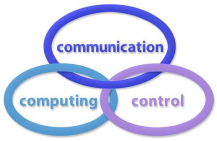
INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
Online ISSN 1841-9844, ISSN-L 1841-9836, Volume: 19, Issue: 1, Month: February, Year: 2024
Article Number: 6030, https://doi.org/10.15837/ijccc.2024.1.6030



CCC Publications

An Indoor Localization System for Automotive Driving Competitions

N. A. Kilyen, R. F. Lemnariu, I. Muntean, G. D. Mois

Nandor Alpar Kilyen
1. Bosch Engineering Center Cluj
No. 30-34 Constaţa Street
Cluj-Napoca 400158, Romania
2. Department of Automation
Faculty of Automation and Computer Science
Technical University of Cluj-Napoca, Cluj-Napoca, Romania
Nandor.Kilyen2@ro.bosch.com

Rares Florin Lemnariu
1. Bosch Engineering Center Cluj
No. 30-34 Constaţa Street
Cluj-Napoca 400158, Romania
2. Department of Mechatronics
Faculty of Automotive, Mechatronics and Mechanical Engineering
Technical University of Cluj-Napoca, Cluj-Napoca, Romania
Rares.Lemnariu@ro.bosch.com

Ionut Muntean*
1. Bosch Engineering Center Cluj
No. 30-34 Constaţa Street
Cluj-Napoca 400158, Romania
2. Department of Automation
Faculty of Automation and Computer Science
Technical University of Cluj-Napoca, Cluj-Napoca, Romania
*Corresponding author: Ionut.Muntean@ro.bosch.com

George Dan Mois
Department of Automation
Faculty of Automation and Computer Science
Technical University of Cluj-Napoca, Cluj-Napoca, Romania
George.Mois@aut.utcluj.ro

Abstract

Localization, in both indoor and outdoor settings, represents a problem that has received increased attention lately. This paper presents the development, testing, and validation of an indoor localization system for 1/10 scale vehicles based on the Robot Operating System (ROS) and ArUco marker detection. It has a distributed architecture, consisting of tens of Raspberry Pi (RPi) single-board computers running ROS nodes and fitted with cameras, that monitor a certain area of the

14x14 m plane. The developed system has been successfully used in three editions of Bosch Future Mobility Challenge, an international student competition, where the participants are required to implement autonomous driving functionalities in an environment resembling a real-life city on small-scale automated vehicles.

Keywords: embedded systems, image processing, localization, positioning system.

1 Introduction

The interest in localization of autonomous mobile systems is a highly active research topic that has been facilitated by the developments in advanced mobile and communication technologies [12, 17]. Computing the position and the orientation of an object is a key technology that can bring us a step closer to autonomous vehicles, enabling them to manage in an autonomous way traffic situations and signifficantly reduce the number of accidents [7, 10]. Communication is also concerned with the placement of the used equipment, the localization problem being a requirement also for wireless sensor networks (WSNs) and vehicular ad hoc networks (VANETs), where position-based routing can improve packet delivery ratios, or communication delays and overheads [13, 18]. The application domain of such systems is even wider, being used in environmental monitoring applications, precision farming, inventory management, road traffic monitoring, health monitoring, and others, besides navigation [8].

While the finding of the position of objects in open field is based on the well-established Global navigation satellite systems (GNSS) tool, a wide range approaches were proposed for places where satellite signals are blocked, such as indoor environments. Indoor Positioning Systems (IPS) deal with locating moving devices indoors and are based on local infrastructure [2]. Some of the used approaches are based on Wi-Fi [16] or Bluetooth [5] technologies, on the emerging Ultra WideBand technology [1], on image processing [4], on the angle of arrival (AoA), on the time of arrival (ToA), or on the time difference of arrival (TDoA) of signals [2], and many others. The increasing amount of smart devices, belonging to the Internet of Things (IoT) vision, lead to techniques that make use of crowdsourced data, that is obtained from many different entities and that is later fused [3]. The existence of such a rich palette of systems for indoor positioning shows that there is no standard approach, each one of the developed methods having its own strengths and drawbacks, that have to be analyzed in the case of a particular application.

The main motivation for developing the localization system presented in this paper lies in the requirements for Bosch Future Mobility Challenge (BFMC), an international autonomous driving and connectivity competition on 1/10 scale vehicles, involving master and bachelor students [9]. The challenge was created to provide participants with a platform for interacting with the autonomous driving domain and to develop and test algorithms in an environment as close as possible to a real one.

Global navigation satellite systems are a common tool used in current real-life mobility solutions and a replica of such a system was also needed for BFMC. In this context, the purpose of the localization system presented in this paper is the acquisition of the position and orientation of the cars on the track, in a manner similar to the one provided by a real-life GNSS (Global Navigation Satellite System). The area that has to be monitored, the track where the autonomous cars navigate, is placed indoors and covers a square area with a side of 14 meters. The chosen solution is represented by a system with a distributed architecture, consisting on tens of Raspberry Pi single-board computers running ROS nodes and fitted with cameras, that continuously monitor the target area for detecting the model cars and computing their position relative to one corner of the square, the origin.

