Using PrudensJS with a Raspberry Pi for a Smart Home Application

Yiannis Michael

September 1, 2023

Abstract

Through this project, the possibilities of using PrudensJS with a Raspberry Pi (RasPi) were explored. A variety of different hardware were considered but due to local availability, RasPi 400 model was selected. Initially the system started with a distance sensor and a single LED, while it slowly evolved into multiple sensors and actuators working at the same time. An Analogue to Digital Converter (ADC) was also implemented to allow for the implementation of analogue sensors. It was programmed using the Python language as well as using libraries that allowed the use of the GPIO pins and the Serial Peripheral Interface (SPI) communication protocol. The role of the developer was made only to provide the proper infrastructure to the technician, who will assist the end user with setting up this Smart Home system. Driver files would be provided by developer or manufacturer to allow for new devices to be implemented into the system. The project was successful with the final result including five sensors and four actuators with nine Prudens policy versions, thus demonstrating the possibilities of such a system. More features that were not outlined in initial plans were also implemented.

1 Introduction

To create such a product, multiple iterations were made and improved upon. Initially, background and market research defined the possibilities for the project proposal, and its aims. Then, a simple proof of concept was engineered onto which more features were slowly added on. The Smart Home application was always considered at each design stage. This report is focused on describing the ideas and methods, therefore it is strongly recommended to read the User Guide to fully understand the capabilities of the system as well as the ideas described here.

1.1 Background and Market Analysis

Prudens is an argumentation-based language for the design of cognitive assistants. Providing it with a Policy and Context, it can deduce certain conclusions that can then be used for any specific requirement. More detail as to how it works can be found in the corresponding research papers. [1] [2]

Existing Smart Home concepts like Google Home and Apple HomeKit already explore the possibilities of having all home devices connected to a hub where they can be controlled by the user. In Apple's case, only devices in the iOS ecosystem can be used, thus making the HomeKit very limiting. Google on the other hand, allows a vast amount of devices to connect to its main hub, but even this comes with security concerns due to multiple third-party companies having access to personal data. Both of these Smart Home applications use an Internet of Things (IoT) system, which facilitates the connectivity to the main hub [3].

There were not many choices of hardware due to the specific case of this project. The system chosen should be able to use JavaScript in order to interact with PrudensJS, or at the very least have internet connection to be able to connect to a server and send requests. For faster implementation the preferred programming language was Python, therefore this ruled out many microcontroller options. One case was the Arduino Nano 33 BLE Sense [4] which uses MicroPython and was designed for creating projects with AI. Another option was the ESP32 [5] which is a more traditional microcontroller designed by Espressif for AIoT (AI and IoT) applications. Evidently, availability in the Thinker Maker Space also affected the hardware options, and a RasPi 400 [6] was selected.

1.2 Project Proposal and Aims

Through researching and analysing the possibilities, project goals slowly became more defined. Therefore, it was decided that a RasPi 400 would be used alongside Prudens, to create a Smart Home AI Assistant. The RasPi would execute the main Python script as well as PrudensJS using NodeJS. The developer would create the architecture to allow for a technician to setup the Smart Assistant for the user. The technician should not be required to write large amounts of code in order to implement a certain sensor or actuator, and the driver files would allow for additional implementation of peripherals when required by the user. The driver files should also allow for easy installation. Furthermore, due to no Natural Language interpreter for Prudens, a button would be used to symbolise user input. A User Guide would also be created to give more insight as to how to execute certain actions.

2 System Description

2.1 RasPi 400

The RasPi 400 system boasts 40 GPIO pins of which almost all of them were used for the project. The absence of an ADC mandated the use of an external ADC chip (MCP3008) using its SPI capabilities (SPI0). A second SPI bus (SPI1) was used to communicate with the TMC5160-BOB. The keyboard is also used to provide a fake user input to symbolise a new Rule being created, and the USB ports are used to automatically copy the driver files. To observe system output, a screen with HDMI connectivity is required to have the output on the terminal. It is also possible to use the system with no screen and only the user input button. Please read the User Guide for more information.

