Intelligent Trash Bin Management System – initial design and implementation

Piotr Filarski[1], Piotr Niedziela[1], Maria Ganzha[1,2], and Marcin Paprzycki[2]

[1] Warsaw University of Technology, Warsaw, Poland,
filarskip@student.mini.pw.edu.pl
[2] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
name.surname@ibspan.waw.pl

Abstract. One of interesting aspects of the Internet of Things (IoT) is development of systems where sensors (and possibly actuators), allow creation of inexpensive human-oriented solutions. In this context, we consider "smart garbage collection". Specifically, we have developed an inexpensive prototype of a solution for poor communities (possibly Third World countries). The proposed approach, consists of a trash bin prototype and an initial version of an application that allows optimisation of trash pickup.

Keywords: Internet of Things, smart trash bin, garbage collection management

1 Introduction

As prices of sensors, actuators and computer hardware decrease, it is possible to introduce inexpensive human-focused solutions. One of areas, where such solutions are highly desirable, is garbage collection. Here, typically, garbage is collected at certain days and times, e.g. every Monday, Wednesday and Friday, before noon. This means that, each time the garbage truck is travelling the same road, regardless if there is a reason to do so, or not (i.e. it will come to pick up trash even if the trash bin is only half full and does not generate unpleasant odours). As a result, fuel is consumed and unnecessary pollution generated. At the same time, if the trash bin is full already on Wednesday in the evening (e.g. because of a birthday party in the area), the truck will come only on Friday, leading to trash piling up (with all unpleasant consequences of such situation).

The aim of this work is to present an initial prototype of an inexpensive smart trash management system, consisting of a trash bin containing sensors and an application that informs about the status of individual bins. To this effect we proceed as follows. We start (in Section 2) with a brief description of the state of the art. Next, in Section 3 we outline the proposed approach, following in Section 3.1 with description of the *smart bin prototype*. Sections 3.2, 3.3 and 3.4 introduce the *database* and both the *server* and the *web application* supporting optimised trash collection. Finally, in Section 4 a simple approach to route generation has been presented.

2 State of the art

Let us start by briefly summarising the current state of the art in the area of smart garbage collection. Here, we focus on (attempts at) actual deployments rather than on research summarised in academic publications.

There are few companies, which provide waste management solutions. One of them is Slovak Sensoneo [3]. They use ultrasonic sensors to measure trash fill level (no weight measurement). Company declares that their solution reduces cost of waste collection by at least 30% and carbon emission by up to 60%. They also offer application, which informs about waste levels in all monitored bins and enables finding the nearest available empty bin. Their planning solution optimises waste collection routes based on data about fleets, depots and landfills.

The second company is Smartup Cities [4]. Their solution is based on wireless ultrasonic sensors (installed in containers). Here, Google Maps API and data from sensors allows generation of optimised collection routes. They also provide a web and mobile platform to monitor waste level. The company claims that it can save up to 50% of costs, by reducing operating hours and trucks maintenance. Solution uses artificial intelligence and machine learning algorithms to optimise collection routes and provide predictive analysis.

Next, the Ecube Labs [5] also offers solution based on ultrasonic sensors, capable of connecting to a cloud platform (named CCN). Sensors can communicate via LoRaWAN or NB-IoT. Ecube Labs trash bins facilitate trash compacting, claiming that this allows containers to hold up to 8 times more waste. Trash bins can be powered by solar energy or AC power. The CCN provides monitoring environment, smart dashboard, analytics and a control centre. Additional product named CCNx uses data analytics to optimize trash pickup routes and to facilitate various monitoring functions. Their solutions are used, among others, at the Dublin Airport, in Seoul and Washington D.C [6].

U.K. based Enevo [6] reduced costs of waste management at seven McDonalds locations across Nottingham. Their technology automatically plans daily routes for an entire collection fleet to make sure containers are collected as needed.

Finally, Ecobins [7] is a Polish company, which provides sensors for trash bins. These sensors measure filling level, current position, tilt level and temperature inside the trash bin.

All these solutions probably work in practice (our assessment is based only on the promotional material found on their websites). However, using them on a large-scale might be expensive. Additionally, presented solutions do not provide weight measurement and warning about fire in the trash bin. The latter is particularly important as burning trash may result in large-scale fires.

