhanced comfort.* – *Improves system responsiveness to external factors.*
Medium	**Check Sensor Data** – *Gives a way to monitor the environment and comprehend the system's decisions.* – *Improves understanding of the system.*

Table 1. Prioritization of System Requirements

utilization, thereby reducing operational expenses and environmental impact, reflecting responsible corporate stewardship. Ensuring system reliability also ranks high, emphasizing its criticality in maintaining continuous operations and user trust by minimizing disruptions within the smart office ecosystem.
The ranking of anticipating precipitation as a medium-priority requirement introduces proactive adjustments that enhance comfort by responding preemptively to external weather conditions. This anticipatory capability contributes to the system's adaptability and resilience, enhancing overall user experience. Its priority is not ranked as high because rain is a more rare environmental event, whereas temperature, humidity and lighting are constant factors that create a need for adjustment, Checking sensor data, also designated as medium-priority, provides essential insights into environmental monitoring and system

decision-making processes, thereby improving the comprehensibility and operational efficiency of the smart office infrastructure. This requirement is not ranked as high because it is a feature to the system and not absolutely essential to the functionality of it.

3 System Architecture Design

The architecture of the smart office system is designed to ensure an integrated and intelligent environment where sensor data is efficiently collected, processed, and utilized to control various actuators based on user preferences and environmental conditions. This system is segmented into multiple layers, each serving distinct but interconnected functionalities, which are crucial for the system's effectiveness and adaptability.

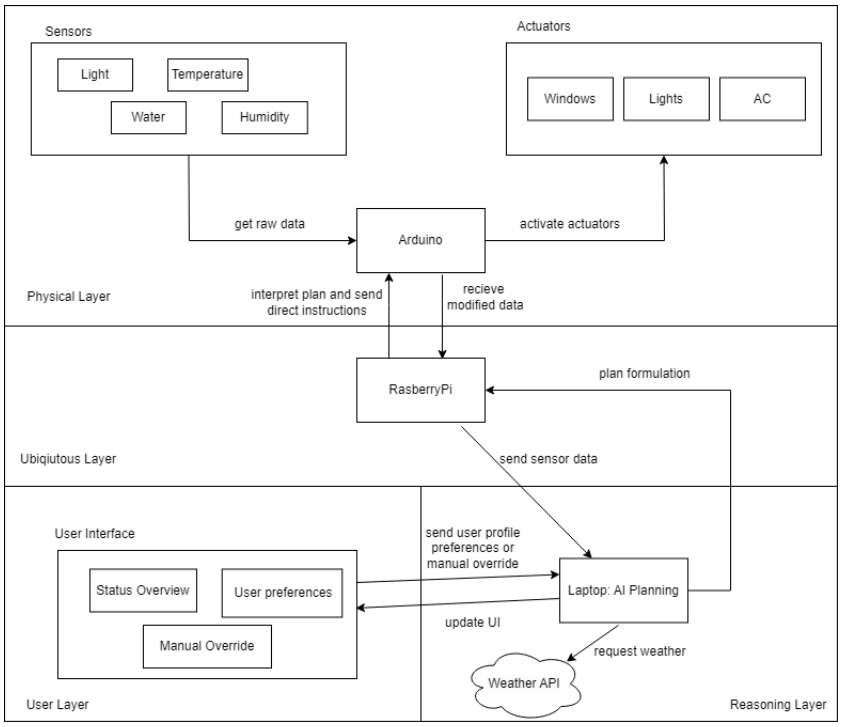

Fig. 2. Architecture design

At the core of the physical layer, we have a variety of sensors that include light, temperature, humidity and water sensors. These sensors collect raw environmental data that is essential for monitoring and managing the internal conditions of the office. The collected data is then fed into a Data Processing Unit, in our

case an Arduino microcontroller. It interprets environmental data and sends the modified data to the Context Management Unit, a Raspberry Pi. Besides the sensors, the Data Processing Unit keeps track of the operational state of the actuators such as windows, lights, and the air conditioning system. The actuators respond to commands from the Context Management Unit via the Arduino to modify the environment based on generated plans and user preferences. Specifically, windows can be opened or closed to regulate humidity and temperature, lights can be turned on or off to control lighting conditions and the cooler and fan can be turned on to ensure the desired temperature.
We chose to incorporate an Arduino in addition to the Raspberry Pi for several key reasons:

