# Prototyping platform design for visual robot control experiments

**V O Mihalca [1], D M Anton[2] and R C Țarcă [3]**

[1] PhD School of Engineering Sciences, University of Oradea, Romania
https://orcid.org/0000-0002-0370-3946/

[2] PhD School of Engineering Sciences, University of Oradea, Romania
https://orcid.org/0000-0002-2839-5599

[3] Mechatronics dep., Faculty of Managerial and Technological Engineering, University of Oradea, Romania
https://orcid.org/0000-0002-1061-7552

ovidiu@vmihalca.ro

**Abstract**. The current work describes a system and application architecture for a platform developed with the objective of prototyping and exploring vision-based control strategies for mobile robots. The main design concept is illustrated in block diagrams and the structure as well as connection of components is discussed. Certain implementation details, including small code snippets or screenshots of the graphical user interface, are also provided in order to facilitate understanding of the proposed architecture. Advantages and disadvantages of the approach, together with several future directions of research and development, are enumerated in the final section.

## 1. Introduction

The idea of a prototyping platform for experiments regarding visual control of mobile robots came into existence as a development for a robotic system comprised of two mobile robots: a target (leader) and a follower. This system was built to implement and test a target tracking task achieved using a vision-based control scheme for the follower robot.

The control scheme involves a specific object having a circular shape. Image features are extracted from each frame captured by the camera sensor mounted on the follower robot. These features describe the pose of the target object which is mounted on the leader mobile robot. The robotic system described in this work is part of a thesis concerned with vision-based control of mobile robots, used in implementing objectives such as target-tracking by means of simple tasks, as described by P. Corke in [1] – driving the prototype to a point, following a line or a moving feature. In certain multi-robot systems, such as swarm types, agents do not possess many hardware features (are unsophisticated) and objectives are accomplished by composition of similar simple tasks[2].

During the implementation of physical mobile robot prototypes, it has been found easier to take advantage of the processing power of a desktop computer for the frame processing algorithms. Another advantage consisted in eliminating the need to deploy changes to the mobile robot application, thus

shortening the write-execute development cycles and improving the overall efficiency of the implementation effort.

This platform architecture draws inspiration from several different works: using TCP/IP based networks for remote operations, such as performing remote experiments in a virtual lab with simulation software MATLAB and Simulink[3] or using a custom desktop application with a graphical user interface for remote control of a robotic arm[4]. As stated in [5], simulation type software can also be used as a highly-visual[6] and interactive VR-like user interface, designed for training UAV pilots[7] or even as a remote interface for an embedded system, with real-time monitoring capabilities[8].

Similar control loops, structured for distributed computing, are used for studying multiple aspects of mobile robots. Not only mobility, but also stability control strategies are tested, such as for six-legged walkers[9]. Another inspiration for distributing the workload between a host computer and mobile robot hardware represents the HIL concept, exposing microcontroller facilities, as described in [10].

Visual servoing is used with mobile robots spanning a wide variety of applications, to give very few examples: inspection of power lines[11] or laboratory electronic instruments[12]. Small-scale mobile robots and systems like the one proposed in this paper are becoming ubiquitous, having numerous applications in healthcare[13], infrastructure maintenance[14], cleaning[15] and UV-C disinfection[16], or agricultural automation[17].

## 2. Proposed system architecture

### *2.1. Robotic system*

The initial system proposed consists of two mobile robots: a target and a follower. The latter features a camera sensor mounted in a central location upon its chassis. The former features a *visual target* object