[3] https://sensoneo.com/
[4] https://www.smartupcities.com/
[5] https://www.ecubelabs.com/
[6] https://www.enevo.com/
[7] http://ecobins.pl/

3 Proposed approach

Our goal is to develop an inexpensive solution for waste management, based on smart trash bins and a web-based application for trash monitoring and pickup routing. Thus, we have identified the following key aspects of the sought solution:

- continuous monitoring of trash level in containers, i.e. to access detailed, current and historical data of each container separately,
- data should consist of both the weight of the trash and the trash-fill level,
- generation of routes, including only these containers which require emptying,
- estimating, which containers will be filled in a given time-frame,
- sensing smoke to generate alerts (trash bin is on fire),
- Internet connectivity,= to communicate data to the central application,
- documenting pick up actions, recognised by the container.

Let us note that we have decided to use both the weight of trash, and the level to which the container is filled. Trash weight provides extra information that might be useful when trash pickup is planned. For instance, if a given bin that should be filled with plastic is very heavy, a warning should be issued that it is likely not to be filled according to the "requirements" / expectations.

Of course, these are basic requirements, based on a minimal number of sensors that allow achieving the proposed functionality. As the price of sensors continues to decrease, further sensing capabilities (e.g. establishing main components of the waste, on the basis of detected chemical composition) that can be achieved at the same price, should be considered. Finally, a GPS sensor might be mounted to warn about potential theft (container movement).

Based on the above assumptions, we have developed an initial prototype of a smart bin, consisting of (see, also, Figure 2):

- ultrasonic sensor detecting level of trash (cost of approximately 3 EUR)
- weight sensor ($\approx$ 2.6 EUR)
- standard smoke detector ($\approx$ 5.4 EUR)
- Raspberry Pi board with WiFi module ($\approx$ 32 EUR)

Hence, the total cost of turning trash can into a *smart bin* was about 40 EUR. Recall that this is a preliminary solution. Prototype might use a microcontroller with a mounted LoRa module, thus reducing the total cost to $\approx$ 17 EUR. Furthermore, in case of mass production, price of a single set of electronics would go down considerably (above prices are based on what we have actually paid for individual components purchased from online retailers).

For the used sensors accuracy of distance measurement from containers cover to the filling level is $\approx$ 1 centimetre, while the weight data has accuracy of $\approx$ 10 grams.

Obviously, the question arises: how the proposed trash bin could be powered? Due to the high demand for electricity, the prototype is currently supplied with "stationary power". However, after replacing Raspberry Pi with a microcontroller, the prototype might become battery-ready. In this case, it will also

send information (to the central management unit) about its battery level. It might be possible to use a solar panel, but it would considerably raise the total cost. Nevertheless, in countries with a lot of sunshine, this may be a reasonable solution (to be evaluated from the financial perspective).

The developed prototype is similar in appearance to a regular municipal waste container (Figure 2). It consists of a basket and a cover. It has been designed so that electronics are not visible to passers-by. The prototype maintenance is supported by its design, which enables easy access to the electronic components, if repairing or replacing them is needed. The bin sends data, as long as it is connected to a power supply and has access to the Internet, via a WiFi network. Instead of WiFi, prototype might use LoRaWAN to reduce power consumption [8]. In the case of radio connectivity a field gateway could be developed, with wired Internet connection and a range of few kilometres.

Separately, to reduce energy consumption, the container sends data about its state only when necessary (when new waste is detected, collection event occurred, or fire has been detected). This is a standard approach for the targeted battery-power-based operation.

To support the garbage collection service (end-user), the developed solution includes a web application. Based on data generated by trash bins, it allows more effective trash collecting. Specifically, it allows user to view containers on the map and access their detailed data (current filling level and weight of waste). Moreover, each container records recognised emptying actions, available within the application. Finally, a very preliminary mechanism of selection of optimised routes for garbage collection has been developed.

In order to verify operation of the application, a separate simulator module has been created. It is a console application, designed to simulate filling of trash bins located in Warsaw.

Figure 1 depicts the system architecture. It is composed of the above mentioned components: Web *Application*, smart waste *Bin*, *Simulator*, *Server* and a *Database*. The *Application* communicates with the *Server* in order to display information generated by the *Bin*. The *Simulator* emulates existence and behaviour of multiple smart bins.

