AR-ENABLED MOBILE APPS FOR ROBOT COORDINATION

Nenad Petrović[1] Milorad Tošić[2] Valentina Nejković[3]

Abstract: Coordinated behavior of multiple devices has potential for more efficient goal achievement, but is not exploited in the existing AR robot companion smartphone applications. In this paper, AR-enabled case study applications enabling coordination scenarios of ROS-based TurtleBots are presented, aiming COVID-19 prevention. The applications are implemented using AppSheet and AR.js. Four aspects relevant to robot coordination are covered: 1) robotic device management; 2) robot control; 3) sensor data monitoring; and 4) coordination scenario design. According to the research outcome, the proposed approach dramatically speeds up the development of AR-enabled robotic mobile applications.

Key words: AppSheet, augmented reality (AR), ROS.

1. INTRODUCTION

The evolution towards Industry 4.0 paradigm has increased the adoption of augmented reality (AR), cyberphysical systems and human-machine communication [1]. At the same time, robots are becoming part of our everyday life, in many other areas outside the traditional industrial environment, from the smart home and entertainment to rehabilitation, social robotics and multi-agent systems. In this context, augmented reality, together with smartphone companion applications plays the role of a new medium for interaction and information transfer between the human and autonomous systems making the process of Human-Robot Interaction (HRI) more efficient [1], [2].

In [3], an AR-powered application aiming teaching the operations with industrial robots in production lines is presented. The robots appear as 3D models over a marker and workers can experiment with their capabilities (such as rotation and movement) without affecting the production line itself. On the other side, the work in [2] shows a multipurpose robot car based on Arduino controlled using the smartphone over WiFi and sending images of the detacted obstacles. Finally, in [4], a mobile AR app for vehcile-like robot motion planning is introduced.

Despite the fact that coordinated behavior of multiple (robotic) devices can lead to a more efficient goal achievement [5], [6], it is not exploited in most of the available AR-enabled mobile companion applications.

In this paper, we aim the usage of AR-enabled mobile applications in order to make possible the coordination scenarios in multi-robot environments tackling the COVID-19 and its spread prevention. As an outcome, we present a case study implementation using AppSheet and AR.js targeting the ROS-based robotic platform [7], built upon our previous work on robot coordination [5].

2. BACKGROUND AND RELATED WORK

2.1. TurtleBot3

TurtleBot refers to a ROS-based robot platform, introduced in 2010 [8], [9]. It covers a range of small, low-cost, battery-powered, vehicle-like ground mobile robot kits relying on open-source software. There are three main versions of the TurtleBot model. In this paper, the focus is on Tutlebot3 series (2017) which fully supports SLAM (Simultaneous Localization and Mapping). There are three TurtleBot3 models: Waffle, Waffle Pi and Burger. The later two are based on Raspberry Pi and equipped with 360-degree LiDAR. Unlike Burger, Waffle also has a camera, making it possible to leverage computer vision techniques, as well. Both of them are controlled by the OpenCR [10] board, which is designed for the ROS embedded systems relying on completely open-source hardware and

[1]PhD Student, University of Niš, Faculty of Electronic Engineering, email: nenad.petrovic@elfak.ni.ac.rs
[2]PhD, University of Niš, Faculty of Electronic Engineering, email: milorad.tosic@elfak.ni.ac.rs
[3]PhD, University of Niš, Faculty of Electronic Engineering, email: valentina.nejkovic@elfak.ni.ac.rs

software, integrating an ARM-based microcontroller and sensors. The main use cases of such robot platforms are education and prototyping.

2.2. The Robot Operating System

The Robot Operating System (ROS) is a collection of open-source software packages, which aims to enable convenient development of robotic application without going into hardware-specific details [7], [9]. It is sometimes referred to as a "meta-operating system", as it really does provide the capabilities of a full operating system (OS), but still relies on the host computer's OS in order to be executed. The main purpose of the ROS is to provide a communication between the user, the host computer's OS, and equipment which is external from the perspective of the host. The supported equipment covers various sensors, cameras and entire robotic devices. Its hardware abstraction gives the ability to control a robot without knowing all of the details related to the robot's implementation. For instance, if we want to move the robot, a ROS command is issued or script written in Python or C++ by the robot designers, which further calls various control programs that enable the motion of robot's wheels. Moreover, the ROS gives the ability to design and simulate our own robots using tools such as Gazebo [11].