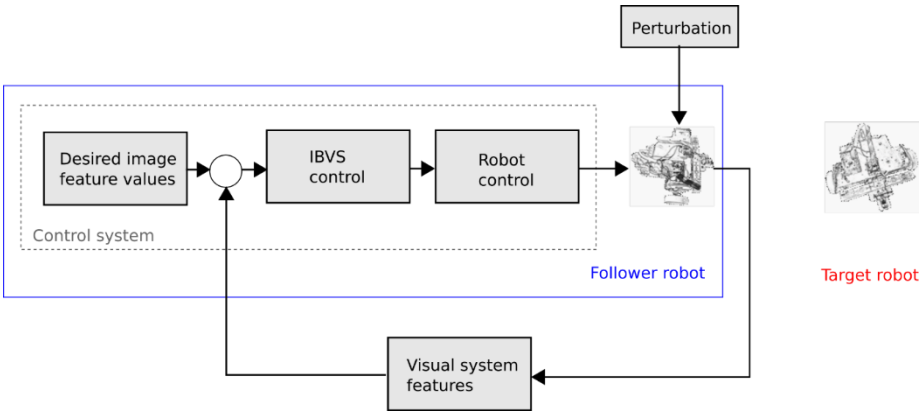


*Figure 1 Information flow in proposed system – block diagram*

with a specific, circular shape also mounted centrally on its own chassis. The follower robot acquired visual information through its sensor, which contains the visual target from the leader robot. This visual information is sent to the follower's control subsystem, which uses a vision-based procedure to extract features and determine the control signals necessary to drive the follower towards its goal. This principle of operation is illustrated in the block diagram from Figure 1.

### *2.2. Prototyping platform*

It can be seen that the system components which define both the processing of visual information and robot control are loosely-coupled. This provides the flexibility to implement them in a distributed computing fashion, as long as there exists a means to preserve the information flow through a communications channel.

In order to take advantage of the processing power that a desktop workstation provides, the system can be designed to implement the high-level control strategy on such a workstation, while obtaining the necessary visual information from the mobile robot. The loop is closed by sending remote control commands to the mobile robot, which translates them into the necessary signals to be sent to the actuators. The mobile robot therefore is given the role of an information-gatherer and actuator platform, while the desktop application becomes the "brain" of the proposed system architecture. This architecture is illustrated in Figure 2.

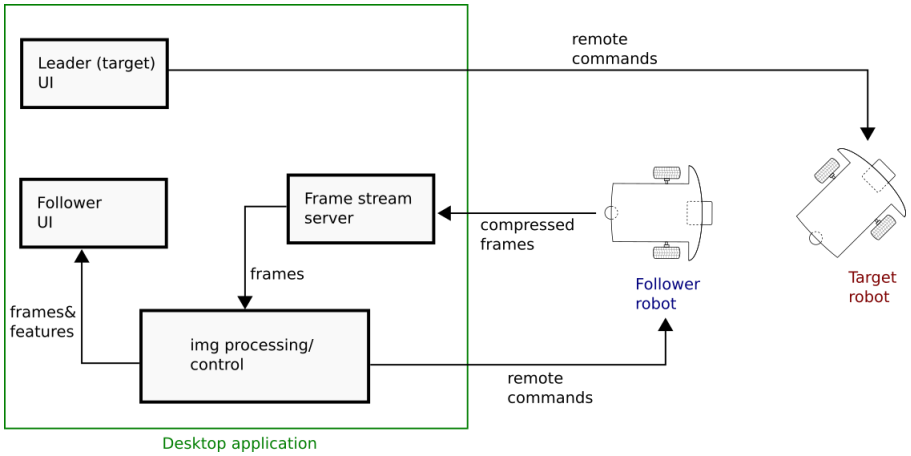


*Figure 2 General architecture of the prototyping platform*

Several subsystems from the first block diagram are now components of the desktop application. They fulfil the roles of control, as well as feature extraction and error computation. There are also auxiliary components which facilitate network communications. All the desktop application components follow a structure imposed by the software framework used in its implementation.

Information flow between the mobile robots and the desktop applications is done through a TCP/IP network, using multiple channels in parallel. These channels are of multiple types, either TCP-based streams for visual information or UDP-based datagrams for remote commands.

Similar with the case of IoT devices, receiving information from a remote data-gatherer agent such as the mobile robots in this case exposes the flow to risks of packet loss, delays or network-specific issues. Some of these issues can be avoided using solutions such as data validation modules[18] or by software means, employing algorithms for signal processing[19], [20].

## 3. Architecture of applications

### *3.1. Desktop application*

Two essential roles of the desktop application are to provide a user interface to visualize results in real time and debug the high-level strategy as needed, as well as the role of implementing the vision-based control scheme by employing image processing procedures upon the frames received from the follower robot.

Given these requirements, the software framework adopted to help build the desktop application was Qt Framework. Reasons for the choice include not only the author's familiarity and experience with it, but also the fact that it provides multiple ways to implement modern user interfaces. Qt Framework codebase is written in the C++ programming language and there are Python bindings which make it a viable option for projects implemented in any of these languages. This allows future optimization by re-implementing proof-of-concept source code from Python to a compiled language such as C++, with performance benefits in doing so.

