Scientific Python

IoT indoor positioning system for disabled people

# Project dEstriptcion

The aim of the project is to develop a tool, which makes independent living easier for the blind and visually impaired people. The basic idea is to locate essential objects in indoor areas. There are four sensors which measure the distance of a specific object, then transfer the data to a central server. The server calculates the exact position depending on the signal strength of four sensors. These coordinates are uploaded to a database. The user-end of this tool is an online GUI, since voice commands are out of the objects of this project.

# Implementation

Since the project has to be implemented in python language and the available devices on the market mostly supported C++ language, we simulate the IoT part of the system. To simulate we use a GUI where the tester can control the type and the place of the objects in the indoor area.

The server uses RethinkDB as a nonSQL database for data storage and archiving. The advantage of this tool is that the queries can subscribe to a particular table and whenever the table is changed, the queries and the client’s data automatically updates.

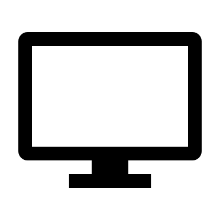
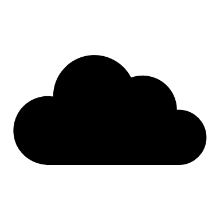
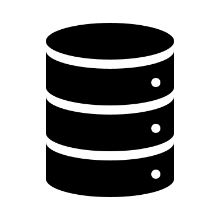
The user-end interface uses a similar GUI as the simulated sensors but instead of set the objects, users can identify the type and the current places of the objects.

# Task distribution

Bence Keömley-Horvat is in charge of the IoT simulation, with the use of ‘blueprint.gif’. The communication between the IoT device and the server has to be via JSON files. The JSON file should contain id, type, measured\_distances fields.

Balázs Lükő is the leader of server implementation. The server calculates the x, y positions from the measured\_distances. Data is uploaded to the database server in JSON format as well, containing id, type and coordinates fields.

Péter Kovách’s main responsibility is to implement the user-end interface using ‘blueprint.gif’. The program loads the data from the RethinkDB. The objects are visualised on the blueprint, with various colour, depending on the type of the opject.



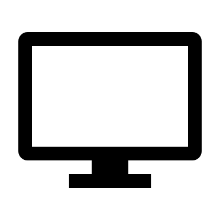
QUERY

JSON I

JSON II

RethinkDB Server

* Calculate position
* Upload to DB
* Store data
* Notify queries



User-end interface IOT simulation

* Query data
* Visualise objects
* Type & position simulation
* Send JSON to server

# documentation

## User-end Interface

The source of user-end interface is the Client.py script. The GUI has a blue script part where the user can track the marked objects and has a filter panel where the user can choose which objects should displayed on the blue print.

### Implemented Functions and Classes

#### Client class

This class implement the GUI (children of tkinter’s Frame) and the functions and data structures of the user-end interface.

#### \_\_init\_\_()

Input arguments: master

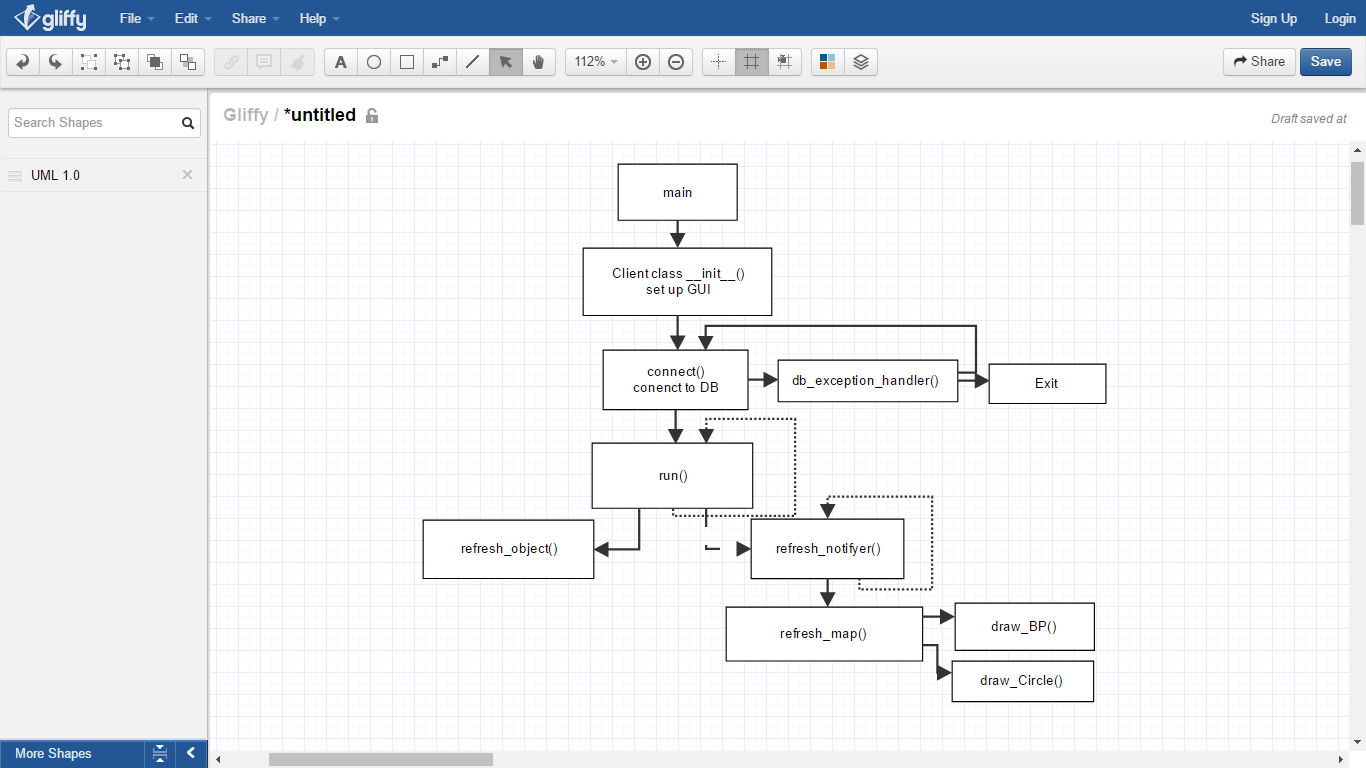
master: set the master frame to root.

