Name: Pransh Rana Date: 28 Dec 2022

Process Documentation

A waterfall design methodology was used to carry out this project. An incremental approach in testing was deployed.

Requirements

- Simple Simulation of turtle bot style robots in an n x n grid
- Multiple robot controller support
- Asynchronous enqueue of commands for each bot independent of other robots motion
- Modular library Implementation
- Show a unit test for a robot going in a 1 X 1 Square

Assumptions

- The Robot exist within a discrete, finite (N x N) grid; each grid cell may only contain one bot at a time.
- It takes each robot a non-zero period of time to travel between grid cells; during this period, assume the robot occupies both the source and destination cell.
- Movement is limited in NEWS direction
- The robot controller will take inputs from the keyboard
- The map array will be shown in some form of GUI

Specifications

- The software in the loop simulation will use a client-server architecture.
- The main logic for the robot motion interacting with the grid will run on the server and multiple robot clients will connect to the server and provide their controller commands.

- The server will update all the connected robot instances with the current grid information.
- If the server dies, all the connected clients will wait for 5 seconds and then exit
- If the client dies, the server will wait for 5 seconds and if no response is recorded, the robot will be deleted from the server and the occupied position associated to the robot will be set to vacant.

System Design

Python programming language will be used

SITL Server

- have an SITL class and a robot class.
- client and server architecture will be implemented with MQTT library, open source mosquitto server will be used
- grid will be implemented using an N x N array.

Robot Controller Instances

- have a way to input control commands using keyboard.
- have an mqtt client object to send requests to the server
- receive the current map array and display it using kivy library, print to terminal incase not enough time

---

Implementation

SITL Server

mqtt_broker()

- Class used to create the broker object
- server listens on sitl/cmd topic and writes on robot/pos topic

sitl_map() Class

- Main class that contains the map array and its methods are used to manipulate the array. It uses a number of get_*() and set_*() functions to manipulate attributes of the object.
    - On Init
        - user is prompted for a size for the grid, which is used to create an NXN array and and is stored as an attribute for further manipulation.
    - Process queue method is run in a while loop which process all the navigation requests coming from the robots and carries out logic by calls other modules

robot() Class

- Class used to instantiate robot objects, used by sitl_map() class. The created objects are used to track position of each robot and are updated according to the navigation commands from the clients

boradcast_map()

- The function is run on a thread and broadcasts the array to the clients

Robot Interface

MapView() Class

- A Kivy gridlayout class, can be used to add GUIs, but will only be used to take inputs from the keyboard and send to the server.
- Methods to show the current map array, prints directly to the console
- Methods to ping server and check if the server is active

- Method to perform a unit test
- On Init
    - the user is prompted for a robot name, after which the user can chose to carry out a unit test or directly command the robot using the W,A,S,D commands for up, left, down, right movement.
    - on first navigation command, init robot request will be sent to the server.

turtleApp() Class

- kivy app class used to create the GUI

Routines

ROBOT ADDITION

1. When Robot interface is instantiated, the user is asked for a robot name.
2. Once the name is saved as a class attribute, the first navigation command input from the keyboard sends a request to the server to create a robot object of the same name.
3. On the server , the request is processed and the add_robot() method is called.
4. A robot object is created and an integer ID is assigned to it.
5. A spawn position on the map array is picked and is assigned to the robots' current and target position.
6. The robots' ID is used to assign the value at the position in the map Array, hereby making the robot visible.
7. The object is saved in a dictionary with key as the ID.
8. Once the add_robot() method returns success to the robot interface, the ID is used to identify any navigation commands sent from that robot instance

NAVIGATION PROCESS

1. Once the client is assigned an id, the keyboard keys W,A,S,D send navigation requests to the server.
2. When the server receives the navigation requests, it will be put into a python list associated to the SITL_map() class object. This list is treated as a queue.

3. The navigation command is identified using the ID and robot associated with that ID is updated for new target location.
4. The process_queue() method is called in a while loop
5. All the navigation commands are processed and the robots are moved from their current location to the target location.
6. The target location index are assigned the id of the robot and the current location is assigned a negative of the ID of the robot, so that no other robot can move into it while the move is commencing
7. A method to free the negative assigned cell of the array is set to zero after time **dT**, which is the time defined in the requirements.

ROBOT DELETION

1. If the robot interface dies or the user presses “X”, the delete_robot() method of the SITL_map() object is called and the robot is dropped from the dictionary, and its ID is also deleted from the active robot lists
2. In case the robot was killed by pressing “X”, resending the navigation requests will make the server respawn a robot on the map, but the ID will be different, though the name will be the same.

The complete system architecture is displayed in SYSTEM_ARCHITECTURE.PNG file