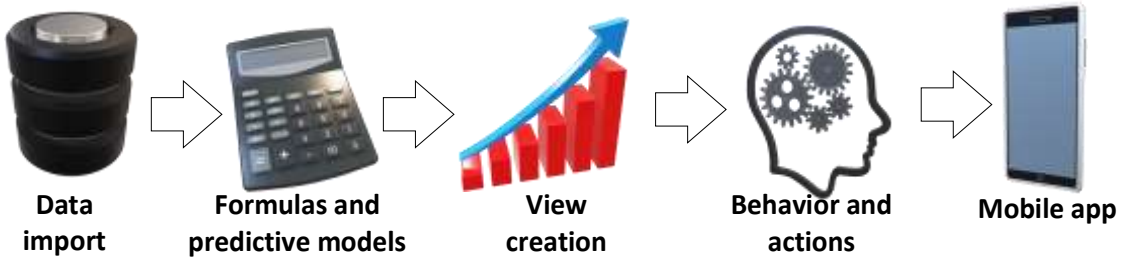
The command issuing and sensor data monitoring in the ROS are based on a publish-subscribe model. There is a list of pre-defined ROS topics that can be seen by typing the *rostopic list* command [9]. A typical example of a topic used in this paper is *Twist*, which is used for the robot movement. It has the following form:

rostopic pub /tb3_0/cmd_vel geometry_msgs/Twist -r 10 -- '[-0.14,0.0,0.0]' '[0.0, 0.0, 0.0]' (1)

In order to move the robot, it is necessary to publish a message with the following arguments: 1) number of commands per second (*-r number*); 2) linear coordinates; and 3) angular coordinates. For the command given in eq. 1, the robot moves backwards for 0.14 steps along x-axis, while it is issued 10 times per second.

2.3. AppSheet

AppSheet[2] is an online platform acquired by Google in 2020, which aims easy creation and distribution of multiplatform mobile applications starting from cloud data sources, such as spreadsheets and databases without almost any coding. It automatically generates the views that can be shown within the application based on the imported data. However, it is possible to customize them by showing or hiding particular columns and writing formulas for data processing and aggregation. AppSheet includes a high-level declarative language for the formula and application behavior definition, relying on a set of pre-defined actions that can be triggered in a particular context, such as transition between views, sending an HTTP request to an external service, an e-mail or SMS. It is free for prototyping and personal use (up to 10 users), while a monthly fee is charged in case of commercial applications. Moreover, it offers advanced machine learning and AI features such as value prediction, optical character recognition (OCR) and voice commands. However, an active Internet connection and a mobile client application for the corresponding operating system are required in order to access AppSheet applications and their features, as they are deployed in the cloud. In Figure 1, an overview of AppSheet-based mobile application development is given.



[2] https://www.appsheet.com/

Figure 1 – AppSheet mobile app development workflow

AppSheet has been approved as an effective solution within various areas so far. In [12] and [13], it was used for supportive mobile apps during the COVID-19 pandemic, considering several case studies: efficient city resource management, indoor safety monitoring, volunteer help, quick assessment, contact tracing and test scheduling. Furthermore, AppSheet together with Apps Script was used in [14] for a personalized fitness trainer application.

2.4. AR.js

AR.js is an open-source, lightweight JavaScript library for the development of augmented reality applications run in the web browser [15]. It offers a wide compatibility, covering traditional PCs, tablets and variouos models of smartphones altogether. On the other side, it also shows solid performance on smartphones without high hardware demands, even on devices that are more than five years old [15]. Moreover, the development of AR applications relying on this library is intuitive and straightforward – it is possible to implement usable AR apps in just several lines of HTML and JavaScript code [16]. The main features of this library include a pre-defined set of markers (numeric barcodes and QR), location and position-based input, 3D object loading and animation. In [16], AR.js was approved as an effective solution for various types of AR-based user interfaces, such as a music loop sampler for the live stage performance. Moreover, in [17], it was used together with AppSheet for the interface enabling convenient smart home remote household device control and energy management.