3.1 The prototype of the smart waste bin

Let us now describe, in more detail, the hardware of the prototype of the smart bin. The core element is a Raspberry Pi 3B+, a single board computer that runs Raspbian, which is a free operating system based on Debian and optimised for the Raspberry Pi hardware [9]. To the board we have connected three sensors: load cell (NA27), distance sensor (HC-SR04) and smoke detector (MQ-2).

To provide basic information about the state of the bin, two LED diodes have been used. The circuits were built on a bread board, which is a construction base for wiring and and prototyping of electronics. The prototype bin is shown in Figure 2.

Let us now describe, each of these components. The MQ-2 smoke sensor ([10])requires connection via a TTL (Transistor-Transistor Logic) converter to

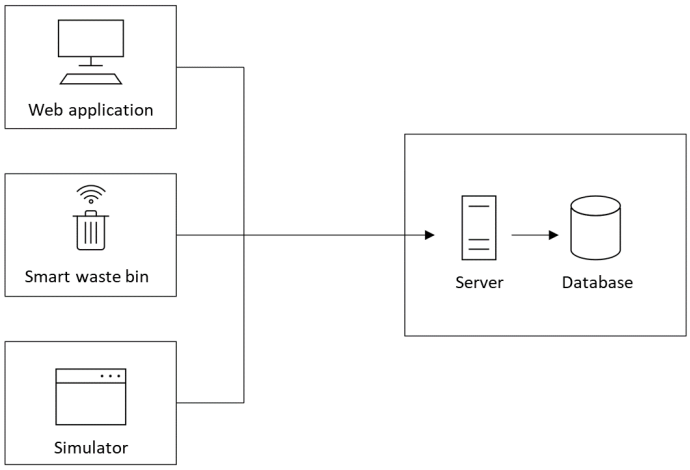

Fig. 1. System architecture diagram.

Fig. 2. Prototype of the smart waste bin.

reduce the signal voltage to acceptable for a GPIO Rapsberry Pi interface. The 5V signal is reduced to 3.3V. Then, an analog signal is converted to a digital one, using an ADC converter (MCP3008). Finally, the sensor data is read via GPIO Raspberry Pi input. The sensor data signal is a voltage signal that allows to determine concentration of gases, and thus the smoke. Here, as a gas concentration in the air increases, the resistance of the internal sensors components increases as well, signalling smoke. The HC-SR04 ultrasonic distance sensor ([11]) also requires a logic levels converter to reduce the data signal voltage to the one acceptable by the GPIO interface. This sensor provides a TRIG input and an ECHO output. Setting high state on TRIG input, for 10s, triggers the distance measurement. The output is a signal, duration of which is proportional to the measured distance.

The load cell requires connection via the HX711 operational amplifier ([12]), which is sensitive to changes in the load cells resistance. Load cell bends under applied force, changing the resistance. For communication with Raspberry Pi it uses two lines: SCLK (clock) for synchronisation and DATA (data) with digital weight values.

The described connections and the single board computer have been mounted under the containers cover, as presented in Figure 3. Two LED diodes visible on the bread board, enable reading the operating state of the prototype, without need of external monitor connection.

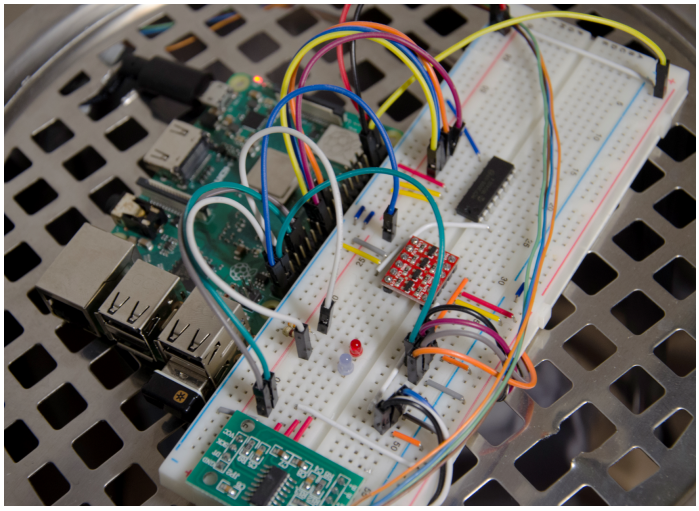

Fig. 3. Electronic device used in the prototype

The distance and smoke sensors are attached to the other side of the metal grille, providing correct readings from the containers interior and, at the same time, limiting the length of wires. The sensors are presented in Figure 4.

