Agent-based Traffic Obstacles Information System

Kamil Ruta, Damian Rakus, Maria Ganzha, and Marcin Paprzycki

Abstract Traffic management is one of important aspects of modern cities. With the advent of, so called, *Smart Cities*, it is claimed that information technologies can be used to deliver substantial improvement in vehicle flow. As a result, among others, travel time and CO_2 generation can be reduced. In this paper, we present an agent-based traffic obstacles information and avoidance system. We discuss advantages of such a system, its key components, and technologies used during implementation. We also present sample scenarios, illustrating how the implemented system works.

1 Introduction

Nowadays, one of important issues is: how to deal with urban traffic. This question concerns all large cities around the world regardless of their location, wealth, or ethnicity. There are at least two main issues related to urban traffic. First, travel time; here, traveling in a car may take more time than traveling, the same road, using a bicycle. Second, pollution; hundreds (sometimes even tens of thousands) of cars barely moving due to traffic jams generate enormous amounts of CO_2 (and other pollutants) that remain trapped within the city are.

Kamil Ruta
Warsaw University of Technology, Warsaw, Poland

Damian Rakus
Warsaw University of Technology, Warsaw, Poland

Maria Ganzha
Warsaw University of Technology, Warsaw, Poland
Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland

Marcin Paprzycki
Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland, marcin.paprzycki@ibspan.waw.pl
Warsaw Management Academy, Warsaw, Poland

It is believed that information technologies, realizing the concept of *Smart City* [22] can, among others, improve traffic management. This claim has been partially validated by "long-distance travel" applications, such as in-car navigation connected to traffic warning systems, or Google maps, among others. However, one of their disadvantages is speed of reaction to changing road conditions. Quite often, warning about (new) traffic jam, due to an accident, is "delayed", or warning remains, about non-existent disruption. Such situations have lesser impact when considering long0-distance highway travel. Here, traffic information can be propagated across longer distances, while cars between two highway exits have "no way to escape". The situation is much more complex in urban traffic, where large number of crossing roads, and very large number of traveling cars, make a difference. On the one hand, it is much more difficult to predict where each given car will move at the next crossing. On the other, large number of possible ways of traveling from "point A to point B" add potential flexibility to any traffic management system.

Keeping this in mind, let us consider *software agents*. As can be seen in Section 2, they are very well suited when transportation schedule is negotiated and, in case of disruptions, renegotiated. The latter case directly deals with the situation when a car, moving towards a specific location, has to adapt its route due to the dynamically changing road conditions. Agents' ability to rapidly exchange information is a significant factor because it may prevent situations when user is informed about road difficulty too late and has to go back in order to bypass the obstacle.

Therefore, we have decided to investigate how software agents can be used for traffic management in a Smart City. Here, for instance, when a car detects an accident, an agent representing it reports this fact to the "central agent", which informs other interested car-agents about the obstacle. By `obstacle` we mean any "serious difficulty" on the route, such as traffic accident or traffic jam. Next, the state of the "obstacle" is to be regularly checked and updates issued if situation changes. In the proposed system, when driver specifies the destination, car agent obtains information about potential obstacles, to try to avoid them. Moreover, cars can obtain "instant information" about "new problems", so their route can be dynamically adjusted.

2 State-of-the-art in pertinent areas

Let us now briefly summarize the related state-of-the-art, starting from "driver support systems". The most advanced and popular, current solutions, are Google Maps [3, 6] and Waze [4]. Google Maps allow users to specify one or more destination points, and show the best route (and, possibly, few alternates) to reach the destination. Expected travel time is computed considering current and predicted traffic conditions. Here, Google Maps accesses users' location (calculated using GPS data and cell tower triangulation). Obstacle information is gathered from local offices and Waze databases. Note that travel time prediction is based current and historical data. Hence, quality of prediction depends on "freshness of data". When a street is being closed, or an obstacle appears on the road, there is some time delay before this fact will

be “noticed” and considered in calculations. Traffic congestion calculation requires correct input data. Hence, the system works best when large number of users travel with mobile data turned on. Note that “uneven distribution of users”, in the city, may affect the precision of predictions delivered by the system.