3. CASE STUDY APPLICATIONS

The main purpose of the presented case study is to enable a convenient mobile robot coordination in scenarios related to area surveillance using a smartphone, with the focus on use cases related to the COVID-19 prevention. It is known the COVID-19 safety guidelines and policies still have to be respected, including mask wearing, social distancing and limitaiton of public gatherings and the number of people involved. The application is implemented using both AppSheet and AR.js. When it comes to the code generation mechanisms responsible for robotic device coordination, it relies on our previous work – the SCOR framework [5], which leverages ontologies and semantic knowledge representation. Four aspects relevant to the robot coordination are covered: 1) robotic device management; 2) robot control; 3) sensor data monitoring; and 4) coordination scenario design. In Figure 2, an overview of the generalized case study applications architecture is given.

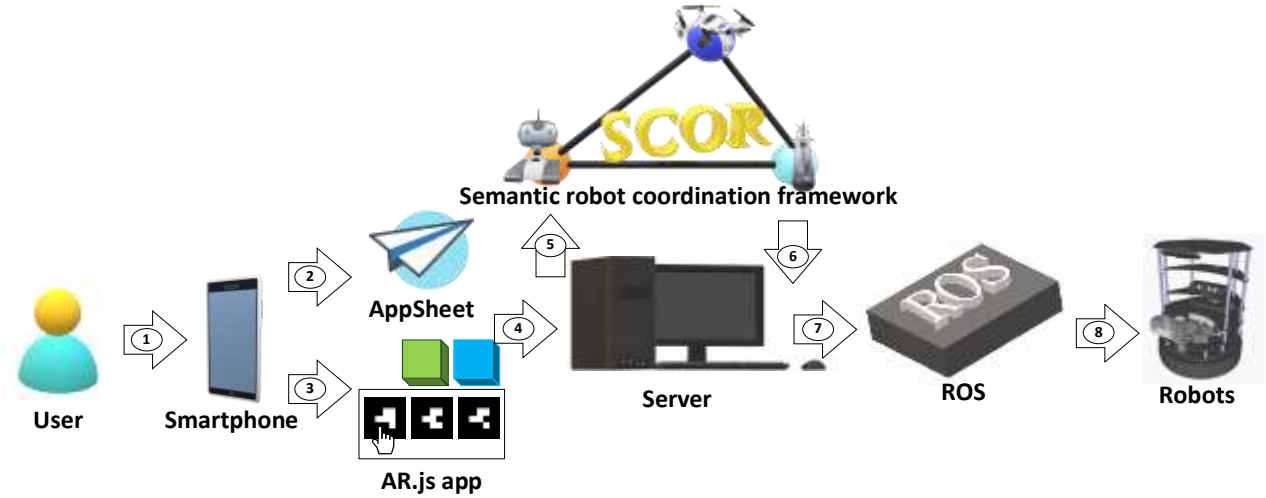


Figure 2 – Robot coordination mobile apps architecture overview: 1-User action; 2-Google Sheets data; 3-Camera input; 4-HTTP requests; 5-Semantic queries; 6-Query results; 7-ROS commands; 8-Low-level device commands

There are two possible user input methods: either using AppSheet app or augmented reality user interfaces written in AR.js. As a response to the user action, HTTP requests are sent to the server, which leverages the SCOR framework for necessary device and sensor data semantic annotations together with robot coordination mechanisms. Moreover, there could be additional scripts (Node.js, Python) on the server that are used to listen to the requests and respond accordingly, in the current context by ROS command execution or data analysis algorithms, which extract knowledge from sensor data.

3.1. Robotic device management

This part of the application serves to provide information about the available devices and their capabilities. The information shown is retrieved from ontologies related to the device capabilities within the SCOR framework [5]. It consists of the follwoing three screens: 1) Robots (Figure 3a) – showing the list of available robots; 2) Sensors (Figure 3b) – the list of all sensors attached to the available robots and their properties; 3) Detailed info (Figure 3c) – the table showing which sensors are offered by each of the robots (such as LiDAR, camera) in the form of *robot->sensor pairs*.