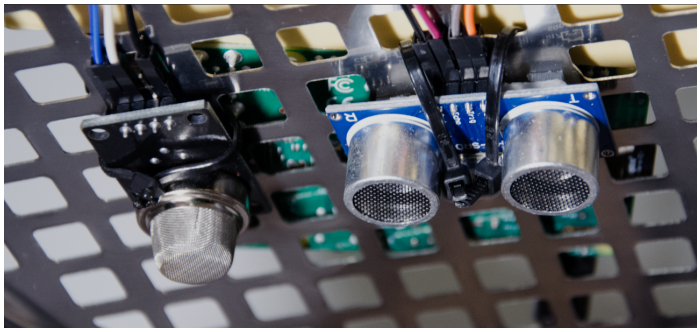

Fig. 4. Sensors; from the left: smoke sensor, distance sensor

A prototypes software, which is reading data from sensors and sending it to the *Server* was implemented in Python 3 programming language. The script calibrates sensors and checks values returned by them in the loop (which is executed every 8 seconds; parameter that can be adapted as needed). When a significant change is detected (what constitutes a significant change is a parameter adjustable for each sensor separately). New data is sent to the *Server* using HTTP protocol and JSON format messages.

The script has also an additional sleeping time of 2 seconds after new data detection. Then data is read again, to eliminate situation when falling garbage "deceives" the sensor reading. The reading is treated as correct, if both reads are the same. In this case, data is sent to the *Server*. If the second read is different, then data will be read again after 2 seconds. In the case when 5 consecutive reads (every 2 seconds) generate different values, warning is sent to the *Server* and the reading process is reset to the basic mode (repetition every 8 seconds). Obviously, in all cases, time between reads is a parameter that can be tuned on the basis of real-world observations.

The prototype starts to work as soon as it is connected to a 5V power supply. Single blinking of the red and blue LEDs indicates automatic script execution. Next, when the sensors calibration completes, user can observe the blue diode blinking. This indicates successive iterations of the main loop and readings from sensors. When new garbage is detected, the red diode blinks once. However, when garbage collection or fire are detected, the red diode blinks twice.

3.2 Database

In the system, it is necessary to store data needed by the *Application*. For this purpose, a relational database was created. Here, note that while the *Database* is described as it was placed within the *Server* (which is the way it was initially implemented), a cloud-based solution is also possible. As a matter of fact, it may be more desirable, once a real-life large-scale deployment is to be instantiated. However, this is only a technical issue that does not influence the design of the proposed system. The *Database* scheme is presented in Figure 5.

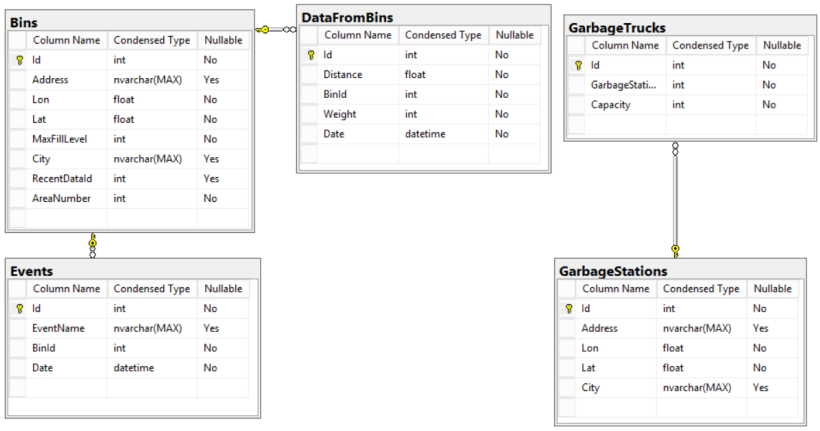

Fig. 5. Database diagram

The *Bins* table stores static data, describing states of individual trash bins. The *BinId* column is auto-incremented, unique key of each waste bin. The *Lon* and *Lat* (longitude and latitude) specify the exact location, where the trash bin can be found. The *MaxFillLevel* contains information about the maximum capacity of each garbage bin (height in centimetres). The *City* is the city where the container is located, while *Address* is the street name along with the building number nearest to the bin and *RecentDataId* is the foreign key of the most current data telemetry from the *DataFromBins* table.