The method is based on the processing of the images captured by the Raspberry Pi cameras and the detecting of ArUco (minimal library for Augmented Reality applications) tags attached to the tracked cars [15]. Determining the displacement of the tags mounted on the cars in correspondence to specific markers placed at previously known locations (matrix points 1 meter apart) leads to the computation of the actual position of the car on the monitored area. Fiducial markers offer the advantage of fast and facile detection with the use of open source libraries, such as OpenCV (Open Source Computer Vision Library) [20]. The system we developed makes use of the free ArUco minimal library for Augmented Reality that is based exclusively on OpenCV [21]. An architecture and implementation that rely on open source code and on image processing were chosen because a reliable and accurate system that

can also easily be exploited and replicated was desired. Solutions that are based on RSSI (Received Signal Strength Indicator) or on signal arrival times require the use of more specialized hardware and are prone to accuracy degradation due to radio frequency interference and NLOS (non-line-of-sight) situations [14].

The Robot Operating System platform runs on all the RPi devices in the system, managing all the actions performed by each one of them (image capture and image processing, calibration, car markers' position calculation, and data communication) [6]. ROS is an open-source framework that supports the rapid development of robot applications, providing services such as hardware abstraction, low-level device control, and message transmission between processes using the publisher-subscriber model [11]. Besides the advantage of accelerating development, ROS was chosen due to the large community and knowledge base. This allows both the organizers of the competition and the participants to develop efficient modules that make use of the simulated-GPS system.

The main contributions of the paper are as follows:

- the development of a real-time localization system based on ArUco markers;
- extensive testing of the developed system during preparations for an autonomous driving competition, BFMC;
- validation of the operation of the proposed system during the BFMC autonomous driving competition.

The rest of the paper is structured as follows. In Section II we present the overall architecture of the implemented camera-based localization system, and detail the used hardware components and the developed software modules. In the following section, we describe the testing and validation activities performed and we provide an evaluation of the performance of the system. Here we highlight the advantages and drawbacks of the developed IPS. Finally, we give concluding remarks in the the last section of the paper.

2 Camera-Based Localization System

The localization system is based on an array of cameras, which are placed above the area of interest, where the robots can move. The individual camera systems in its composition identify the robots based on their attached markers and estimate their position and orientation in the working area coordination frame. The estimated poses of the robots are collected by a main server for monitoring and transmitting the information to other clients, such as the actual robots. Based on these data, the robots can take action based on their position in the field.

2.1 Hardware Setup

Each Raspberry camera was set at 3.3 m height from the ground, mounted on the elements on the ceiling with the mechanism presented in Fig. 1. This is made up of a shaft, a flexible joint, and 3D printed components that hold the pieces of each individual system together (Raspberry Pi and Raspberry Pi camera) and fix each one of the separate kits on the objects on the ceiling. All these are connected to the mains through a 5 V power supply adapter.

The resolution of the camera was set from a SW (Software) point of view at 1640x1232 binned, with a full FOV (Field Of View), more precisely 48.8 degrees Vertical and 62.2 degrees Horizontal. The focal length of the camera is 3.04 millimeters. Lens distortion was tested and no major side-effect was noticed from it. In conclusion, the decision of not considering the lens distortion matter was taken. This leads to an area covered by each Raspberry Pi of 3.36x2.46 m (8.29 m^2). The full area of the track is 204.1 m^2, so we installed 24 cameras. The part of the competition track covered by the RPi cameras is presented in Fig. 2 (coloured in red).

The ArUco markers for the detection of the model cars are set at 4x4 bits with a total dimension of 10x10 cm. The markers are placed on top of each car, so at a height of approximately 20 cm. This being testified, each marker is the equivalent of 51.9x53.1 Px (Pixels), which gives a theoretical

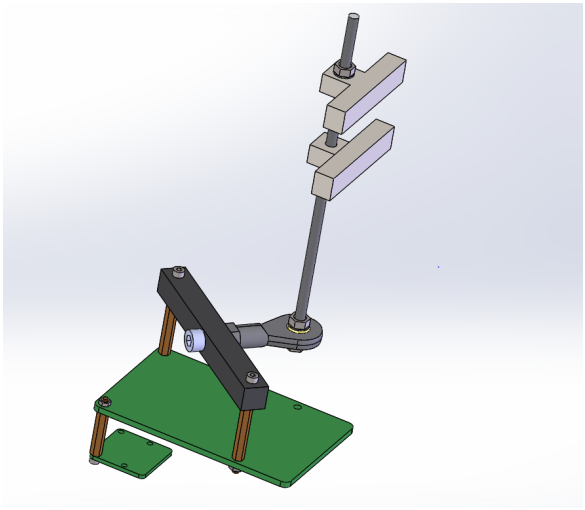


Figure 1: Camera system mechanical design

maximum error of 1x1 Px, e.g.: 5.19x5.31 cm. The software running on each RPi contains the list of car markers, and each time one of the cameras detects a known marker, it sends the corresponding computed location and orientation to the server.