Waze is a navigation app with data provided by the users community called Wazers. They report traffic events such as accidents, jams, street protests. Users can also update more persistent map data, like landmarks, or new roads. Collected data, and anonymous user information, including location and speed, are stored on a central server. Using this information, Waze allows user to find the fastest route for the destination. In 2013 Waze was bought by Google and since then it also shares data with Google Maps.

Apple Maps[5] works in almost exact same way. Traffic events are reported mostly by users that are willing to share information and warn others about difficulties on the road. However, in addition to crowdsourced information, Apple is also gathering data from local databases, containing information about traffic accidents and road works. Moreover, Apple Maps track users’ average speed is based on their GPS location and can determine high traffic congestion.

Let us consider information about obstacles, available in described solutions. Use of Google Maps, or Apple Maps, is associated with storing user-location data. Here, we will immediately note that, as will be seen, only data about coordinates of obstacles is stored in the proposed system. The platform also “links the user to the streets” that are a part of the route, but after crossing a given street, this information is deleted. In addition, each city may provide its own platform instance, so none of the data is being stored globally and, in this way, the proposed system can be scaled.

One more solution, similar to Waze, is Yanosik [2], which also provides users with navigation based on, user-generated, almost-real-time traffic data. Specifically, during travel with Yanosik, user can alert other drivers about accidents, road works, police controls and speed cameras. Here, note that community-driven apps, need some initial number of users to provide useful service for drivers. Moreover, information reported by individual persons may be of “different quality”, so the more reports about an issue, the greater certainty that the data is reliable.

Another traffic navigation service is provided by INRIX. Company’s app, called ’INRIX Traffic’ [8, 15], collects anonymized data on congestion, traffic incidents, parking, and weather-related road conditions, from large number of data points, daily in over 80 countries. Data is aggregated from connected cars, mobile devices, state’s Department of Transportation, cameras and sensors on roadways, and major events expected to effect traffic. In addition, INRIX works with local authorities to digitize rules of the road for highly automated vehicles (HAVs) operating on public roads. Moreover, information gathered from HAVs can be used for infrastructure improvements. INRIX’s software updates (every 60 seconds) information about traffic conditions, such as accidents, road works and speed of traffic in different lanes. It then suggests the fastest route, taking into account all available factors. However, to achieve its wide functionality, a lot of data is being sent and processed. Setting up and maintaining infrastructure consisting of millions of cameras and sensors generates enormous costs and requires permission from various authorities.

Finally, work presented in [18] proposes another solution to traffic information distribution. Here, information is gathered in key points in a city district. Proposed system is composed of cameras, GPS locators, motion detectors, variable message road signs and mobile applications. Moreover, detectors are placed at road crossings, and help to measure traffic congestion. Available cameras can be used to detect accidents and other obstacles. Information is provided to drivers by road signs and the application. Thanks to the detectors and cameras, it is possible to detect traffic issues almost instantly. This allows to calculate the route, taking into consideration most recent road conditions. However, because of the costs and need of manual work, the system assumes installing cameras only in few "strategic" points. That may lead to incomplete data about "small roads". Here, our proposal is to use drones, so it is not necessary to install extra cameras.

As we can see, current solutions accommodate information about traffic obstacles and effects of congestion. While our system can be extended with this functionality, it was deemed to be outside of scope of our preliminary investigation.