The *DataFromBins* table stores the telemetry data of a given container. The *Id* is a unique, self-incremented key, the *Distance* is the distance from the bins cover to the level of garbage (in centimetres), the *BinId* is the foreign key of the garbage bin, the *Weight* is the weight of the garbage (in grams) and the *Date* is the time of receiving given data, in the standard format.

The *Events* table stores information about anomalies and events that have occurred, such as a garbage fire. Here, the *Id* is a unique, self-incremented key, the *EventName* is the name of an event (any text), the *BinId* is the foreign key of a bin, which this event has affected and *Date* is a time of receiving this data in a standard format.

The *GarbageStations* table stores information about the garbage trucks stations. The *Id* is unique, auto-incremented record key. *Lon* and *Lat* are the exact location, in which given station is located. *City* is the city where the station is located and *Address* is the name of the street with the number of a stations building as a string type.

The *GarbageTrucks* table stores garbage trucks information. The *Id* is a unique, auto-incremented key. The *GarbageStationId* is a foreign key pointing to the station, to which the given garbage truck belongs. *Capacity* is the capacity of the garbage truck, defined as the maximum number of waste bins, which can be emptied by a given truck.

Data generated, and used by the system can be divided into two types: *static data* (information about garbage containers, e.g. *Location*, information about garbage trucks, e.g. *Capacity*) and *time series data*, i.e. telemetry data generated by sensors mounted in containers.

3.3 Server

An application is running on a remote machine, hereinafter referred to as a *Server*, which is used for communication between smart waste bins and the web application. *Server* should be considered as a central computing resource and might be replaced with a cloud solution.

The *Server* performs reading and writing of data from and to the *Database*. It also performs data processing such as computing percentage filling level from the distance values and thus provides the logic of the *Application*.

The *Server* has been created according to the MVC pattern (Model-View-Controller) which is an architectural pattern used to organise the structure of applications with graphical user interfaces. MVC divides applications into three main components:

- *Model* – is a certain representation of the problem or application logic.
- *View* – describes how to display a certain part of the model within the interface.
- *Controller* – accepts input data from the user and responds to her/his actions by managing model updates and refreshing views.

The *Server* is based on the REST concept (Representational State Transfer). REST is a set of practices, which determine how network services should be implemented and is based on entities. Operations on entities are performed using HTTP queries. Entities can be modified using different query types. Types of queries are defined by verbs: Get, Post, Put and Delete. These verbs together with the URL address define what operations will be performed on a given entity.

The *Server* has been implemented in ASP.NET MVC technology using `C#` language. The application is divided on the following layers: presentation, business logic and database access. Presentation layer provides *Server* communication with clients (*Bins* and web *Application*). Each controller is assigned to one table in the *Database*. In addition, there is a controller responsible for displaying the main view. Business logic layer processes the data and prepares it for sending. In this layer each class corresponds to one of the *Database* tables and provides data processing on it. In particular, smart bin data is prepared to be displayed on the map. The *Database* access layer is responsible for updating data in the *Database* and returning the current data. Access to the *Database* is provided thanks to the EntityFramework. The *Server* also sends information about received data, so that user can observe changes (as they happen). This is ensured with the SignalR library. ISBContext interface is used to communicate with the *Database*.

3.4 The Web Application

The aim of the *Application* is to present data processed by the *Server*. *Application* retrieves data using Ajax queries. Smart *Bins* are displayed on the map as marker points in two different colours, depending on whether the *Bin* is full or not. When clicking on a given marker, in the right panel, detailed information about the selected *Bin* is displayed. The most recent weight and fill-level data are presented there as well. Moreover, user can view a table containing events that have occurred (e.g. garbage arson, emptying event). Thanks to the *SignalR* technology, data in the application is being updated without reloading the page [7]. The corresponding module receives data from the *Server* and changes data in JavaScript. In addition, the application displays the routes of collection trips using Google Maps API. We decided to use Google Maps API, because it is the market leader. Nevertheless, it should be stressed that, thanks to the modular design of our application, Google Maps can be easily replace with another mapping software, e.g. Open Street Maps.

A map with generated smart *Bins* markers is shown in Figure 6. Blue markers correspond to the bins that are not completely filled up, whereas orange markers represent the bins, which are already full. There is an information panel on the right side of the application, which appears when one of the containers on the map is selected. It contains data about selected *Bin*. At the top of the panel, basic information is displayed, such as identification number, filling level, or weight of waste. Below, there is a location of a selected *Bin* and the date of the last message received form it. At the bottom of the panel, there is a table containing the last 15 filling level updates received. Below, there is another list of the last 15 events received for the selected *Bin* along with the date and time of the event.