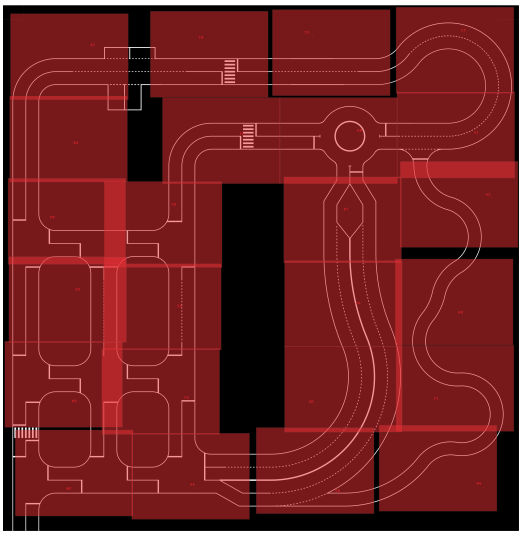


Figure 2: Camera system FOV

2.2 Software Modules

The system is composed of one server module and many camera modules, where the camera modules can start and stop independently from the server. The camera module aims to identify the vehicles by detecting the markers and transfers their poses on the track to the main server. The camera module has two phases, which are the calibration and localisation processes, each of them based on marker detection. The camera modules are ROS nodes realized with OpenCV with C++. The main server deals with multiple tasks: it collects the robot's poses from each camera module and makes a list of all acquired poses of the robots present in the tracked area. The main server is a ROS node implemented in Python. It also has some features for controlling and monitoring all the camera modules on the local network and visualizes the detection on the track map. A diagram scheme can be seen in Fig 3.

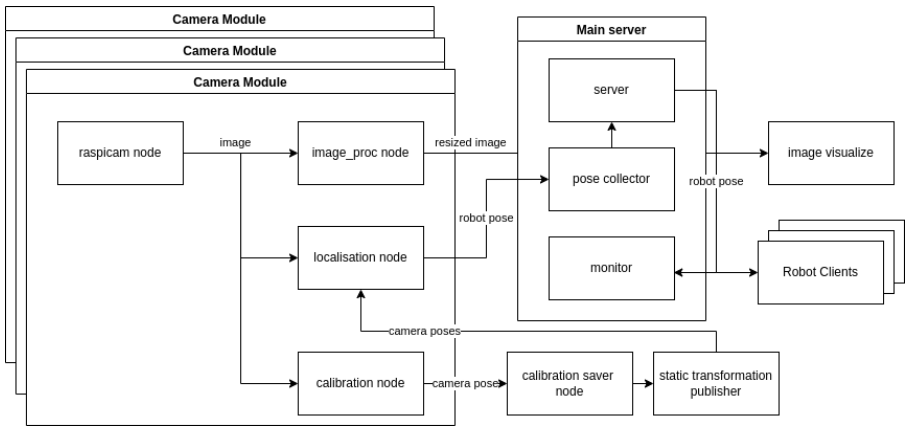


Figure 3: Software architecture

2.2.1 Marker Detection Process

There are two aspects identical for both the calibration and the localisation phase: the camera handling and the marker detection.

For the camera handling we use the raspicam node, a predefined ROS package. It implements the low level interaction with the raspberry camera and it publishes the images on a ROS topic. The image_proc image processing package [23] was used for resizing the acquired images and publishing them on a different image topic. This was performed for debugging purposes, allowing the visualization of images on another device, with an improved frame rate. The image_proc image processing package also publishes and subscribes the camera parameters on a camera info topic, where these parameters can be changed.

The detection process takes the following inputs: the dictionary of ArUco markers to look for, the image topic, and some camera parameters. By using this data and the OpenCV library, more precisely the detectMarkers function, we identify the detected markers and find their corner positions. These are then used for computing the position and orientation of the markers in the camera frame.

Fig. 4 shows the image captured by a Raspberry Pi camera, where indicators are placed on the detected markers.

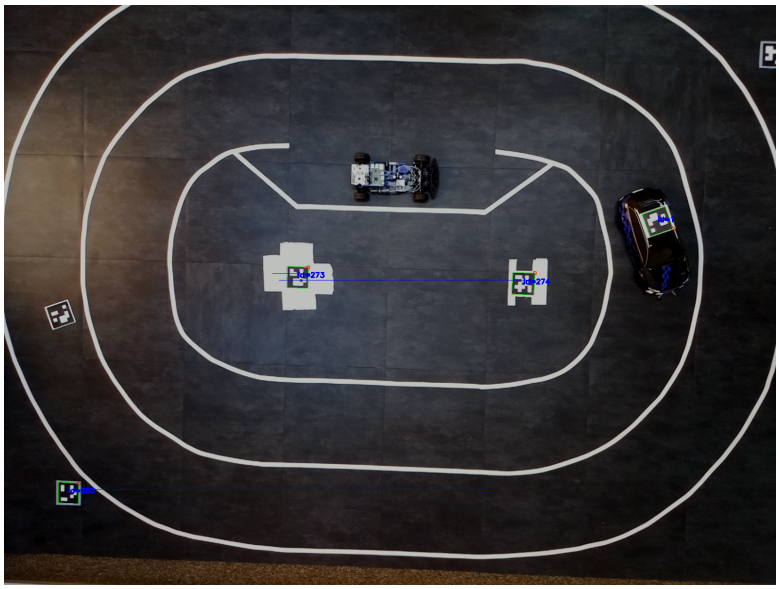


Figure 4: Detected markers

2.2.2 Calibration process

In the calibration phase, the system focuses on determining the poses for each camera on the track coordination system. For calibration purposes, ArUco markers are set at 5x5 bits with a total dimension of 10x10 cm. Those are are placed on the flat surface with rotation 0, in a matrix where they are set one meter apart from each other. In this way, a matrix like the one in Fig. 5 is constructed on the track. This being testified, each marker is the equivalent of 48.7x49.9 Px, with a maximum error of 1x1 Px, e.g.: 4.87x4.99 cm. [19].