Next, let us summarize key developments in the area of application of software agents in logistics (travel scheduling). In [17], a MAGENTA multi-agent logistic system supporting fleet scheduling is described. The technology is composed of *Ontology Management Toolkit* and *Virtual Market Engine*. The *Ontology Management Toolkit* works as a business knowledge base for agents, representing concepts and interrelationships between them. Example concepts are "Customer", "Cargo" or "CarrierCompany". The *Virtual Market Engine* allows to run agents, and monitor what they are doing. In the studied case, the system was deployed to optimise shipping schedules, as events (e.g. new cargo) occur. As a response to an event, agents start to negotiate an alternative solution, which must satisfy the specified criteria. Providing different criteria for optimisation, human scheduler is able to get various reports. Because decisions involve negotiations, human schedulers get comprehensible rationale for the proposal. Although the schedule is determined based on multiple variables, the system finds an approximation of the "ideal" solution. Here, use of semantic technologies, while interesting, seems like an overkill, in terms of system complexity and usability, outside of dedicated systems for large fleets.

Next example of transport optimisation, with the use of software agents, is presented in [16]. The Whitestein Technologies' system calculates the optimal routes for trucks, while minimizing the cost. Majority of processing, allowing handling almost-real-time changes, is done automatically. Only in special situations human involvement is needed. The solution may be implemented using agent-based approach in two different ways. Both of them concern the exchange of orders through concurrent negotiations between agents. The first approach is to assign agents to trucks. Then, high granularity and scalability is achieved, but we increase computation time and resource usage, because of a large number of exchanged messages. The second solution is to assign agents to regions, which manage groups of trucks. Here, optimisation is performed within regions (first) and between regions (next). This approach produces fewer messages, however, division into regions decreases overall system quality, because initial considerations concerning a new order will take into

account only trucks from the order's region. Overall, agent based approach has been found to be effective for large fleet transport optimization (e.g. DHL Europe).

On the basis of the above examples, it can be concluded that agent-based approach is worthy trying in the context of smart city traffic management. Agents reflect distributed nature of the problem, with large number of users, allow creation of a scalable solution, and provide easy way to exchange information.

3 System overview

Let us now outline the key aspects of the proposed approach. Let us start from main assumptions that underline system design. First, we assume that all vehicles are connected to the Internet. This assumption can be questioned. It cannot be expected that, anytime soon, all vehicles will be Internet enabled. However, it is obvious that, already today, almost all vehicles, when moving, have on board at least one Internet-connected device (driver's cell phone). Therefore, this assumption is not too far fetched. Second, there exists an infrastructure that allows car-devices to communicate with each other. Here, in the near future, 5G networks are to facilitate car-to-car communication [20]. However, let us stress that the way that communication is to be realized, is inconsequential for presented work. Finally, we assume that cars will be able to recognize and report various road obstacles. This involves both "human reporting" (now) and "car reporting" (future) capabilities.

Taking into account these general assumptions, let us now outline the design of the proposed `Traffic obstacles information system`. Here, let us note that the *user*, can be either a person driving a car or a system responsible for managing a self driving car. With this in mind, we have formulated the following requirements for the system:

- User should receive "almost real-time" information about problems on the road that the vehicle is to travel.
- User will join and listen for information about obstacles on Internet channels representing roads that are to be traversed.
- User should be notified asynchronously without explicitly requesting the data (data push model) only about obstacles that are pertinent to the current route.
- User should receive alternative route if the current is "blocked" by an obstacle.
- Users should be able to inform system about difficulties spotted on the road.

Now, let us define actions that the proposed system should be able to perform:

- Receive information about difficulties on the road (from drivers and cars with installed obstacle detection systems).
- Use drone to check and update the status of the obstacle.
- Report coordinates of all obstacles within specified area (e.g. district, city) on route that user requested.
- Propagate information about recently spotted obstacles to users on Internet channels (e.g. using JADE's Topic) named after streets.

Based on these requirements we can, in the next Section, define technologies that we have decided to use to realize the proposed architecture.

4 Technologies used

Based on comprehensive analysis of available agent platforms, we have decided to use to JAVA Agent DEvelopment Framework (JADE [9]). The system could be created using Spade [23] (multi-agent systems platform written in Python), Jason [14] (an interpreter for an extended version of AgentSpeak) or other similar platform. However, we decided to use JADE because of its popularity and integration with Java. Currently, the system is not a big programming project, but it may be easily extended to include other city traffic services. Here, managing a static-typed code repository should increase the system stability (e.g. comparing to Spade).