2.2 Supported Devices

Supported devices so far are,

1. Button
2. HC-SR04 Ultrasonic Sensor [7]
3. LEDs
4. RGB LEDs
5. Analogue devices (with MCP3008),
   (a) Potentiometres
   (b) DFR0023 LM35 Linear Temperature Sensor [8]
   (c) DFR0026 Ambient Light Sensor [9]
6. TMC5160-BOB Stepper Motor Controller [10]

2.3 Breadboards

In this prototype there are three breadboards each one for a different usecase in the system. Splitting the functionalities aids with presentation and is more user-friendly. Please read the User Guide for the diagrams.

2.3.1 Sensor Board

This board (shown Figure 1) is responsible for any sensor data input into the system. It features a HC-SR04 Ultrasonic Sensor, a button, a potentiometre, a DFR0023 LM35 Linear Temperature Sensor, and a DFR0026 Ambient Light Sensor. All analogue sensors send analogue data to the MCP3008 which is connected by the SPI bus to the system. In a practical scenario the user would purchase a new sensor or actuator and a technician could assist with installation if required. A voltage divider is also present for the HC-SR04 output (100 Ohms and 220 Ohms for voltage divider which gives 1.56 V at GPIO pin).

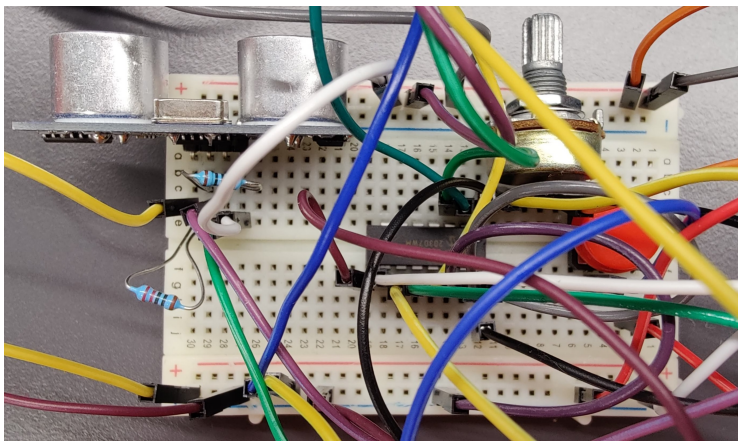

Figure 1: Sensor breadboard with everything connected.

2.3.2 Actuator Board

As the name suggests, this board (shown Figure 2)contains all of the actuators of the system. They are controlled by the Prudens Policy and the literal they require to actuate. Installed is one green LED, one white LED and an RGB LED.

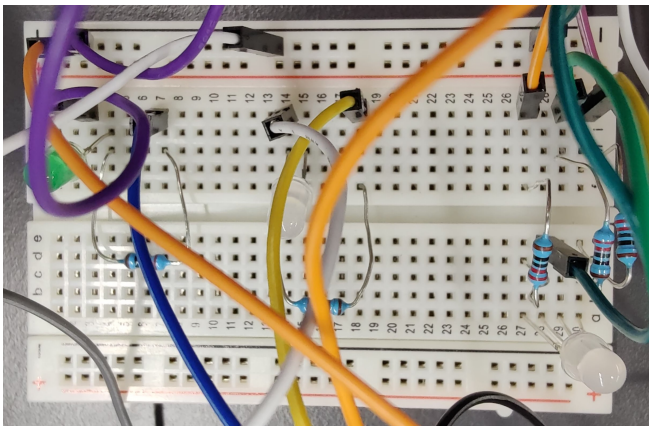

Figure 2: Actuator breadboard with everything connected.

2.3.3 User Input Board

Symbolising the user input is necessary for the system, which is why this board (shown Figure 3) is required. In a real scenario, the user would add new Rules to the Policy by describing new functionality for the sensors or actuators. Such a user input is symbolised by the blue button which iterates through each Policy version, thus adding new rules to the system for more functionalities.

A system restart can be initiated by pressing the smaller button on the board. This allows for any changes made to the technician file to be loaded without a full system power cycle.

2.3.4 TMC5160-BOB

Using this breakout board (shown Figure 4), the TMC5160 was connected with a stepper motor and therefore could interact with the RasPi 400.