Figure 5: Calibration setup

During the calibration process, each Raspberry Pi system runs a script capturing the static image with the calibration markers placed on the ground. We are using the above mentioned set of markers, whose position are predefined on the track and can be detected by all the camera modules. The position and the size of the markers in the track coordination frame represent the inputs of the system. Based on these inputs, the pose of each corner is calculated. For each marker detected by a camera module, the software application creates one list with the position of the corners in the world and one with the position of the corners on the frame. At this point we have a set of image points and the correlated 3D points in the track coordination frame. Finally, based of the detected corners, we estimate the camera rotation and translation within the track coordination frame and we minimize the reprojection error from 3D points to 2D image points. We determine the transformation and rotation vectors by using another function provided by OpenCV, SolvePnP. The vectors are obtained taking into account the camera intrinsic parameters. These vectors transform a point expressed in the world coordinate frame to the camera coordinate frame. By inverting the transformation matrix formed by these vectors, we get the translation and rotation vectors, which transform a point from the camera coordination frame to the world coordination frame. The resulted transformation is broadcasted using the ROS tf package [24]. Another ROS node running on the calibration server collects the estimated transformations for all the devices in the image frame. At the end of the calibration phase, the calibration saver calculates a mean of the transformations for each camera and saves them in a ROS launcher file for their application in the localization phase. After the calibration phase, the translation and rotation vectors are memorized, the marker matrix is no longer necessary, and the markers are removed from the track. An image of the calibration matrix for a test track, with markers positioned so that they are on the same plane with the one mounted on a car, can be seen in Fig. 6. This setup was used for testing the concept presented in this subsection.

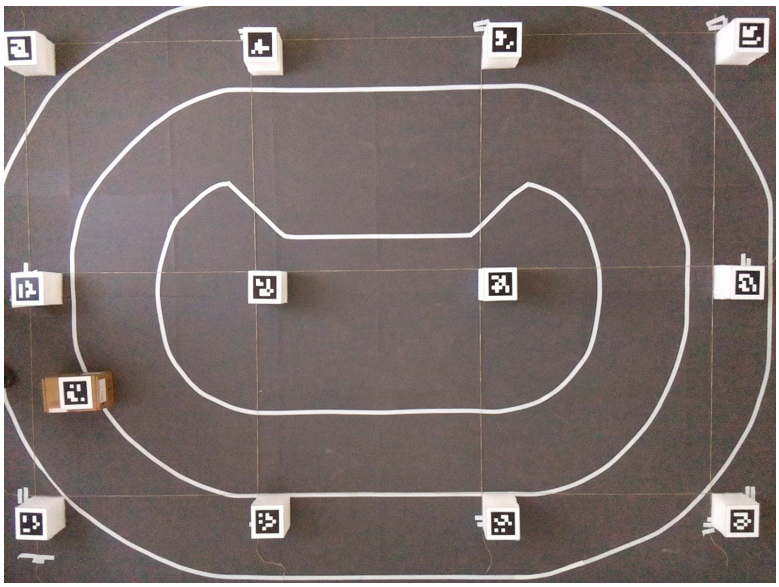


Figure 6: Calibration setup for test track

2.2.3 Localisation process

In the localisation phase, the camera module aims to determine the car position and orientation in the world coordination frame. The already saved transformations between the camera and track coordination frames are repeatedly published in the ROS system. A different marker is placed on the top of each car to identify the robot and also to detect it on the image frame. In the first step, the camera system and detection system detect the marker corners in the image frame, which can belong to a car or not, and filter the unidentified markers. The detector module uses the camera intrinsic parameters and a marker size to estimate each car translation and rotation in the camera coordination frame and it runs the estimatePoseSingleMarkers OpenCV function. Based on the size of the marker the estimatePoseSingleMarkers function predicts the corners position around the marker center and it solves the pose estimation problem by minimizing the reprojection error from 3D points to image points. This action will result in the translation and rotation vectors of markers on the camera coordinate frame. By applying the previously estimated transformation matrix from the camera coordination frame to the world coordination frame, we will get the car's position and orientation in the world coordination frame. Each individual camera system repeats the process for each image frame, where it can detect the markers with a known ID. The camera modules publishes the detected cars' position and orientation on different ROS topics, identified to the camera itself, to be read by the main server.

2.2.4 Servering

The main server module aims to serve the robots with their estimated position and orientation on the race track. It collects all the translations from the camera modules. The module discovers automatically all the ROS topics that belong to a camera module and subscribes to them to get the estimated poses on each camera frame. These poses are stored in a table based on the robot identification number defined by the marker placed on them. The server module is designed to provide the robot's poses for clients without using a ROS subscriber. The server broadcasts periodically a beacon through UDP (User Datagram Protocol), which contains the server's port number. On this port the server waits for connections from clients and creates a new thread for serving each of them. The server implementation is based on ThreadingTCPServer from the SocketServer Python standard library. For establishing a connection, a series of validations need to be done: the robots request to connect to the servers port number by sending the ID they are interested in. The server then replies with a custom message sent in plain text and the same one encrypted with a private RSA (Rivest-Shamir-Adleman) key. The client then confirms the server identity by making a decryption of the

coded message with the public key of the key-pair, previously provided together with the API. If the decrypted message is the same as the one in plain text, the server is validated and the connection is finalised. After they establish the connection, the previously created thread periodically transfers to the robot the coordinates (orientation and position) from the corresponding connection ID. The main server module can register more than one client to the same identification number and in this way a robot can obtain also the position of other robots.