1. **Layered Development**
   Splitting the problem into low-level and mid-level layers simplifies development. The low-level layer handles basic electrical signals, such as detecting a HIGH signal on pin 3. The mid-level layer processes this information into actionable commands, like turning on the air conditioner by setting the cooler-pin and fan-pin to HIGH. While this problem-splitting could be achieved purely in software, utilizing dedicated hardware for each layer results in a more structured and organized approach.
2. **GPIO Limitation Handling**
   The GPIO pins on the Raspberry Pi are limited in number. By offloading some of the sensor and actuator management to the Arduino, we prevent running out of GPIO pins on the Raspberry Pi. This setup makes it easier to scale the system by adding more sensors and actuators without extensive reworking of the implementation.
3. **Enhanced Cybersecurity**
   The Arduino, when configured without a bootloader, has no internet connectivity and is connected to the system via a single USB connection. This isolation significantly enhances security by reducing the attack surface. The Raspberry Pi, being internet-connected, is more vulnerable to cyber attacks. By isolating critical actuators on the Arduino, such as servos restricted to specific movements to avoid damage, we ensure that these components remain secure and operate within safe parameters.

By implementing this dual-layer architecture, we achieve a more manageable, scalable, and secure smart office system.

The Ubiquitous Layer, consisting of the Raspberry Pi, plays a very important role in the system. This Context Management Unit carries out three major functions:
Context Management: The Raspberry Pi acquires and interprets sensor data to get a clear overview of what is currently happening in the environment. This is a complete understanding of the situation and guarantees that any information required for the AI planning is available.
Communication Brokerage: The Raspberry Pi is an intermediary for data that flows between different layers and their parts. This feature allows smooth and

effective communication between the physical and reasoning layers and back, ensuring real-time data exchange and system responsiveness.
Data Repository: The Raspberry Pi also serves as a storage centre where device information, system configurations and the actuator states are stored.
In the reasoning layer, the Raspberry Pi collaborates with a laptop dedicated to AI planning. This laptop is equipped with sophisticated algorithms such as the Metric-FF [2] planner capable of making intelligent decisions. It receives processed sensor data and actuator states from the Raspberry Pi and user preferences from the interface. Using this information, the AI on the laptop formulates and solves given plans. The resulting solution is then sent back to the Raspberry Pi.
The user layer primarily consists of an interface that provides users with a status overview of the system, including current environmental conditions and the operational status of actuators. Users can input their preferences, which are directly fed into the AI planning on the laptop for consideration in future environmental adjustments. Additionally, users can manually override automatic controls through the interface, offering immediate adjustments to the system's operations, such as turning the light on or opening windows. This layer is essential for ensuring that the system remains flexible and responsive to individual user needs.
The architecture provides a robust framework for a smart office environment, where each layer works in concert to create a responsive and intelligent system. The flow of data from sensors to actuators, mediated by advanced computing platforms and user inputs, ensures that the office environment can automatically adjust to both the preferences of its occupants and external environmental conditions. This dynamic interaction achieves a balance of comfort, efficiency, and user satisfaction. Our project presents this sophisticated system in a miniature form within a shoebox, demonstrating its design and functionality on a smaller scale.

4 System Implementation

The implementation was done using the programming language Python and can be viewed at `https://github.com/joholle/Smart-Office-in-a-Shoebox`.
The above mentioned architecture design was implemented by an interface realizing the user layer, a PDDL planner implementing the reasoning layer, a Raspberry PI as ubiquitous layer and an Arduino unit connecting Sensors and Actuators in the physical layer.

4.1 Arduino

As outlined in the System Architecture Design, the Arduino component is responsible for receiving raw data, i.e., electrical signals, through its General-Purpose Input/Output (GPIO) pins. Each pin on the Arduino corresponds to a specific device, allowing us to easily interpret the data by knowing which pin

is associated with which device and understanding the impact of the electrical signals received.
For instance, various sensors and actuators can be grouped into logical units; an example of this would be combining a fan and a cooler to form an air conditioning system, which we refer to as a virtual sensor. By doing so, we simplify the management and control of related components. Once interpreted, the data from these sensors and actuators is reformatted into a JSON structure for consistency and ease of use.
The Arduino, by default, supports sending and receiving byte data through a SerialPort connection. Utilizing the "pyserial" package [6], the Raspberry Pi can interface with the Arduino via a Python script. This setup facilitates reliable and straightforward communication between the two devices.
On one hand, the Raspberry Pi can request sensor readings from the Arduino, and on the other, it can send commands to control actuators, such as adjusting servo positions. A crucial aspect of this communication is the defensive implementation of the interface, which ensures that illegal commands are filtered out and not executed, enhancing system stability and security.