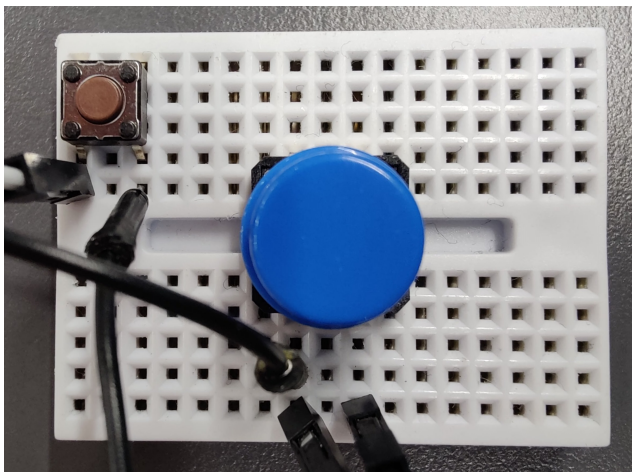

Figure 3: User input breadboard with everything connected.

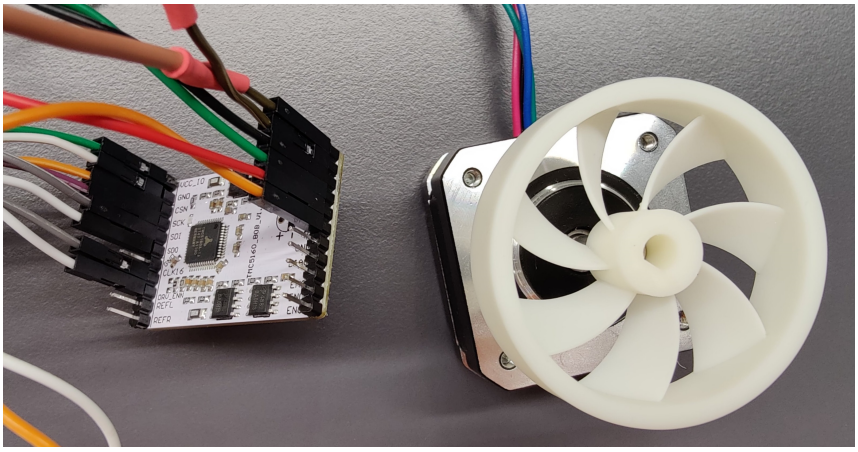

Figure 4: TMC5160-BOB with all connections.

2.4 Parts List

For this project to be recreated the following parts are required,

1. 1 x RasPi 400 [6]
   (a) 1 x Mini-HDMI to HDMI cable for monitor connection
   (b) 1 x Mouse for easier navigation
   (c) USB-C power supply
2. 2 x Half-sized breadboards and 1 x Mini breadboard
3. For Motor,
   (a) 1 x TMC5160-BOB (soldered on heads) [10]
   (b) 1 x Stepper Motor [11]
   (c) 1 x 3D printed fan attachment (on the GitHub repository)
   (d) 6 x female jumper wires soldered on ends of motor and power supply
   (e) 6 x heat shrinks for soldered wires to ensure safety
   (f) 8 x female-to-female jumper wires for RasPi-TMC5160 connection
   (g) Power Supply for TMC5160 (Used the LRS-150-12 [12])
4. For Actuator board,
   (a) 1 x RGB LED
   (b) 1 x Green LED

(c) 1 x White LED

(d) 5 x 220 Ohm Resistors (resistances are only for protection of the LEDs, any values can be used and should not be very large)

(e) 6 x male-to-female jumper wires

(f) 3 x male-to-male jumper wires

5. For User Input board,

   (a) 2 x Buttons

   (b) 1 x male-to-male jumper wire

   (c) 2 x male-to-female jumper wire

   (d) 5 x 220 Ohm Resistors (resistances are only for protection of the LEDs, any values can be used and should not be very large)

   (e) 6 x male-to-female jumper wires

   (f) 3 x male-to-male jumper wires

6. For Sensor Board,

   (a) 1 x HC-SR04 [7]

   (b) 1 x Button

   (c) 1 x Potentiometer

   (d) 1 x DFR0023 Temperature Sensor [8]

   (e) 1 x DFR0026 Ambient Light Sensor [9]

   (f) 1 x MCP3008

   (g) 1 x 100 Ohms and 1 x 220 Ohms for Voltage Divider (gives 1.56 V at GPIO pin)

   (h) 10 x male-to-male jumper wires (6 for DFR sensors)

   (i) 10 x male-to-female jumper wires

7. 5 x Cable Ties for cable management.