2.2.5 Monitoring

Two separate modules have been developed for monitoring the entire localisation system, one focused on the hardware functionality (ensuring that the processes are running on all the devices) and one focused on the controlling and monitoring of the entire competition infrastructure. The first module is monitoring the state of each Raspberry Pi single-board computer in the system. This application runs on the server and periodically checks that the remote systems making up the GPS are alive and the ArUco marker detection script runs on every one of them. The main loop starts a thread for each Raspberry Pi in the system, using the `paramiko` package from Python3, opens up an SSHv2 (Secure Shell 2.0 or SSH 2) connection, and launches a command that searches for the ID of the process belonging to the detector script. It then checks the output of the remote system, and if the command does not contain any lines, this means that detection is not turned on. This situation is indicated on the screen of the server, so that action can be taken. This could mean starting the script remotely, or restarting the Raspberry Pi manually, in case no SSHv2 connection can be established. The Observer pattern is used for subscribing to the checking action performed for each Raspberry Pi and for getting notifications about changes in their states. These notifications are used for updating the interface with the user. The output of the monitoring script can be seen in Fig. 7.

```
RPi status (True - WORKING, False - NOT WORKING):
('192.168.1.50', True)
('192.168.1.51', True)
('192.168.1.52', True)
('192.168.1.53', True)
('192.168.1.54', False)
('192.168.1.55', True)
('192.168.1.56', True)
('192.168.1.57', True)
('192.168.1.58', True)
('192.168.1.59', True)
('192.168.1.60', True)
```

Figure 7: Monitoring system output

The second module is a GUI (Graphic User Interface) of the entire competition environment. Visually, the user of the application can see the detected cars on the track, the data accuracy, delay and potential problems. Additionally, the user can activate and monitor the states of all the active elements on the track (semaphores, pedestrians, timers). The application is developed in ROS and it has a node for each car ID, which can spawn on the map and it uses the API (Application Programming Interface) that we share with the teams. The API is creating a thread that is connecting to the server and gathering the acquired position and orientation pairs. If the connection doesn't take place, then we can identify the problem on the server. The node then publishes the raw given coordinates on a topic where the GUI application is subscribed. The GUI has a frame rate of 10 fps (frames per second) and a map of the competition as background. At each frame update it deletes the old positions of the cars and updates it with the newly acquired ones. If no position is gathered in a certain area from a single car, we can then check if the car marker is rightly set or, if no position is gathered at all, we can take proper action on the system itself. The output of the GUI can be seen in Fig. 8.

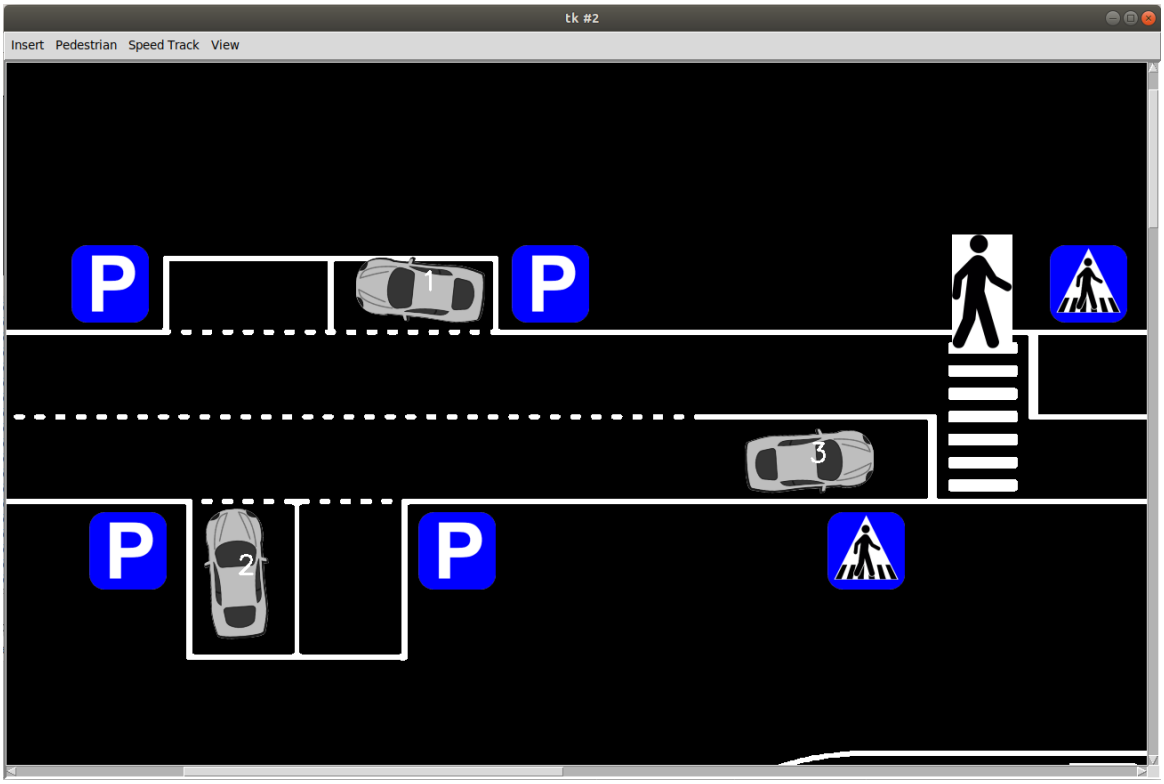


Figure 8: GUI application (masteriX)