4.2 Indirect Communication

In our smart office implementation, indirect communication between the Raspberry Pi and the laptop responsible for AI planning is facilitated using the Message Queuing Telemetry Transport (MQTT) protocol Mosquitto [9]. MQTT is an efficient, lightweight messaging protocol optimized for high-latency or unreliable networks, making it ideal for IoT applications.
The Raspberry Pi indirectly collects data from various sensors connected to the Arduino and slightly extends that with a timestamp. This data is already formatted in JSON format and can therefore simply be published to an MQTT channel.
The laptop subscribes to that MQTT channel and receives the sensor data in real-time. Using the previous described AI Planning method, the laptop generates an action, which then can be sent back to the Raspberry Pi through the MQTT protocol.
The Raspberry Pi, receives the high-level commands, i.e. the action and interprets it into specific commands for the Arduino. For instance, an action called "close_windows" would be translated into setting the degree angle of the servos, acting as windows, to 0°.
This cycle is repeated in regular periods and ensures real time decision making. In addition, complex AI-driven decisions are broken down into simple, executable commands for the Arduino, so not much further processing is necessary.

4.3 User Interface

The implementation of the user interface for our smart office system is developed using the Python Tkinter library [7], creating a practical and visually appealing means for users to interact with their office environment. The interface is

organized into several sections: sensors, actuators, weather, desired targets, and settings, each displayed in dedicated frames within the main application window (Figure 3).

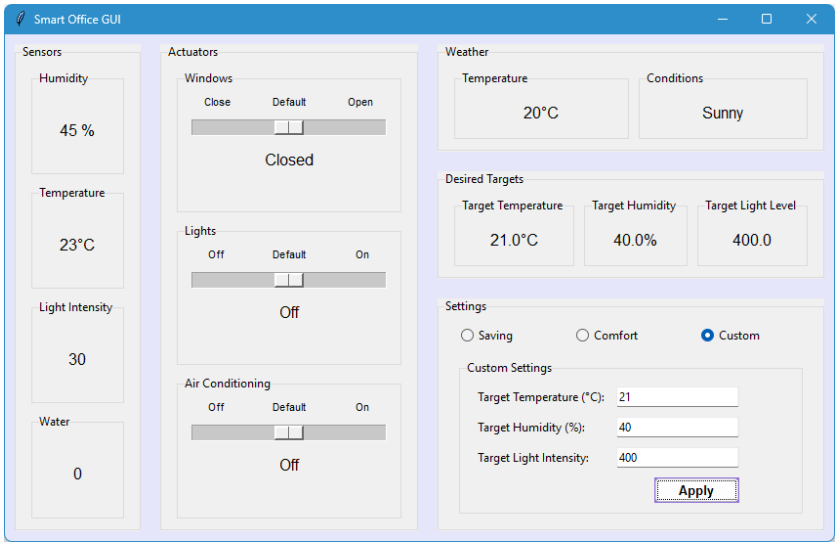

Fig. 3. GUI for the Smart Office

Users can see real-time data on environmental conditions such as humidity, temperature, light intensity, and water detection through the sensors section. Actuator controls are provided through sliders and toggle buttons that allow users to manually adjust the settings of windows, lights, and air conditioning. Beneath each slider the user is shown the current status of each actuator (Open/Closed or On/Off respectively). The weather section displays current external weather conditions and temperature. In the settings section of the user interface, three operation modes are offered to suit varying user preferences: "Saving" for energy efficiency, "Comfort" for enhanced user comfort at the expense of higher energy consumption, and "Custom" for user-defined settings. The "Saving" mode optimizes the environment to minimize energy usage by setting the temperature to 20°C, humidity to 50%, and establishing a light intensity threshold at 50 light units. If ambient light falls below this threshold, the system automatically activates additional lighting to maintain adequate illumination, effectively balancing energy conservation with user comfort. Conversely, the "Comfort" mode enhances user comfort by setting the temperature to a more pleasant 22°C, humidity to 40%, and a higher light intensity threshold of 70 light units, thereby creating an environment that may enhance productivity[10] but uses more energy to maintain these optimal conditions. Following these, the "Custom" mode allows users to specify their exact preferences for environmental conditions, in-