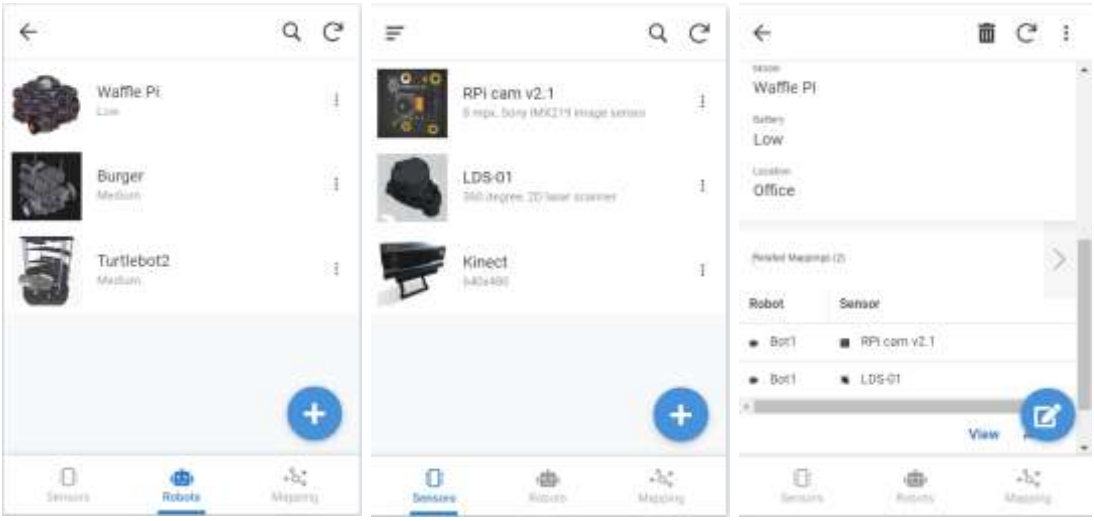


Figure 3 – Robotic device management app screenshots: a) Robots; b) Sensors; c) Detailed info

For each of the robots, the following information is shown after selecting it from the list: image, id, model, battery status, location and list of available sensors (Figure 3c). Furthermore, for each of the sensors – image, id and description are visible, together with all the robots equipped with this kind of sensor. Moreover, using this application, both the robots and their sensors can be switched to standby mode, and turned off or on remotely, which can be quite useful when battery saving is critical.

Finally, the application also includes the AR interface for a convenient robot info visualization, as shown in Figure 3. To each robot, a distinct barcode marker is printed out and assigned. When the smartphone camera is pointed to the barcode, a 3D cube with basic information (model name, battery status) about the targeted robot appears. Furthermore, putting the hand over the cube redirects the user to the AppSheet page with detailed robot information, as shown in Figure 4. For AR app generation starting from SCOR ontologies, our framework for automated AR.js code generation [16] was used.

Figure 4 – AR interface for quick robot info visualization

3.2. Robot control AR interface

This augmented reality web application is written in AR.js and provides the ability of synchronized two-robot control using hand gestures, as shown in Figure 5. For this purpose, three markers showing 3D cubes with different colors and meanings are used: 1 – moves robots backwards (red) 2 – moves robots forward (green) 3 – rotates robots (blue). A demo video of this application is available on the YouTube link[3].

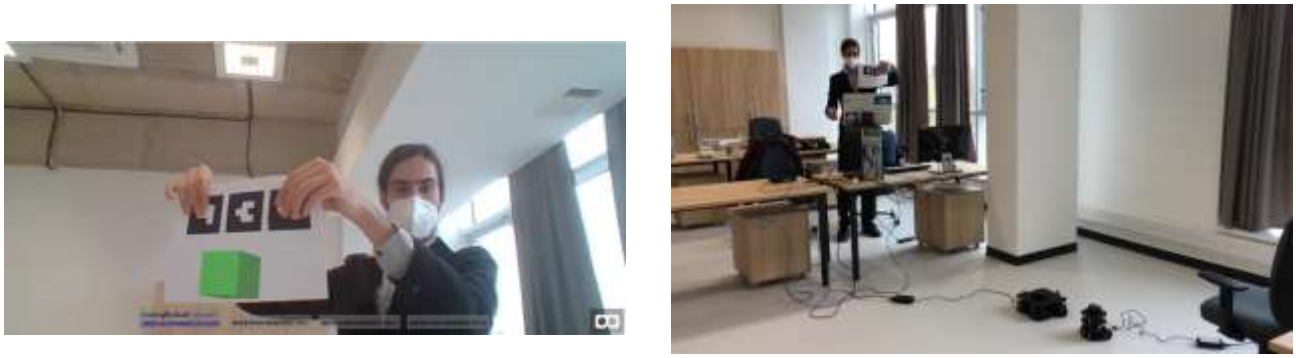


Figure 5 – AR.js mobile application interface for synchronized control of two robots