3 Performance Evaluation

In order to best evaluate the performance of the system, no uniform control of the lighting in the environment was performed, since during the competition itself there is no possibility of creating even and controlled lighting conditions. Four different aspects were analysed:

- **Delay:** The time passed between the captured frame (CF) from the camera until the served position (SP) was read by the vehicle.
- **Frequency of delivery:** The frequency of the received messages by the vehicle. The theoretical frequency of the captured frames by each device of the system, which is around 4 Hertz.
- **Ratio between the captured positions and lost positions:** On average, how many lost positions should be expected?
- **Accuracy:** The difference in position between the position of the car in the captured frame and the actual position of the car in the frame.

The network of the competition has no internet access and a method of time synchronisation had to be adopted. A custom Network Time Protocol (NTP) server was installed on the master computer (the one that runs all the servers of the competition) and all the connected devices were configured to look for time synchronisation not on specific Pool Zones, but on the custom server. An additional UDP exception for communication with the server had to be implemented on the clients.

Ten randomly considered consecutive streamed positions were taken into account for the calculation of the Delay (Table 1). It should also be noted that on the network there were no participant vehicles. The presence of communicating vehicles in the setup would have most probably lowered the network speed.

Served position time[s]	Captured frame time [s]	x [m]	y [m]	delay time [s]
1614869132.38	1614869131.97	0.79	14.76	0.41
1614869132.63	1614869132.17	0.81	14.67	0.46
1614869132.88	1614869132.37	0.80	14.59	0.51
1614869133.13	1614869132.77	0.83	14.40	0.36
1614869133.38	1614869132.97	0.87	14.30	0.41
1614869133.63	1614869133.17	0.91	14.21	0.46
1614869133.88	1614869133.57	1.02	14.05	0.31
1614869134.13	1614869133.77	1.11	13.96	0.36
1614869134.38	1614869133.98	1.17	13.89	0.40
1614869134.63	1614869134.38	1.26	13.81	0.25

Table 1: Run data sample

Average Delay:

$$\frac{\sum delay}{10} = \frac{3.9}{10} = 0.39s \tag{1}$$

The calculus in equation (1) shows that the delay is a problem to be considered. This delay is the sum of all the periods of time required by calculations and data transmissions within the network. The following delays have been identified:

- the time starting from the frame capture until the calculation of the marker pose
- time of data delivery from the device to the server,
- time of data delivery concerning the position from the server to the robot,
- time lost in the synchronisation of data read and write actions.

It is also worth mentioning that, in an empty environment on the same server computer, the communication between two ROS nodes completes in around 0.15 seconds.

From this same table, we can also calculate the frequency of detection and of data delivery.
Frequency of delivery:

$$\frac{SP_{max} - SP_{min}}{10} = \frac{1614869134.63 - 1614869132.38}{10} = \frac{2.3}{10} = 0.23T = 4.35Hz \tag{2}$$

The calculus in equation (2) indicates that without interference, the ArUco markers are detected flawlessly and sent accordingly to the server clients.

In order to correctly evaluate the signal loss, a trajectory was highlighted on the ground at known positions, starting from an area of the track with full natural light and reflective surface to an area where indoor lighting was necessary, and where no reflective surface was encountered. The vehicle was driven manually through this trajectory on three separate runs, at nearly 5 minutes difference between them, and the middle one in performance will be considered onwards.

The chosen run took 76.6 seconds to be completed, which means that there should be 338.86 theoretically captured positions (TCP). On our data set instead, there was a number of 200 captured positions(CP), meaning that we have a considerable amount of lost positions. Based on this, the ratio of lost positions can be computed (equation (3)).

Ratio of lost positions:

$$\frac{CP}{TCP - CP} = \frac{200}{333.86 - 200} = \frac{200}{133.86} = 1.49 \tag{3}$$

A visual representation of the run with the acquired positions can be seen in in Fig. 9.