3 System Roles

In order to create a functioning and organised Smart Home assistant with Prudens, the different user roles for the system had to be considered. They are listed and described in detail in this section.

3.1 Developer

The role of the developer was set to only create the foundation that the technician and the user will build on. They create the interface for the sensors, actuators and Prudens to work together.

3.2 Technician

A technician would assist the user in installing and instantiating a new sensor or actuator in the system. Using a user guide, a minuscule amount of Python code would be written in order to allow the sensor to interface with the system. Driver files with setup functions and action functions are required for a new device to be used.

3.3 End User

The person using the product in their home should not have to care about the technicalities of the product, and should work up to the limits of the specification. For example, if only three ADC channels are left, then they should only be able to connect three new analogue sensors to the system. They can also be helped by the technician for troubleshooting or installing new peripherals. The new sensors or actuators should arrive with their respective drivers, in order for them to be setup correctly by the technician.

4 File and Folder Description

The code for the project are divided into different files and directories for organisation and ease of implementation. All of the GPIO implementation was done using the RPi.GPIO library. Some errors may appear using the cloned repository due to the file paths, please see user guide to resolve such issues.

4.1 Drivers Folder

This subdirectory contains all of the necessary driver files for a sensor or actuator to work. These have been coded by the developer without any reliance on external GPIO libraries except RPi.GPIO. Each file is specific to the device in the file name, considering that when such a device is purchased it would come with a driver file for the system. Some devices, like those connected to the ADC, have similar drivers but for other devices (i.e. other Distance Sensors) would require different driver files to be coded by the developer or the manufacturer. New driver files can be automatically copied if the USB is named *DRIVER_USB* and has a */RasPi-PrudensJS/drivers/* directory. This allows along with the restart button, allows for an uninterrupted installation of new devices. Please read the User Guide for more information regarding this.

4.2 PrudensJS Folder

In this location where the Prudens repository should be placed in order for system interaction to occur. It uses a modified version of PrudensJS and it requires specifically the *node/app.js* file to run using NodeJS.

4.3 Text Folder

The system is programmed to cycle policies on button press, therefore this folder contains all of the available Policy versions the system will cycle through.

4.4 classes.py

Contains the classes and class methods used by the system. They are split into *Sensors* and *Actuators*. The setup functions for both, only allow up to four-pin devices to be setup in the system. The max pins used currently by a device is the RGB LED which uses three. Similarly to the setup functions, the action functions, which are the functions that execute certain functionality when the sensor or actuator is activated, also allow up to four pins to be used.

An exception to this is when a sensor is assigned an ADC channel and the ADC function, rather than a list of pins and an action function. This is because analogue sensors only send data to the ADC, so the data can only be read and cannot be interpreted differently like with other sensors. For example, a button can be held or pressed once and the corresponding function will activate.

The *Sensors* class has an extra function that the *Actuators* class does not, and that is due to the requirement of inputting data into the literal to be interpreted by Prudens. The *data_in_literal()* function, finds if there is the letter X in the literal and if successful, it assumes the data is numeric and replaces X with the reading. Only binary outputs are accepted by the system for the actuators, therefore there is no current need for such a function in the *Actuators* class.

4.5 sys_fcns.py

Some functions are specific to the system and not used by any device for interaction with the software. They are helpful for the developer and provide better code organisation. Functions for the Prudens interaction, ADC reading, system exit, system restart, driver USB file copying, and debugging assistance are all located here.

4.6 tech.py

The technician will only access this file to instantiate a new sensor or actuator. Only a single line of code is required for the system to use a new device. Both lists here are used in the developer file to setup the devices, execute their functions, and to read or write Prudens data. The format for installing a device is explained in the User Guide.