When the user puts their hand over the marker, AR.js mobile app sends an HTTP request to Node.js script on the server side, which then executes the corresponding ROS command for the robot movement. In Figure 6 the main part of this script is shown.

```
const server = http.createServer((req, res) => {
...
const query = url.parse(req.url, true);
const path=query.path;
const command=querystring.parse(path);
const code=command["/command"];

   if(code==1)
   {

            exec.exec("timeout 2s rostopic pub /tb3_0/cmd_vel geometry_msgs/
Twist -r 10 -- '[-0.14,0.0,0.0]' '[0.0, 0.0, 0.0]'", (error, stdout, stderr)
=> {
                  if (error) {
                        console.log('error: ${error.message}');
                        return;
                  }
                  if (stderr) {
                        console.log('stderr: ${stderr}');
                        return;
                  }
                  console.log('stdout: ${stdout}');
            });

      }

   ...
   if(code==3)
   ...
}).listen(port, ip);
```

[3] https://youtu.be/S6dsYiTXguo

Figure 6 – An excerpt from Node.js server-side script calling ROS commands

Moreover, there is a Python script relying on rospy[4] (a client Python library for ROS) that listens to the commands received by one of the robots and forwards them to the other. In Figure 7, the code of this Python script for the robot movement command forwarding is given.

```
import rospy
from geometry_msgs.msg import Twist

def callback(data):
    robot = rospy.Publisher('/tb2_0/cmd_vel',Twist,queue_size=10)
    command = Twist()
    command = data
    robot.publish(command)

def listener():
    rospy.init_node('roscopy', anonymous=True)
    rospy.Subscriber('/tb3_0/cmd_vel', Twist, callback)
    rospy.spin()

if __name__ == '__main__':
   try:
      listener()
   except rospy.ROSInterruptException: pass
```

Figure 7 – Python rospy script for command forwarding

3.3. Sensor data monitoring

On the other side, the application aims to visualize the sensor data coming from robotic devices or the results of their analysis. The proposed solution considers LiDAR data and knowledge extracted from the camera images relying on the computer vision algorithms. In Figure 8, the screenshots of this application are given.

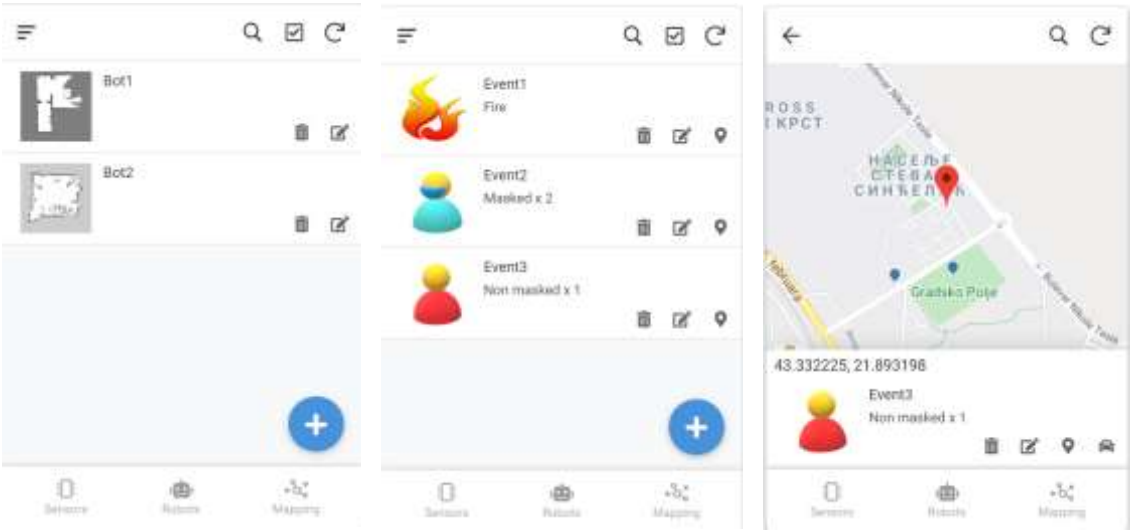


Figure 8 – Sensor data monitoring app: a) LiDAR map view; b) Event view; c) Map of events

The view shown in Figure 8a provides the ability to see the current LiDAR map snapshot taken by the selected robotic device. The implementation was done relying on two JavaScript libraries: 1)

[4] http://wiki.ros.org/rospy

roslibjs[5] – offers interface to ROS from web browser; 2) ros2djs[6] – 2D visualization library for ROS. The main part of the LiDAR map saving in the PNG format using Node.js is given in Figure 9.