The developed system emulator has been written using Java 8. System GUI has been created using JavaFX and a mapJFX controls [19], which display the map. The GUI might be also written using Swing and JMapViewer [10] control, but we decided to take advantage of JavaFX [1], which is a successor to Swing [21], and allows to separate the application's view layer.

Two routing services, based on OpenStreetMap [13] data, have been used. OpenRouteService API [12] is used to calculate a route from point A to B, while avoiding obstacles. Moreover, Nominatim API [11] is used to check the name of a new street, based on car position. Although a similar service is also available in the OpenRouteService API, Nominatim reverse geocoding returns better results while searching for specific streets. Note that there exist other routing service options to choose from. Instead of OpenStreetMaps we could use Google Maps, but that would require payment, in case of a large number of queries. The most popular routing services using OSM and providing a service for avoiding areas are OpenRouteService and GraphHopper [7]. We decided to take advantage of the first one, because GraphHopper does not guarantee regular routing data updates.

5 Architecture

Let us now describe the core architecture of the proposed system. The information system runs on the JADE platform and consists of three agent types. Each car is represented as an instance of a (`CarAgent`) agent. Data about blockades is sent (by `CarAgent`) agent to the central agent (`JanosikAgent` agent), which stores and processes it. Information about a new accident / traffic jams is sent (by the `JanosikAgent`) to the `DroneAgent` agent, which is responsible for checking, whether the obstacle persists or has been removed. After user defines start point and destination of travel, `CarAgent` asks `JanosikAgent` about current blockades and calculates route that omits (minimizes effects of) present obstacles. When `CarAgent`

receives information about new obstacle on the road (message form `JanosikAgent`), it tries to adapt the route to avoid the problem. The main components of the proposed system have been depicted in Figure 1.

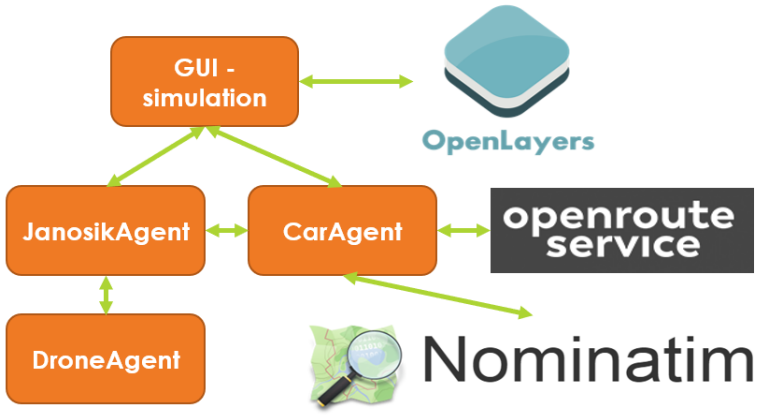


Fig. 1 System components and their interactions.

Here, the three agent types and their interactions are depicted. Note that the potential bottleneck problem, in the case of the `JanosikAgent` can be easily avoided by splitting the area of interest into smaller ones (the way that Whitestein does), each overseen by a separate instance of the `JanosikAgent` agent (demarcated as responsible for a specific area). In this case, `CarAgent` agents would connect to the `JanosikAgent` agent associated with a given area, to request information about blockades. Obviously, issues related to cars moving between areas overseen by separate `JanosikAgents` would have to be dealt with, but this is only a technical problem. The `CarAgent` uses the following services: *OpenRouteService* and *Nominatim* to calculate route that avoids known obstacles, and to request its current location information. Here, the route calculation is deliberately dedicated to the `CarAgent` agent, following the general principles of an edge-type distributed architecture (calculations should take place as soon close to the edge of the network as possible). Next, we see the `DroneAgent` agent, "managed" by the `JanosikAgent` agent, which assures that one of its instances (a physical drone) updates status of (a) specific road blockade(s). Finally, the GUI, which is based on OpenLayers, is depicted. Let us now describe each of system modules in more detail.

