Documentation ASUS Robotics & AI Center Challenge

Ghormeh-Sabzi

7 November 2020

1 Introduction

This file serves as documentation for the project presented by our team "Ghormeh-Sabzi" consisting of four members: Afra Amini, Andisheh Amrollahi, Rouzbeh Hasheminezhad and Amin Moghaddam, presented at PolyHack 2020. This documentation is a more high level overview serving the purpose of easing the transition of the reader to the README and source code of the project.

2 Documentation

2.1 System architecture

2.1.1 Sensors and Actuators

We have provided a variety of sensors already implemented and also a means for the user to implement their own sensors. The current sensor implementations by us includes a motion sensor, a proximity sensor and a noise sensor. Through the UI the user can add as many of each types of these sensors as they want to their customized smart city, see Figure 1. The sensors interact with the main server via sockets. We refer the reader to the "Object entities" section of the README for the API for implementing their own sensors. In figure 2 we show how a sample smart city looks like. We allow the user to enter values for sensors either manually or toggle to a random mode simulating a real environment. For example for the proximity sensor we use an exponentially distributed random variable modelling a Poisson process for the arrival times of objects close to the sensor.

Regarding actuators we have provided a door and lamp. If the user need implement their own actuators they can refer to the "Object entities" section of the README for the API. The actuators receive "acts" through sockets from the main server.
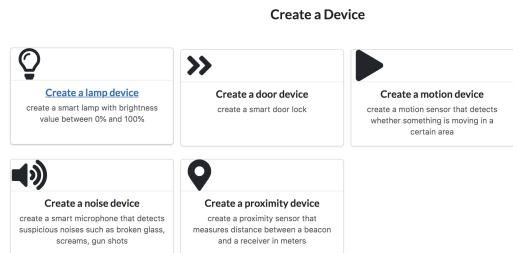
Figure 1: UI for creating sensors and actuators
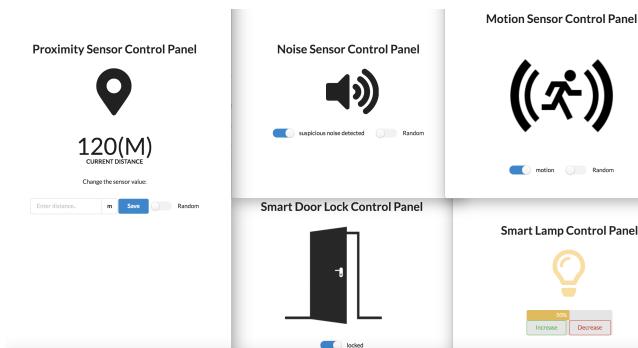
Figure 2: A small smart city

State of Devices

Device ID	Type	State
5065329edf5648868afa71b34c2ebd20	proximity	120
a704aea69e9845bfae9fc7097ba2ed34	noise	true
2c300b9bcd614ca98999fb6af142af12	motion	true
d7012a653187456d8205a0de2a2bd940	lamp	0.5
97a7cf10b09047bc83605d7a7ce2e7e0	door	true

Figure 3: Control panel showing status of all devices (sensors and actuators)

2.2   The central server

Change of values in sensors fire events which are transmitted to the central server via sockets. The central server maintains a list of rules that each of the sensors is dependent upon. When a new value for a sensor is received all rules are reevaluated and actions are taken if those rules evaluate to true. Here actions consist of sending commands to actuators.

The central server also interacts with the front end and provides the user with a control panel in which they can view the current status of all the devices (sensors and actuators, see Figure 3). Internally the server keep implements classes "Actuator" and "Sensor" to keep track of each Sensor and Actuator. Each new specific sensor or actuator , for example a lamp most inherit from these classes. Please read the README for further details.

2.3   Rule Engine

2.3.1   Internal data structure

Internally each rule is kept as tree on the central server. Each internal node is either an "any" or an "all" node. Each leaf is a Boolean expression that can be evaluated to true or false by querying states of sensors or actuators. The value of an "any node" is true if any its children evaluate to true. The value of an "all" node is true if and only if all its children evaluate to true. Each leaf represents an "atomic" simple expression. These are typically checking if actuator or sensor states are equal to less that or greater than some constant values.

If a rule evaluates to true at any time point a list of actions provided by that rule are taken. Each action consists of setting the state of an actuator to a constant value. Actions are also specified through the UI

2.3.2   Rule Engine UI

We provide easy access for the user to make rules through an intuitive UI. The indentations provide the user with a better visualization of how the logic is nested.
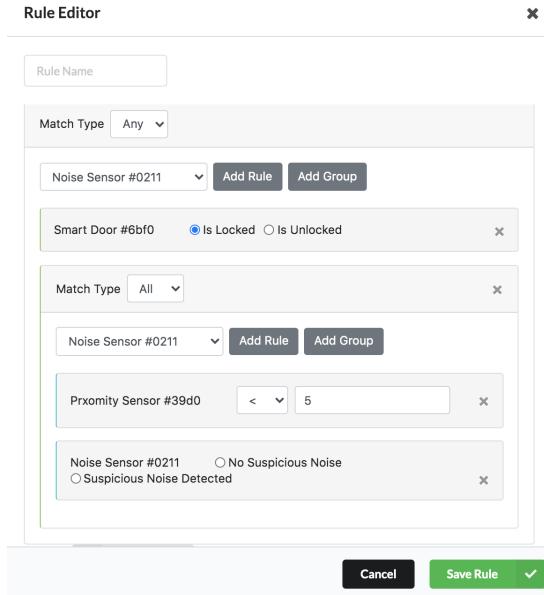
Figure 4: UI for new Rules

3   A special/happy feature

We implement a sensor that uses a light-weight machine learning model to detect how happy a face is. By sending this happiness value from the frontend to the backend, we can incorporate this sensor into our system, including the user-defined rules.

In a video presentation, we showcase an example of this. This feature demonstrates our approach and system architecture's capability to handle higher-level and resource-wise, more demanding sensory inputs with ease.