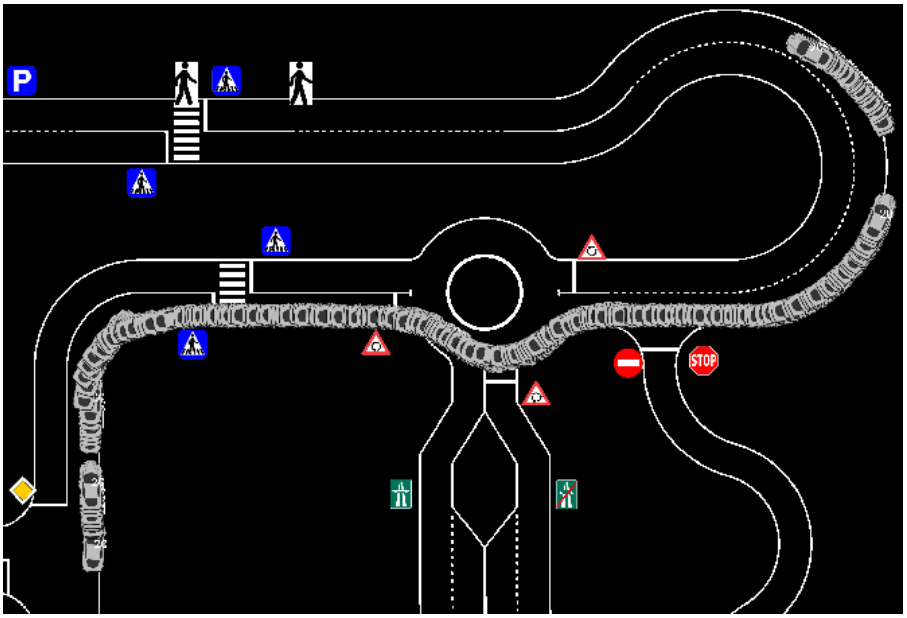


Figure 9: Complete run output

A sum of factors have or can potentially influence this ratio, drastically and dynamically, all touching in fact the camera parameters. The uneven lighting conditions on the track, sudden light flashes (due to people walking) or the speed of the vehicles and its trembling (due to inappropriate fixing of the car body or the ArUco marker).

Accuracy: 10 cm radius circle

For the last evaluated point, accuracy, a reverse calibration method was used: after running the calibration process, we gave as an input the same calibration dictionary for the detection process, saved the data of all the IDs on the map and then compared them to their initial positions, which we already knew. The maximum error was observed, having a value of 10 cm. The accuracy of the camera position in the world coordinates was not observed separately, as it would include not only the measurement of position of the camera, but it's rotation matrix as well.

There are two principal aspects that influence in big terms the accuracy:

- the distance between the markers and the camera – 3.2 meters;
- the fact that we have two accumulated errors, i.e., one from the calibration and the one from the detection.

The accuracy was determined also empirically, by placing a vehicle on the track, and comparing the positions given by our localisation system with physical measurements. This comparison was performed in the parts of the track that provide the worst conditions for position calculation: area covered by a minimum number of cameras, low level of natural light, and high amount of reflections. These areas were selected for obtaining the accuracy in the most unfavourable cases.

Different aspects were raised during the development and during the competition, and they led us to the following observations:

- **Difficulty in setting up the system:** There is no possibility of setting up a fixed matrix of devices covering the entire competition area, due to the fact that the area was not designed specifically for this purpose. As a conclusion some parts can be uncovered.
- **Difficult calibration process:** Setting up a floor matrix of approximately 200 ArUco markers is quite repetitive and challenging.
- **Problematic maintenance:** If during the competition, a device malfunctioned, it was necessary to re-calibrate that specific device, which would require some time.

- **Best for controlled ambient:** In our situation, many aspects can influence the specifications of the system (car speed, car body mounting, light disturbance, people, network usage).

4 Conclusion

The work presented in this paper deals with the development of an indoor localization system for use in the Bosch Future Mobility Challenge (BFMC) international autonomous driving and connectivity competition on 1/10 scale vehicles. Our solution is a distributed system, which is based on a network of Raspberry Pi single-board computers running ROS nodes, fitted with cameras, that monitor a certain area of the 14x14 m plane. The camera-based systems monitor the area of interest and detect and calculate the position of ArUco tags attached to the tracked robots. The system was extensively tested and used in the actual competition, proving to be a robust and scalable solution. The presented system can be easily adapted for different coverage areas with minimum effort. The performance was similar between a system with 6 devices and one with 24 devices.

The initially planned target points of the developed system were met. The entire system performed within acceptable parameters, representing a replica of real-life GPS. The system was successfully used during the BFMC2021 qualifications and finals, where nine teams used it without major problems. Some of them were able to create the map based on the data gathered from the server. In the future, we propose to upgrade our system to ROS2 for reducing the appeared latency and we intend to compare the performance of solution with a localization system based on Ultra WideBand (UWB).

Funding

This work was developed under IPCEI ME/CT EuroDrives.

Author contributions

The authors contributed equally to this work.

Conflict of interest

The authors declare no conflict of interest.

References

[1] Alarifi, A., Al-Salman, A., Alsaleh, M., Alnafessah, A., Al-Hadhrami, S., Al-Ammar, M. A., Al-Khalifa, H. S. (2016). Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances. *Sensors*, 16(5). https://doi.org/10.3390/s16050707

[2] Assayag, Y., Oliveira, H., Souto, E., Barreto, R., Pazzi, R. (2020). Indoor Positioning System Using Dynamic Model Estimation. *Sensors*, 20(24). https://doi.org/10.3390/s20247003

[3] Chang, Q., Li, Q., Shi, Z., Chen, W., Wang, W. (2016). Scalable Indoor Localization via Mobile Crowdsourcing and Gaussian Process. *Sensors*, 16(3). https://doi.org/10.3390/s16030381

[4] Heya, T. A., Arefin, S. E., Chakrabarty, A., Alam, M. (2018). Image Processing Based Indoor Localization System for Assisting Visually Impaired People. *2018 Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS)*, 1–7. https://doi.org/10.1109/UPINLBS.2018.8559936

[5] Hirota, T., Tanaka, S., Iwasaki, T., Hosaka, H., Sasaki, K., Enomoto, M., Ando, H. (2007). DEVELOPMENT OF LOCAL POSITIONING SYSTEM USING BLUETOOTH. In *E. Arai & T. Arai (Eds.), Mechatronics for Safety, Security and Dependability in a New Era (pp. 309–312)*. Elsevier. https://doi.org/https://doi.org/10.1016/B978-008044963-0/50063-1

[6] Hua, J., He, L., Kang, Z., Yan, K. (2019). A force/position hybrid controller for rehabilitation robot [Article]. *International Journal of Computers, Communications and Control*, 14(5), 615–628.