5.1 JanosikAgent

This is the central module, realized through one or more `JanosikAgent` agents. Its role is to manage information about road obstacles. It receives information from

CarAgent agents. It also responds to them with current information needed for trip planning. Upon receiving message about new obstacle on a given street, it broadcasts it on the JADE's Topic corresponding to that street. This is how CarAgents receive obstacle information updates. Next, it requests that the DroneAgent controls status of given obstacle. Upon disappearance of the obstacle, it informs cars, by sending message to the given Topic.

Note that the only data that the JanosikAgent agent stores, are coordinates of obstacles, which it sends to cars and to drones. Thus, it should be able to "service" large number of vehicles, and also preserve privacy of vehicle movement.

5.2 CarAgent

The CarAgent agent represents human user. It can be an independent software entity placed on "any device" (in case of older cars), or a module of the software running the autonomous vehicle. It completes multiple tasks in support of the traveller.

- Obtains information (from the JanosikAgent agent) needed to plan the route.
- Calculates the optimal route, registers to pertinent street Topics, and knows the current position of the car.
- Listens to messages concerning streets it is about to drive (street Topics).
- During travel, CarAgent agent de-registers from the Topics of streets that it has already travelled.
- When a message with a given Topic arrives, and indicates a new obstacle, the CarAgent agent asks JanosikAgent for updated information about obstacles in a given area (city) and calculates a new route to the destination (avoiding/minimising effect of) obstacles.
- Informs the JanosikAgent agent about obstacles spotted on the road.

The CarAgent agent works as an "edge client". Therefore, it receives only obstacle coordinates. Then it connects with the routing service and sends a request for a route avoiding areas around the blocked points.

As far as communication with the human is concerned, when CarAgent agent receives information about a new obstacle, it can display it to the driver and ask whether to change the route. However, in the implemented simulation module the route is adjusted automatically, to illustrated instant reaction to the appearing problem.

5.3 DroneAgent

At this stage of system development, the DroneAgent agent is a virtual entity that was introduced to illustrate proposed future functionality. It receives requests from the JanosikAgent to check status of an obstacle at given coordinates. When a drone arrives at the destination it undertakes one of the following actions:

- If the obstacle is no longer present, it informs `JanosikAgent`, and flies to check the status of the next obstacle on the list (or back to the base – depending on the level of fuel/energy).
- If the issue still exists, it flies to the next obstacle. The current obstacle is marked as still active, and is placed at the end of the list of coordinates to be checked by this (or the next) drone.

The `DroneAgent` agent manages subsequent checks of active obstacles. Currently, only a very simplistic version of this functionality has been implemented. Obviously, as the system is further developed, `DroneAgent`'s logic and responsibilities will need to be adjusted to make it more realistic.

6 Simulation and testing

Because testing of the proposed solution would require using cars and drones, in real traffic circumstances, we had to create a simulation module, to test system behaviour in different scenarios, reflecting real situations.

The simulation module displays a JavaFX GUI. For the location we have used map of Mexico City. Clicking on a map sets a marker on the chosen coordinates. Then the user can (a) "create a car" in that location, (b) set a route for the car, or (c) create a obstacle at a given location. Moreover, the GUI displays routes of remaining cars, so it is possible to observe how the system works, and how the routes are adapted on the basis of materializing obstacles.

In addition, within the GUI, the user also see the current position of the drone (in our simulation we have used only a single drone). It is also possible to adjust the simulation speed. When it is set to *1*, the cars speed is equal to the real speed on the road received from the routing service.

Let us now describe the behaviour of the system for two selected situations. While, due to space limitation we can only present two scenarios, it has to be stressed that the implemented system has been thoroughly tested, and we are certain that it works as desired (for all, however limited, functions).