Moreover, Figure 8b shows a list of events detected by the computer vision algorithms – such as fire, presence of people, their number, whether they wear masks or not. Finally, the map view from Figure 8c displays the places on the map where these events have been detected. When it comes to event detection capability, it relies on the computer vision-based object detection implemented in Python, using OpenCV library, built upon the works from [18] and [19]. The images are streamed using ROS from the robot to the edge server, where they are analyzed. Therefore, these features are only available for Waffle Pi robots, as they are equipped with the camera.

```
window.showMap=()=>{
  var viewer = new ROS2D.Viewer({
    divID : 'map',
    width : variables.MAP_WIDTH,
    height : variables.MAP_HEIGHT
  });

  var gridClient = new ROS2D.OccupancyGridClient({
    ros : variables.ROS,
    rootObject : viewer.scene,
    continuous: true
  });

  // update map
  gridClient.on('change', ()=> {
    viewer.scaleToDimensions(gridClient.currentGrid.width,
                             gridClient.currentGrid.height);
    viewer.shift(gridClient.currentGrid.pose.position.x,
                 gridClient.currentGrid.pose.position.y);
  });
 }

 window.saveMap=()=>{
  const a = document.createElement("a");
  document.body.appendChild(a);
  window.scrollTo(0,0);
  html2canvas(document.getElementById("map")).then(canvas => {

      a.href = canvas.toDataURL();
      a.download = "map.png";
      a.click();
      document.body.removeChild(a);
  });

}
```

Figure 9 – An excerpt from Node.js server-side script saving LiDAR map snapshot

3.4. Coordination scenario design

Finally, this application provides the user interface for the coordination scenario design. It is based on AppSheet and gives the ability to define rules in the form:

if (event:e occurs at location:l) then send(robot:r with device:d to location:l) (2)

[5] http://wiki.ros.org/roslibjs
[6] https://github.com/RobotWebTools/ros2djs

Therefore, it is possible to select which robot, equipped with a specific device (sensor or actuator) will be sent to the location where some specific type of event has occurred. For example, if a group of people without masks is detected, then, another robot carrying masks can be sent to the location, so the persons can take them. This application also offers the map view of the coordination rules. The coordination scenario consists of one or many coordination rules. In Figure 10, the screenshots of this application are given.

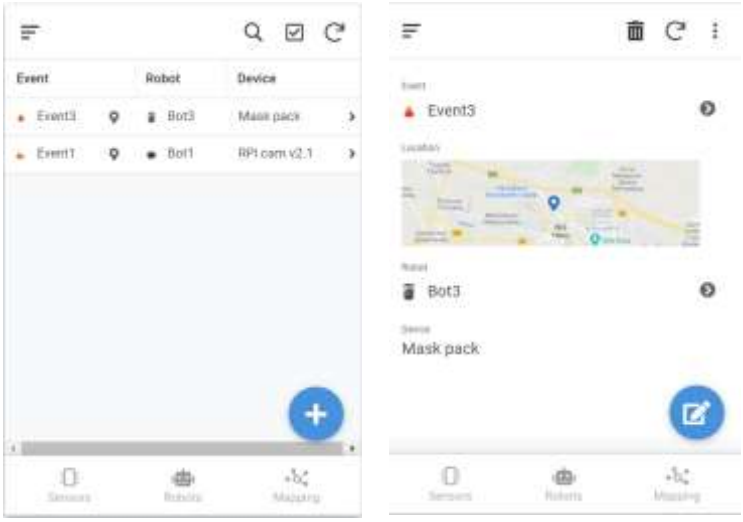


Figure 10 – Coordination scenario design app: a) Rule list; b) Detailed view

4. CONCLUSIONS

In this paper, the implementation of several AR-enabled mobile applications aiming robot coordination with the goal of the COVID-19 spread prevention based on AppSheet and AR.js is presented. The adoption of these two technologies significantly reduces the time necessary for the development of multiplatform AR-enabled mobile applications, which is from 10 to almost 30 times shorter, based on the results and estimations from [16] and [17].