cluding temperature, humidity, and lighting, which can be set in the targets section of the interface (See Fig 3). This flexibility ensures that users can tailor the smart office environment to their individual needs, whether they prioritize energy conservation, comfort, or a personalized balance of both.
The backend methods of the interface are crucial for seamless communication between the user interface and the AI planning system on the laptop. These methods include setters and getters that handle the flow of data within the system. Setters are used to update the current status of the actuators and to refresh the values displayed from the sensors and weather API. This ensures that the interface reflects the most current state of the environment, based on both internal changes and external conditions.
The getters are needed to allow the AI planning program on the laptop to retrieve the target values set by the user as well as the current or forced states of the actuators. This is important for respecting user preferences in the automated planning process, where the AI adjusts the office environment according to both predefined user settings and real-time adjustments made via the interface.

4.4 AI Planning

The AI Planning component of our "Smart Office in a Shoebox" system is integral to autonomously managing the office environment by optimally performing actions such as adjusting lighting, controlling the air cooler, and manipulating window positions based on real-time sensor data and predefined goals. This component employs the Metric-FF [2] planner, a sophisticated tool designed for generating plans in domains specified in the Planning Domain Definition Language (PDDL) [3].

The system is equipped with a variety of sensors and actuators that continuously monitor and adjust the office environment. The sensors, including those for light, temperature, humidity, and precipitation, provide real-time data about the current state of the office. Additionally, a weather API supplies supplementary data, such as outdoor temperature and weather conditions, to enhance the system's responsiveness to external changes.
The actuators, which consist of a light switch, cooler, and window mechanisms managed by a servo, are responsible for executing the adjustments necessary to maintain the desired conditions within the office. The control logic for these actuators is driven by the plans generated by the AI Planning component.
The planning process begins with the creation of two PDDL files: the domain file and the problem file. The domain file, domain.pddl, defines predicates, functions and the available actions, including their preconditions and effects. Each action corresponds to a potential adjustment in the office environment, such as turning on the lights, adjusting the air cooler, or opening a window. The actions (see Table 2) are defined in such a way that the plan selects one action and then terminates successfully, ensuring that each plan is concise and immediately executable.

Action	Saving	Comfort
Light Actions:		
turn on light	$light < 50$	$light < 70$
force on light	*on user input*	*on user input*
force off light	*on user input*	*on user input*
Window Actions:		
open window	[*inside temp.* $> 20°C$ `or` *humidity* $> 40\%$] `and` *no rain outside* `and` *outside temp.* $>$ *inside temp.*	[*inside temp.* $> 22°C$ `or` *humidity* $> 50\%$] `and` *no rain outside* `and` *outside temp.* $<$ *inside temp.*
close window	[*inside temp.* $< 20°C$ `and` *humidity* $< 40\%$] `or` *rain outside* `or` *outside temp.* $>$ *inside temp.*	[*inside temp.* $< 22°C$ `and` *humidity* $< 50\%$] `or` *rain outside* `or` *outside temp.* $>$ *inside temp.*
force open window	*on user input*	*on user input*
force close window	*on user input*	*on user input*
Cooler Actions:		
turn on cooler	*inside temp.* $> 20°C$ `and` *inside temp.* $<$ *outside temp.*	*inside temp.* $> 22°C$ `and` *inside temp.* $<$ *outside temp.*
turn off cooler	*inside temp.* $< 20°C$ `or` *inside temp.* $>$ *outside temp.*	*inside temp.* $< 22°C$ `or` *inside temp.* $>$ *outside temp.*
force on cooler	*on user input*	*on user input*
force off cooler	*on user input*	*on user input*
No Action:	*negation of all other action preconditions*	*negation of all other action preconditions*

Table 2. List of all Actions

If no action is possible under the current environment conditions, an action called "no_action" is executed. This action requires all other action preconditions to be false and has no effects that influence the environment. It guarantees that the planner terminates fast and by doing that, makes it possible to call the planner iteratively in short succession.
The domain file is updated whenever the user inputs new desired environment variables. If the user selects a different mode, e.g. switches from *Saving* to *Comfort* mode, the thresholds of the actions change, making it necessary to update the domain.pddl file.