6.1 Rerouting after obstacle detection

Let us, first, consider scenario where *Car 1* and *Car 0* travel to the northwest of the city, as presented in Figure 2. Initially, there are no obstacles, so routes for both cars are the same.

After some time, an accident takes place in a location "*Car 1* and *Car 0*". As we can see in Figure 3, *Car 1* does not have the obstacle in its route; hence, nothing changes. However, the accident has blocked the original route of *Car 0*, so it has received ("instant") information from the `JanosikAgent` and changed its route to avoid the blocked street.

Fig. 2 Routes of two cars without obstacles

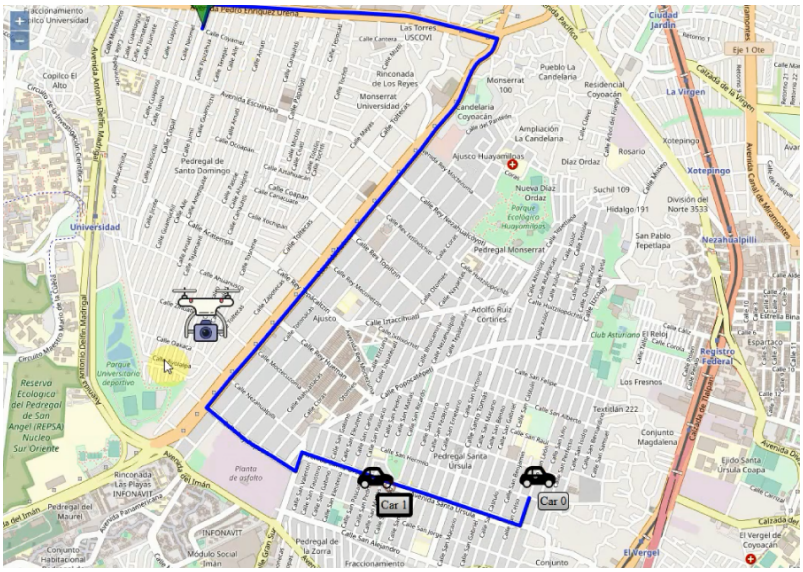
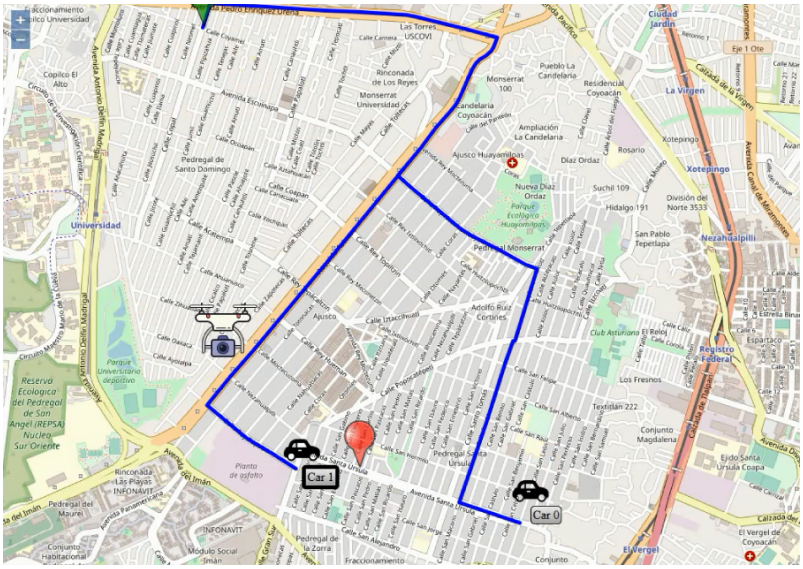