[7] Kala, R. (2016). 2 - Basics of Autonomous Vehicles. In *R. Kala (Ed.), On-Road Intelligent Vehicles (pp. 11–35)*. Butterworth-Heinemann. https://doi.org/https://doi.org/10.1016/B978-0-12-803729-4.00002-7

[8] Khelifi, F., Bradai, A., Benslimane, A., Rawat, P., Atri, M. (2019). A Survey of Localization Systems in Internet of Things. *In Mobile Networks and Applications*, 24(3), 761–785. https://doi.org/10.1007/s11036-018-1090-3

[9] Kilyen, N. A., Lemnariu, R. F., Mois, G. D., Chen, Y., Morris, B. T., Muntean, I. (2021). The IEEE ITSS and Bosch Future Mobility Challenge: A Hands-on Start to Autonomous Driving [Technical Activities]. *IEEE Intelligent Transportation Systems Magazine*, 13(3), 276–282. https://doi.org/10.1109/MITS.2021.3081939

[10] Kushwaha, M.; Abirami, M.S. (2023). Intelligent Model for Avoiding Road Accidents Using Artificial NeuralNetwork, *International Journal of Computers Communications&Control*, 18(5), 5317, 2023.https://doi.org/10.15837/ijccc.2023.5.5317

[11] Lee, H., Yoon, J., Jang, M.-S., Park, K.-J. (2021). A Robot Operating System Framework for Secure UAV Communications. *Sensors*, 21(4). https://doi.org/10.3390/s21041369

[12] Mocanu, I., Scarlat, G., Rusu, L., Pandelica, I., Cramariuc, B. (2018). Indoor localisation through probabilistic ontologies [Article]. *International Journal of Computers, Communications and Control*, 13(6), 988–1006. https://doi.org/10.15837/ijccc.2018.6.3022

[13] Nebbou, T., Lehsaini, M., Fouchal, H., Ayaida, M. (2019). An urban location service for vehicular area networks. *Concurrency and Computation: Practice and Experience*, 31(24), e4693. https://doi.org/https://doi.org/10.1002/cpe.4693

[14] Obeidat, H., Shuaieb, W., Obeidat, O., Abd-Alhameed, R. (2021). A Review of Indoor Localization Techniques and Wireless Technologies. *Wireless Personal Communications*, 119(1), 289–327. https://doi.org/10.1007/s11277-021-08209-5

[15] Oščádal, P., Heczko, D., Vysocký, A., Mlotek, J., Novák, P., Virgala, I., Sukop, M., Bobovský, Z. (2020). Improved Pose Estimation of Aruco Tags Using a Novel 3D Placement Strategy. *Sensors*, 20(17). https://doi.org/10.3390/s20174825

[16] Schauer, L. (2019). 2 - Wi-Fi Tracking Threatens Users' Privacy in Fingerprinting Techniques. In *J. Conesa, A. Pérez-Navarro, J. Torres-Sospedra, and R. Montoliu (Eds.), Geographical and Fingerprinting Data to Create Systems for Indoor Positioning and Indoor/Outdoor Navigation (pp. 21–43)*. Academic Press. https://doi.org/https://doi.org/10.1016/B978-0-12-813189-3.00002-2

[17] Tomažič, S. (2021). Indoor Positioning and Navigation. *Sensors*, 21(14). https://doi.org/10.3390/s21144793

[18] Zhang, L., Cheng, Q., Wang, Y., Zeadally, S. (2008). A Novel Distributed Sensor Positioning System Using the Dual of Target Tracking. *IEEE Transactions on Computers*, 57(2), 246–260. https://doi.org/10.1109/TC.2007.70792

[19] [Online]. Available: https://picamera.readthedocs.io/en/release-1.13/, Accesed on 14 March 2023.

[20] [Online]. Available: https://opencv.org/#, Accesed on 1 September 2023.

[21] [Online]. Available: https://sourceforge.net/projects/aruco/, Accesed on 10 August 2023.

[22] [Online]. Available: http://wiki.ros.org/nodelet, Accesed on 10 August 2023.

[23] [Online]. Available: http://wiki.ros.org/image_proc, Accessed: 7 December 2022.

[24] [Online]. Available: http://wiki.ros.org/tf, Accessed: 20 july 2023.

Journal's webpage: http://univagora.ro/jour/index.php/ijccc/

This journal is a member of, and subscribes to the principles of,
the Committee on Publication Ethics (COPE).
https://publicationethics.org/members/international-journal-computers-communications-and-control

Cite this paper as:

Kilyen, N.A.; Lemnariu, R.F.; Muntean, I.; Mois, G.D. (2024). An Indoor Localization System for Automotive Driving Competitions, *International Journal of Computers Communications & Control*, 19(1), 6030, 2024.

https://doi.org/10.15837/ijccc.2024.1.6030