The problem file, problem.pddl, is dynamically generated based on the current state of the office and the real-time data from the sensors. This file initiates predicates and functions reflecting the current conditions, such as whether the window is open or closed, the status of the lights, and the exact temperature, humidity, and precipitation levels. By initializing the problem file with the latest sensor data, the system ensures that the generated plan is always relevant and accurate.
To automate the creation of these PDDL files, we utilize the Python PDDL extension [4]. This extension facilitates the seamless integration of real-time data into the problem file, enabling the system to rapidly respond to changes in the office environment. Once the domain and problem files are prepared, the Metric-FF planner is invoked to generate the optimal plan. The planner processes the PDDL files, evaluates the current state and the defined goals, and produces a sequence of actions that the system should execute to achieve the desired conditions.
Using this setup, we periodically create plans for our system. The sensors feed the system with environment data, causing the domain.pddl and problem.pddl files to be created. We then call the Metric-FF planner to create a plan and extract the next action. The internal state of the system is updated and the action is sent back to the actuators as an instruction, thus facilitating the systems reaction to the current environment.

4.5 Testing

To test the functionality of our system, we implemented a Tester class in Python. Our aim was mainly to test the correctness of the AI Planning used in our project. The Tester class feeds our system with artificial inputs, changing over time. Instead of using the actual sensor data, manual user inputs and weather API data, we used the Tester class to create an artificial changing environment for our system.
We conducted tests regarding the light sensors and actuators and the temperature, humidity, precipitation and the window and cooler actuators. For each test we first changed the artificial environment in such a way that would let the system switch on the light, open the windows, and so on. We then forced the system to reverse this by giving an artificial user input. After each test we reset the system to its original state.
Our testing yielded positive results, AI Planning worked as intended and the changes in the artificial environment incited the system to the correct responses. Another critical part of the system, as seen in the exceptions of the use cases is the correctness and functionality of the sensors and actuators. We tested the sensors thoroughly, comparing them to other sensors to ensure that they are reasonably correct. The actuators functioned well throughout the implementation and consecutive testing of the shoebox.

5 Discussion and Conclusions

The "Smart Office in a"Shoebox" system effectively showcases how IoT and AI can transform office environments into energy-efficient and comfortable spaces. Key strengths include its modular architecture, which separates low-level and high-level processing between the Arduino and Raspberry Pi, and its user-centric interface that allows for manual interventions and real-time adjustments.
However, several limitations were identified. The reliance on continuous internet connectivity for the weather API and MQTT communication introduces a potential point of failure. Additionally, while the system's reliance on AI planning is innovative, it may not always account for unpredictable human behavior or environmental changes that could affect office comfort and efficiency.
Future enhancements could focus on improving the robustness of the system's connectivity, incorporating machine learning to better predict and adapt to user behaviors, and expanding the range of sensors and actuators for more comprehensive environmental control. Despite these limitations, the project successfully demonstrates the potential for smart office technologies to reduce energy consumption and improve workplace wellbeing, laying a foundation for further research and development in this field.

References

1. International energy agency - buildings, https://www.iea.org/energy-system/buildings
2. Metric-ff planner, http://fai.cs.uni-saarland.de/hoffmann/metric-ff.html; https://github.com/Vidminas/metric-ff-crossplatfor
3. Pddl, https://planning.wiki/ref/pddl
4. Python pddl parser for pddl 3.1, https://github.com/AI-Planning/pddl
5. U.s. energy information administration - energy use in commercial buildings, https://www.eia.gov/energyexplained/use-of-energy/commercial-buildings.php
6. pyserial documentation (2024), https://pyserial.readthedocs.io/en/latest/pyserial.html, accessed: 2024-07-17
7. Tkinter — python interface to tcl/tk (2024), https://docs.python.org/3/library/tkinter.html, accessed: 2024-07-17
8. Knave, B.: Ergonomics and lighting. Applied Ergonomics **15**(1), 15–20 (1984). https://doi.org/https://doi.org/10.1016/S0003-6870(84)90117-0, https://www.sciencedirect.com/science/article/pii/S0003687084901170
9. Light, R.A.: Mosquitto: server and client implementation of the mqtt protocol. Journal of Open Source Software **2**(13), 265 (2017)
10. Seppanen, O., Fisk, W.J., Lei, Q.: Room temperature and productivity in office work (2006)
11. Zhang, X., Zheng, P., Peng, T., He, Q., Lee, C., Tang, R.: Promoting employee health in smart office: A survey. Advanced Engineering Informatics **51**, 101518 (2022). https://doi.org/https://doi.org/10.1016/j.aei.2021.101518, https://www.sciencedirect.com/science/article/pii/S1474034621002664

All links were last followed on July 17, 2024.