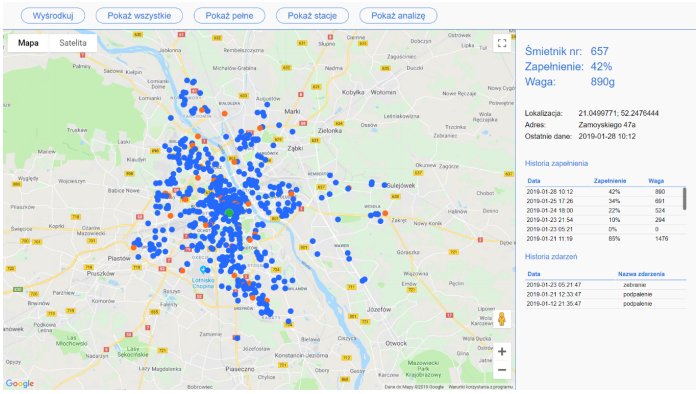

Fig. 6. A map with generated smart waste bins markers

Garbage collection routes are shown in Figure 7. This is the view of the application, when the *Show stations* option has been chosen from the menu,

followed by clicking *View routes* in the right panel. The information on how many trucks are needed is also displayed. On the map, except routes, there are also full *Bins* presented, which are to be visited during the service trips with starting points as trucks stations (purple markers).

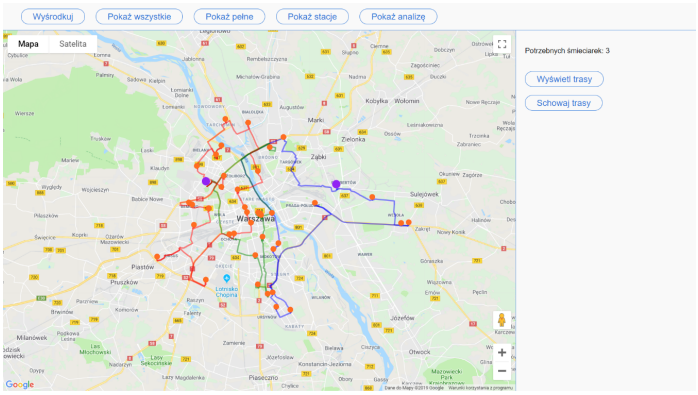

Fig. 7. Garbage collection routes

In Figure 8 a list of Bins, which are likely to fill up within the next two hours is displayed. This application view appears while choosing "Show analysis" option from the menu. On the map, yellow markers correspond to the bins shown in table in the right panel. The prediction is based on the history data and has been modelled as the average filling speed of the bins. The average was calculated as total filling level of trash collected over a period of time divided by this time. Obviously, this is a very simplistic approach and in a real-life deployment it would have to be replaced by a more sophisticated one, e.g. based on machine learning. However, since our data is only simulated, using a simple averaging is justified, as a way of illustrating how the proposed approach might work.

To ensure proper usability, the *Application* is adapted to the standard screen sizes of PC stations and portable computers. The *Application* has an intuitive and easy-to-use dashboard and works on Mozilla Firefox (from version 57.0) and Google Chrome (from version 62.0). It should work also on other browsers, but they have not been thoroughly tested.

3.5 Simulator

Due to labour-intesive process of building physical prototypes of smart bins, simulation environment (*Simulator*) has been created. *Simulator* of smart bins in Warsaw is a console application implemented in `C#` language and requires a .NET platform as a runtime environment. In order to retrieve current data about simulated containers the *Simulator* requires a *Database* connection. The supported input arguments of the program are decimal numbers from 1 up to