4.7 dev.py

In the dev.py file, is where the developer connects all of the elements of the system. A while-loop is used to provide unlimited linear iterations of the program logic. Only the developer should access and modify this file. Firstly the devices are setup using their corresponding class method and then it executes the main function. Python threads is used as to allow for text to be inputted by the user at any point. The main function combines the *program_thread()* and *user_input_thread()* functions in a try-block so as to then exit gracefully in the finally block using the *sys_exit()* function. It first checks for a button press to change the current policy version, then it checks if the restart button is pressed. Subsequently, it checks if there is a USB directory to copy from, then the sensor readings and the literal processing occur. After it received the conclusions from Prudens, it checks each result to check if the conclusions correspond with the literals setup for the actuators in the technician file. If they do then the actuator action is executed but if they don't then nothing happens. This then repeats until the program is terminated using Ctrl+C.

4.8 Other Files and Folders

Some other files are included but not necessary for the system to run. There are the *tests*, *docs*, *setup*, *fan files*, *documentation*, *setup*, and *blank_system* files or folders. Most are self-explanatory but a brief description for them is on the GitHub repository [13].

5 Evaluation and Future Work

A lot can still be done with the system, as it is still a prototype. Exploration with different libraries, like GPIOZero, may yield better results. However, it was not used due to the data each object had to be associated with (e.g. literals), in order for the system to work as planned. A different implementation could use GPIOZero but it would certainly be limited by the supported devices, and creating a custom object could prove to be complicated. Built-in error checking could halt the system, whereas the current implementation allows to modify it as problems arise or even ignore them. Another library to use could be wiringPi, which is a C library written specifically for the RasPi SoC devices. This could provide faster speeds and more real-time inputs and outputs, therefore a future implementation of that can also be considered. For ease of development and as a proof of concept, this project was coded in Python.

Initially the RasPi 400 was not the best candidate for the project, its 2 SPI buses, USB ports and keyboard functionalities were fully utilised thus allowing it to be ideal for this prototype. Other microcontrollers, which would allow the ideas to be implemented in the real world, can still be explored. One may argue that a RasPi is not the perfect solution for a real-time AI Smart Home Assistant, due to there being an OS layer between the project software and the CPU. Another microcontroller with MicroPython and internet capabilities may be better suited for future implementations. The peripherals also may need adjustments (e.g. in the case of the HC-SR04) to communicate with the RasPi safely.

Even though the program is a functioning prototype that shows a lot of potential, there are numerous limitations that should be remedied or considered. For example, driver files have to be coded by a developer or manufacturer to be used by the technician or the user. Such an ecosystem does not currently exist and therefore has to be done manually for each device, by the developer. Furthermore, each actuator action has to have a separate actuator object instantiation, due to the different literal for each action function. This is not a problem on its own but it can cause confusion when the system is expanded into tens of devices. Lastly, there is currently no way of providing a natural language user input, and having it convert to the format Prudens understands, for Policy or Context creation. This is the reason the program iterates through different Policy versions, which contain the updated rule set.

6 Conclusion

To conclude, this paper alongside the User Guide, describe how a RasPi 400 and PrudensJS were used to create a Smart Home AI Assistant prototype. The final result accepts digital sensors, up to eight analogue sensors, digital actuators and a TMC5160 motor driver. Although it has its limitations, the outcome exhibits great potential and allows for vast exploration of its new capabilities in the future.

References

[1] V. T. Markos and L. Michael, "Prudens: An argumentation-based language for cognitive assistants," *ResearchGate*, 2022.

[2] L. Michael, "Machine coaching," *ResearchGate*, 2019.

[3] J. Bitner, "Google home vs. apple homekit: which is the best smart home platform?," 2023.

[4] Arduino, "Arduino nano 33 ble sense," 2021.

[5] E. Systems, "Esp32," 2016.

[6] R. Pi, "Raspberry pi 400," 2020.

[7] Sparkfun, "Hc-sr04," 2017.

[8] DFRobot, "Dfr0023 temperature sensor," 2000.

[9] DFRobot, "Dfr0026 ambient light sensor,"

[10] Trinamic, "Tmc5160-bob," 2019.

[11] O.-S. Online, "Nema 17 bipolar stepper motor,"

[12] M. Well, "Lrs-150-12," 2019.

[13] Y. Michael, "Raspi-prudensjs," 2023.