Qt-based applications are organized around multiple loosely-coupled components which in practice are derived from a base class named `QObject`. These QObjects make use of an asynchronous message-

passing mechanism called Signals-Slots. This allows both direct and queued method invocation which is thread-safe and can be done from multiple threads, making concurrency easy to implement. The UI-specific components are also QObject-based, therefore they integrate well with other subclasses which follow the same conventions.

The architecture of the desktop application, showcasing its major components, is shown in Figure 3.

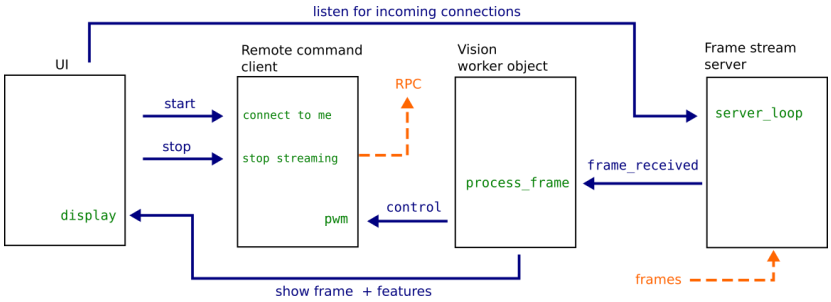


*Figure 3 Desktop application architecture*

*3.1.1. Network communication.* Visual information is received through a TCP stream. The component responsible for this acts as a TCP server, embedding the receiving network code loop in a Qt slot. The loop breaks upon a stop condition which can be controlled from outside the server class. Each image frame received is decoded as a NumPy array and emitted as a Qt signal which can be connected to any slot accepting a NumPy array. This isolates the code necessary to implement the custom streaming protocol from the rest of the application, ensuring a better separation of roles.

Remote commands are sent to the robot making use of a RPC-based technology. Given that both the desktop and robot application have Python implementations, the RPC implementation chosen is provided by the RPyC package. It makes use of language features to achieve simple yet efficient Remote Procedure Call services. RPyC requires a connection to be established first, which yields a connection object. Method calls may be issued directly on this proxy object, which will be serialized and transported to the RPC server where they will be interpreted and the result returned over the wire back to the caller.

This flow is embedded within another QObject. In order to allow greater flexibility in issuing remote commands, the remote command client object exposes several slots which can be called from anywhere inside the application, regardless of the thread of execution. These slots wrap the actual method calls on the proxy object and hide RPyC-specific implementation details from the rest of the application.

*3.1.2. GUI.* User interface components are part of the Qt Widgets sub-framework. Qt Widgets represents a technology for building traditional desktop user interfaces in C++. There is a PySide6 sub-package which wraps all the components of Qt Widgets, allowing for easy definitions of components necessary to build the user interface. These components are connected to the rest of the application through the same signal-slot mechanism.

The interface is composed of two essential windows: the Connect Window and the Follower Main Window. The first one represents a modal dialog window which blocks any other part of the application until a connection is successfully established with the robot controlled. It allows the user to input an address-port pair which is typical of TCP networks.

The main follower window allows the actual control of the robot. It does not feature many elements, rather it delegates to the other components of the application the responsibility of sending control commands. The UI allows the user to start streaming of image frames, which are processed and allow remote commands to be issued.

### *3.2. Robot application*

The application executing on the follower robot is also implemented in Python, not only for faster development but also due to the availability of modules implementing hardware functionality. The main

processor board driving the follower robot is a Raspberry Pi model 3B running a Linux-based operating system. The board interfaces with the motor driver bridge through GPIO pins and makes use of the `RPi.GPIO` Python module. In addition to digital input, the H-Bridge module also requires PWM input. Since the number of hardware PWM channels on the board is limited to 2 and one of them is needed to drive the camera micro-servo motor, an additional module board is used which delivers up to 16 PWM channels and is controlled via an $I^2C$ interface.

Additional reasons to use a higher-level approach (such as implementing the application in Python) are the possibility to structure the application in a multithreaded fashion, as well as using TCP/IP programming interfaces easier.

The architecture of the application is based on multiple threads. The main thread runs the RPC server loop and listens for incoming connections. A secondary thread spawns on demand, in order to establish a TCP-based stream and start sending images captured from the camera sensor.