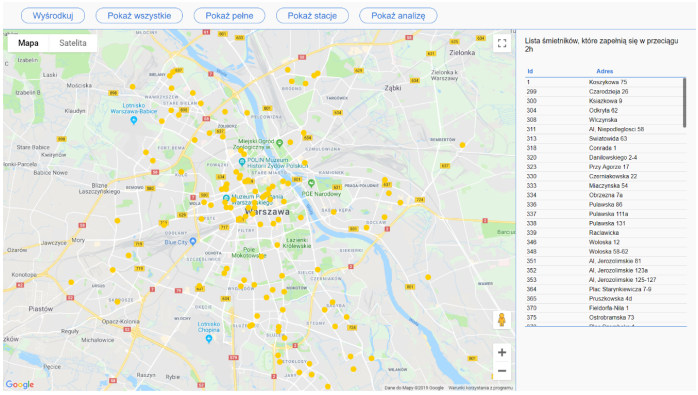

Fig. 8. Bins, which are probable to fill up within two hours

10, as a simulation speed, and parameter c for simulation of cleaning service collection trips. A parameter with a value of 1 means generating about one action per second, while parameter 10 about 6 actions per second. In the case of a numeric parameter, the application simulates throwing garbage into bins by passers-by in a loop. Pseudo-random garbage data is sent, representing readings from three sensors: distance from containers cover to a new level of filling, current weight, and messages about random events (garbage fire). Iterations of simulated actions are executed in intervals, timing of which is controlled by the input parameter. In addition to generating data of smart waste bins, the application also simulates the real-time flow. It allows continuous simulation even if simulator is closed and restarted. If simulator is run with the c parameter, the collection of garbage is simulated by sending event messages of emptying bins. From the *Server*'s point of view the *Simulator* cannot be distinguished from the physical waste container prototype.

4 Generating services collection routes

Let us now illustrate how the proposed approach can help generating trash bin collection routes. Here, let us emphazise that we are aware of a number of works that deal with optimal route generation, including optimal truck dispatch routing (see, for instance, [1–5]). However, finding the best algorithm for route generation was out of the scope of our current contribution. Nevertheless, we can already illustrate that availability of a smart trash bin, like the one described above, can reduce costs of waste collection.

A simple way to achieve this goal is to take into account only these trash bins, which filling levels exceeded a set threshold. This solution eliminates the necessity of visiting and emptying unfilled waste bins. We assume that each city has several garbage stations, in which trucks are stationed. However, since each trucks station has a limited number of trucks available, it is not sufficient to use

a trivial solution, in which the bin is served by the nearest station. Let us look into some more details of this issue.

4.1 Routes calculation scheme

To illustrate potential positive effect of the proposed approach, let us assume that only bins which are at least 80% full are considered for pick-up. Assigning containers to the garbage truck stations takes into account limited number of available trucks, as well as their capacities. In this way, bins are automatically divided into collection areas.

The proposed approach, starts from assigning bins, for which assignment to another station (not the nearest one) would have the most significant negative effect (largest distance difference). In other words, bins that are in the closest proximity of each station are allocated first. As the assignment process progresses, bins that are located "in between" stations are considered (difference in distance to the possible pickup stations is decreasing with each assigned bin). Moreover, starting form a certain moment, the capacity limitation comes to play, as stations may not be able to serve more bins. However, this means that these bins that are "left" are also "close" to the other pickup stations. After all bins are assigned, the routes are displayed in different colours and the *DirectionsRenderer* class from Google Maps API has been used. Functionality has been obtained by creating a list of full containers in JavaScript and sending a query to Google Maps API.

Figure 9 visualises the division of filled containers into collection areas. Three truck stations were instantiated for this simulation. The area corresponding the first station contains 24 bins, the next areas contains 10 and 6 bins. While the number of bins in each area differs, they are the closest to their respective collection stations.

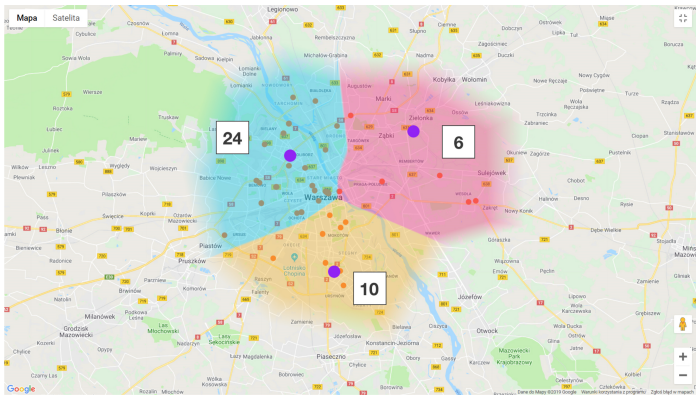

Fig. 9. Visualizations of the division of filled containers into 3 areas

In the case of two truck stations, Figure 10 presents the new division. In this example the first station is to serve 29 bins and the second one 11. This shows how the proposed algorithm works for different number of stations.