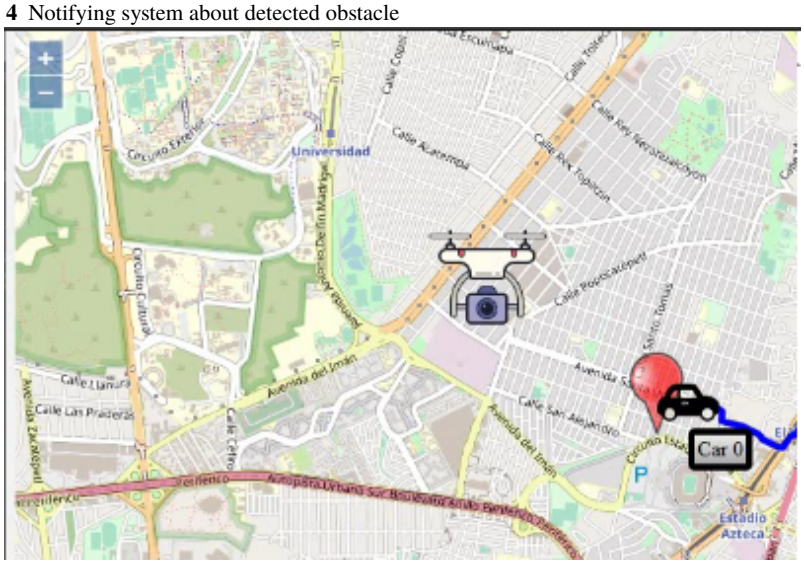
Fig. 3 Routes after a obstacle detected



6.2 New obstacle notification and use of `DroneAgent`

Let us now illustrate what happens when `JanosikAgent` is notified about a new obstacle. Figure 4 portrays situation after car detected obstacle and notified `JanosikAgent` about it.

Fig. 4 Notifying system about detected obstacle



Here, `JanosikAgent` added obstacle to its knowledge database and asked `DroneAgent` to check obstacle's status. `DroneAgent` added obstacle to the end of its queue of coordinates that need to be visited. Since there are no other obstacles currently in the system, the drone travels to the location of the detected obstacle.

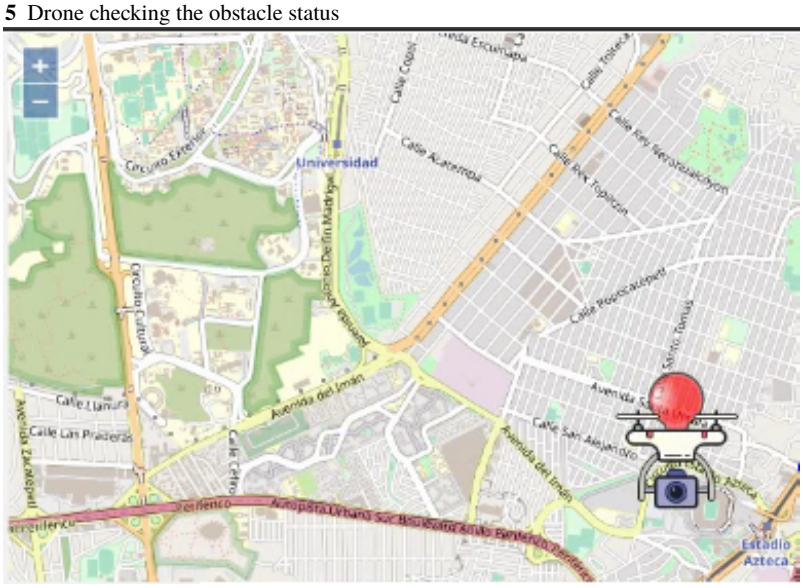
When the drone arrives to the obstacle, as depicted in Figure 5, it checks if the obstacle is still present:

- If not, then it notifies `JanosikAgent` that the road is passable now and flies to check the next point in its queue.
- If the obstacle still exists, then it enqueues point to the end, but it will not visit this point until a certain amount of time passed after last status check for this obstacle (10 second delay was used for testing purposes).