Init generate the GUI and try to connect to the database.

#### set\_changes()

Whenever the GUI needs to be updated this function raise a flag, which result in GUI update.

#### connect()

Try to connect to the RethinDB database.

#### db\_exception\_handler()

Visualize the generated errors and try to reconnect or close the application.

#### draw\_BP()

Draw blue print

#### draw\_circle()

Draw circle onto the blue print in given colour at a given position.

#### refresh\_cursor()

Try to refresh the query from the database. If not possible call the db\_exception\_handler() function.

#### refresh\_object()

Refresh object from the database.

#### refresh\_map()

Refresh blue print and the selected objects.

#### run()

Actualise the objects positions from the database.

#### refresh\_notifyer()

This function call a refresh event in every 0.5 second when the changes flag is true.

## Server

### Functions in server implementation and position calculation:

#### Calculate()

It is a function for calculating the possible positions for an object, based on it’s distence from two reference points (which are two simulated sensor). So basically this function calculates the intersection points for two circles.

Parameters: sensor1pos,sensor2pos,r1, r2

sensor1pos: the (x;y) coordinates of the firs simulated sensor

sensor2pos: the (x;y) coordinates of the second simulated sensor

r1: distance of the object from sensor1

r2: distance of the object from sensor2

Return value: result

result: it is a vector containing the two possible (x;y) coordinates

#### position()

It is a function for extracting the correct (x;y) position of an object given all the possible, previously calculated points.

Parameters: coordinates

coordinates: it is a vector containing all the previously calculated circle intersection points

Return value: position

position: it is a vector containing the correct x and y coordinates

## Sensor simulation

Due to the limited resources, we had to make a sensor simulator (and the objects too of course). This is a GUI in which the user can place the object in the layout of the house and after that it will send the data in json format to the server.

### Implemented classes and functions

#### Obj class

This will store the data of the object, namely the position, name, colour and the circle, button tkinter variables.

#### \_\_init\_\_ (in the Obj class)

This is the constructor of the class, it gives name and color of the object.

#### get\_Colour

It returns a colour according to the given index

#### Window class

This class implement the GUI (children of tkinter’s Frame) and the functions and data structures of the sensor simulation interface.

The "objects" variable is an obj type vector. The "types" contains the names of the objects and the selected object is the index of the active object.

#### \_\_init\_\_ (in the Window class)

This constructor creates the GUI itself. Initiating the objects, creating the blueprint, panels, image, buttons, label, grid.

#### draw\_BP

It creates the image.

#### Deselect

Command of the deselect button it sets the selected\_object to -1 (this means that no button (object) is selected) and reactivate the previously disabled button.

#### select\_Object

Command of the objects's button. Reactivate the previous selected button and disables the new button. And changes the selected\_object to the index of the object.

#### draw\_Circle

It is activating when the user clicks on the image. It creates a small circle and safes its positions.

#### del\_sel

Deletes the circle and its position of the selected object

#### del\_all

Deletes the circle and its position of the all object

#### get\_distant

Called from the make\_json function, calculate the distances from the sensor of the given object (o variable).

#### make\_json

Called from send, creates the json\_object that will be sent to server.

#### send

Command of the send button. It try to connects to the server and sends the data in json format.