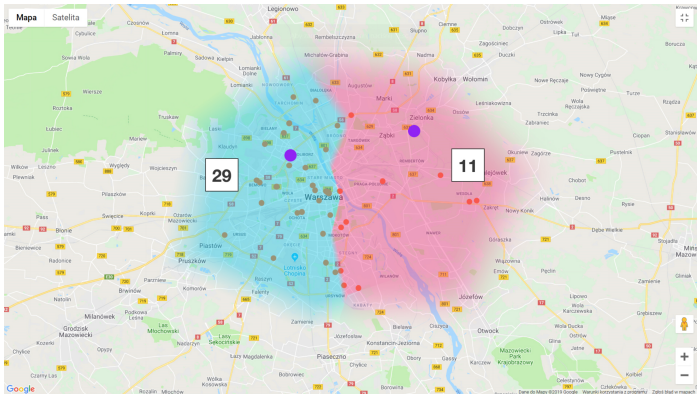

Fig. 10. Visualizations of the division of filled containers into 2 areas

Simulations with varying numbers of smart bins and truck stations (with varying total capacities of available trucks) have been performed and correctness of the developed algorithm verified.

5 Concluding remarks

The aim of our work was to show the possibilities and benefits brought by analysis of data collected from sensors placed in urban waste containers. The goal was achieved by creating a solution consisting of three elements. The first of them was an inexpensive smart bin prototype, which collects data about filling level, waste weight and is detecting potential arson. Implemented simulator enables testing the solution at the city-scale, while the web application presents opportunities of data analysis. In order for this solution to be ready for go-to-market process some improvements and extensions have to be considered. A more advanced algorithm of route calculation would lead to better optimisation of costs incurred by the city. It should be beneficial to extend the current division algorithm by adding (in the second stage) optimisation of routing in each of the areas that have been established in the area division stage (following results found in pertinent literature). Moreover, prototype requires further development of electronic components, for which miniaturisation would be necessary. Smart bin should be powered by a microcontroller and the circuits should be created based on a dedicated PCB board. In addition, communication technology should be updated so that a greater range and less power consumption could be achieved. Taking it all into consideration the second version of the prototype would be based on

LoRa32U4 II board, bringing miniaturisation, battery supply and radio connectivity. It seems beneficial to use artificial intelligence algorithms for prediction of filling as well to correlate data from sensors with external data, such as people concentration in the city.

References

1. G.B.Dantzig, J.H.Ramser, The Truck Dispatching Problem, Management Science, (1959)
2. M.L. Optimal Solution of Vehicle Routing Problems Using Minimum K-trees, Operations Research, (1994)
3. J.K. Lenstra, R.Kan, Complexity of vehicle routing and scheduling problems, Network, (1981)
4. T.K.Ralphs, L.Kopman, W.R. Pulleyblank, L.E.Trotter, On the capacitated vehicle routing problem, Mathematical programming, (2003)
5. E. Uchoa, D.Pecin. A.Pessoa. M.Poggi etc, New benchmark Instances for the Capacitated Vehicle Routing Problem, European Journal of Operational Research, (2016)
6. Ecube Labs Case Studies https://www.ecubelabs.com/references/. Last accessed 6 Apr 2019
7. Microsoft ASP.NET SignalR Homepage https://dotnet.microsoft.com/apps/aspnet/real-time. Last accessed 6 Apr 2019
8. S. L. Martinet, 3 Differences between LoRa and WiFi. Medium (2018), medium.com/@samLmartinet/lora-vs-wifi-3-questions-d9c93137fca. Accessed 6 Apr 2019
9. Raspbian Homepage, https://www.raspbian.org/. Last accessed 6 Apr 2019
10. Learning about Electronics, http://www.learningaboutelectronics.com/Articles/MQ-2-smoke-sensor-circuit-with-raspberry-pi.php. Last accessed 6 Apr 2019
11. Codelectron, http://codelectron.com/measure-distance-ultrasonic-sensor-pi-hc-sr04/. Last accessed 6 Apr 2019
12. Rasberry Pi Tutorials, https://tutorials-raspberrypi.com/digital-raspberry-pi-scale-weight-sensor-hx711/. Last accessed 6 Apr 2019