7 Concluding remarks

In this paper we have introduced an agent-based system for traffic obstacle information detection and management. The system, in its current state, can be used to help

Fig. 5 Drone checking the obstacle status



save drivers' time during unexpected travel problems. Implemented agents can be tested using simulation module, with maps of any chosen city. At the moment, it is assumed that the drone video should be analyzed by a human. However, it is obvious that, already today, it is possible to extend the system, to make it substantially more autonomous (by adding visual obstacle recognition based on machine learning). In the paper we have illustrated capabilities of the implemented system, using a developed simulation module. The system was implemented to be easily extensible. Here, extensions can involve all agents presented above. In particular, of great interest is use of a drone fleet, with realistic flight length (energy consumption) parameters. We plan to proceed with system development in the near future.

References

[1] (2016) Javafx. [online], URL `https://www.java.com/pl/download/faq/javafx.xml`, accessed: 2019-08-31
[2] (2017) Yanosik. [online], URL `https://yanosik.pl`, accessed: 2019-09-24
[3] (2019) About google maps. [online], URL `https://www.google.com/intl/pl/maps/about/`, accessed: 2019-08-31
[4] (2019) About waze. [online], URL `https://support.google.com/waze/?hl=en#topic=9022747`, accessed: 2019-08-31
[5] (2019) Apple maps. [online], URL `https://en.wikipedia.org/wiki/Apple_Maps`

[6] (2019) Google maps documentation. [online], URL https://developers.google.com/maps/documentation/, accessed: 2019-08-31
[7] (2019) Graphhopper. [online], URL https://www.graphhopper.com, accessed: 2019-08-31
[8] (2019) Inrix traffic ai. [online], URL http://inrix.com/products/ai-traffic/
[9] (2019) Jade site. [online], URL https://jade.tilab.com, accessed: 2019-07-03
[10] (2019) Jmapviewer (josm). [online], URL https://josm.openstreetmap.de/doc/index.html?org/openstreetmap/gui/jmapviewer/JMapViewer.html, accessed: 2019-08-31
[11] (2019) Nominatim. [online], URL http://nominatim.org, accessed: 2019-07-05
[12] (2019) openrouteservice. [online], URL https://openrouteservice.org, accessed: 2019-07-05
[13] (2019) Openstreetmap. [online], URL https://www.openstreetmap.org/, accessed: 2019-07-05
[14] bordini, jomifred, Media S (2019) jason - sourceforge.net. [online], URL https://sourceforge.net/projects/jason/, accessed: 2019-08-31
[15] Constine J (2016) Inrix now collects traffic data from 100m drivers, shows black friday congestion up 32.5% despite ecommerce. [online], URL https://techcrunch.com/2012/11/26/inrix/
[16] Dorer K, Calisti M (2005) An adaptive solution to dynamic transport optimization. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, ACM, pp 45–51
[17] Himoff J, Skobelev P, Wooldridge M (2005) Magenta technology: multi-agent systems for industrial logistics. pp 60–66, DOI 10.1145/1082473.1082805
[18] Kasprzyk Z, Rychlicki M, Paciorek R (2017) Proposal for implementing information systems on the road traffic conditions in warsaw in ochota district. Autobusy : technika, eksploatacja, systemy transportowe
[19] Meisch PJ (2019) mapjfx|sothawo. [online], URL https://www.sothawo.com/projects/mapjfx/, accessed: 2019-07-05
[20] Mumtaz S, Huq KMS, Ashraf MI, Rodriguez J, Monteiro V, Politis C (2015) Cognitive vehicular communication for 5g. IEEE Communications Magazine 53(7):109–117
[21] Müller B (2013) Why, where, and how javafx makes sense. [online], URL https://www.oracle.com/technetwork/articles/java/casa-1919152.html, accessed: 2019-08-31
[22] Nam T, Pardo TA (2011) Conceptualizing smart city with dimensions of technology, people, and institutions. In: Proceedings of the 12th annual international digital government research conference: digital government innovation in challenging times, ACM, pp 282–291
[23] Palanca J, Foundation PS (2019) spade - pypi. [online], URL https://pypi.org/project/spade/, accessed: 2019-08-31