5. ACKNOWLEDGMENT

This work has been supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia.

6. REFERENCES

[1] Makhataeva, Z., Varol, H. A. (2020). *Augmented Reality for Robotics: A Review*. Robotics 9, 21, 1-28, 2020. https://doi.org/10.3390/robotics9020021

[2] Teterbay, B., Bhati, A., Srivastava, A., Deshpande, A. (2020). *Smartphone Controlled Multipurpose Robot Car*. International Journal of Engineering Research & Technology (IJERT), Vol. 9 Issue 05, May2020,485-488. http://dx.doi.org/10.17577/IJERTV9IS050382

[3] Zhang, F., Yin Lai, C., Simic, M., Ding, S. (2020). *Augmented reality in robot programming*. 24th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, Procedia Computer Science 176, pp. 1221–1230.

[4] Fuste, A. et al. (2020). *Kinetic AR: Robotic Motion Planning and Programming using Augmented Reality Interfaces*. HRI '20: Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction, pp. 641-641.https://doi.org/10.1145/3371382.3378394

[5] Nejkovic, V., Petrovic, N., Tosic, M., Milosevic, N. (2020). *Semantic approach to RIoT autonomous robots mission coordination*. Robotics and Autonomous Systems, Volume 126, April 2020, 103438,1-19, 2020. https://doi.org/10.1016/j.robot.2020.103438

[6] Williams, A. E., Petrovic, N. (2020). *The Role of General Collective Intelligence in Overcoming the Barriers to Achieving an Internet of Things*. ResearchGate preprint,1-9.https://doi.org/10.13140/RG.2.2.33502.66885

[7] Fairchild, C., Harman, T. L. (2017).*ROS Robotics By Example*, 2nd edition, Packt Publishing.

[8] *Turtlebot*(2020, January 28). Retreived from: https://www.turtlebot.com/

[9] Pyo, Y. S., Cho, H. C., Jung, R. W., Lim, T. H. (2017). *ROS Robot Programming*. ROBOTIS Co. LTD.

[10] *OpenCR 1.0 ROBOTIS e-manual* (2020, January 28). Retrieved from: https://emanual.robotis.com/docs/en/parts/controller/opencr10/

[11] *Gazebo*(2020, January 30). Retrieved from: http://gazebosim.org/

[12] Petrović, N., Radenković, M., Nejković, V. (2020). *Data-Driven Mobile Applications Based on AppSheet as Support in COVID-19 Crisis*. IcETRAN 2020, pp. 609-614.

[13] Petrović, N., Dimovski, V., Peterlin, J., Meško, M., Roblek, V. (2021).*Data-Driven Solutions in Smart Cities: The Caseof Covid-19 Apps*. WWW '21 Companion - 7th WebAndTheCity – Web Intelligence and Resilience in Smart Cities, Ljubljana, Slovenia. https://doi.org/10.1145/3442442.3453469

[14] Radenković, M., Petrović, N., Nejković, V. (2020). *Razvoj podacima-vođenih multiplatformskih mobilnih aplikacija korišćenjem AppSheet i Apps Script na osnovu Google Sheet tabela u oblasti zdravlja*. IEEESTEC – 13th Student Projects Conference, pp. 1-4

[15] Etienne, J. (2020, January 29). *Creating Augmented Reality with AR.js and A-Frame*. Retrieved from:https://aframe.io/blog/arjs/

[16] Đorđević, L., Petrović, N., Tošić, M. (2020). *An Ontology-based Framework for Automated Code Generation of web AR Application*. Telfor Journal 12(1),67-72. https://doi.org/10.5937/telfor2001067Q

[17] Petrović, N., Roblek, V., Nejković, V. (2020). *Mobile Applications and Services for Next-Generation Energy Management in Smart Cities*. eNergetics 2020, pp. 1-9.

[18] Petrovic, N. (2018). *Surveillance System Based on Semantic Video and Audio Annotation Leveraging the Computing Power within the Edge*. XIV International SAUM 2018, pp. 281-284.

[19] Petrović, N., Kocić, Đ. (2020). *IoT-based System for COVID-19 Indoor Safety Monitoring*. Proceedings of IcETRAN 2020, pp. 603-608.