## 4. Components implementation

### *4.1. Network components*

Receiving image frames captured by the camera sensor takes place over a TCP stream. When designing the system, several approaches have been made to achieve this information flow, including both high-level and low-level protocols and technologies. One of the attempts involved using a M-JPEG stream that would be delivered by a web server running on the robot[21]. The Python framework Tornado was considered for implementing the server, as it is asynchronous in nature and allows for high concurrency, making it easy to integrate with other application components. The overhead of HTTP which reduced performance (and even created issues when testing with a web browser), as well as a lack of quality M-JPEG client packages led to discarding this approach.

The final solution was based on the idea presented as a "recipe" in the documentation for the camera sensor library, `picamera`[22]. It consists of a simple, custom protocol which continuously streams frames separated by a 4-byte integer representing the size of the next frame to be read. If the size value is zero, it flags the end of the stream; no more frames are to be read and the connection is closed on the server-side as well. This protocol is shown visually in Figure 4.

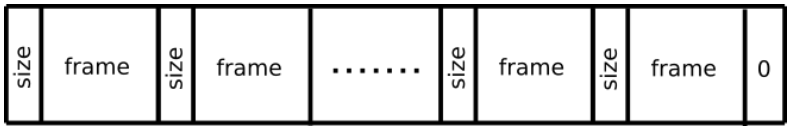


*Figure 4 Camera frame streaming protocol*

In order to control the robot and close the feedback loop, a mechanism of remote commands has been developed. While possibly more efficient to build a custom protocol using TCP or UDP, it was opted for a Remote Procedure Call approach. The `rpyc` package provides a RPC implementation taking advantage of Python language features to create services for distributed computing. One such service runs inside the robot application. Its loop listens for connections and method calls in the main thread of execution.

The service defines the available operations that can be invoked remotely. This is done using a Python class with decorated method definitions, as illustrated in the following code listing.

```
@rpyc.service
class CmdService(rpyc.Service):
    def on_connect(self, conn):
        print(self.remote_addr(), 'connected')

    # Get sensor data (control streaming)
    @rpyc.exposed
    def stream(self, port: int):
```

```
        self.context.client.addr = self.remote_addr(), port
        self.context.client.start()

    @rpyc.exposed
    def pause(self):    # stop streaming camera
        self.context.client.stop()

    # Robot control
    @rpyc.exposed
    def pwm(self, left: int, right: int):
        self.context.driver.move(right, -left)

    @rpyc.exposed
    def stop(self):
        self.context.driver.stop()
```

The desktop application can invoke the methods defined in the service using a proxy object after establishing a connection. A wrapper class is defined, which contains a property returning the proxy object member:

```
@property
def remote(self):
    return self.conn.root
```

A connection is established using the following call:

```
self.conn = rpyc.connect(host, port, service=self)
```

Afterwards, remote methods are invoked directly by their name, on the proxy object:

```
self.remote.pwm(left, right)
```

Each of these method calls are wrapped in a Qt slot, to be invoked from anywhere within the application and integrate with the other components.

```
@QtCore.Slot(int, int)
def pwm(self, left, right):
    self.remote.pwm(left, right)
```

### *4.2. Desktop interface windows*

An important requirement of the desktop application is to provide a graphical interface in order to visualize various aspects of the control strategy and to execute manual remote control when necessary. The interface features two windows: first there is a Connect modal dialog (Figure 5) which blocks the rest of the interface and allows the user to connect to the RPyC server running on the robot.

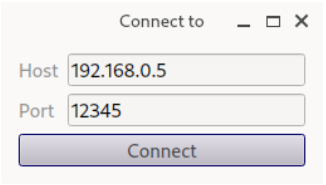


*Figure 5 Connect window*

The Connect window contains two text inputs, for the IP address of the Raspberry Pi and the port that the application process is listening on. This pair is used in establishing the connection once the button press signal is emitted.

The other window, named the *Follower Main Window*, allows the control loop to begin once the user clicks on the *Start Stream* button. The same button breaks the control loop after it has started. Another important element of the window is the viewport object which displays each frame received from the robot, alongside any additional graphics drawn on top of it. These graphics illustrate the visual features extracted from the frame by the visual target detection procedure, or any other image features of interest. An example frame displayed by the Follower Main Window can be seen in Figure 6.

The viewport component receives the frames asynchronously through the signal-slot mechanism and is reusable in other projects based on Qt technologies and the OpenCV framework. It acts as a bridge between the data types used in Computer Vision (n-dimensional arrays) and the image formats being displayed by the Qt graphical user interfaces (`QImage`), doing the conversion within the `convert_cv_qt` instance method. The respective method's source code draws inspiration from an idea

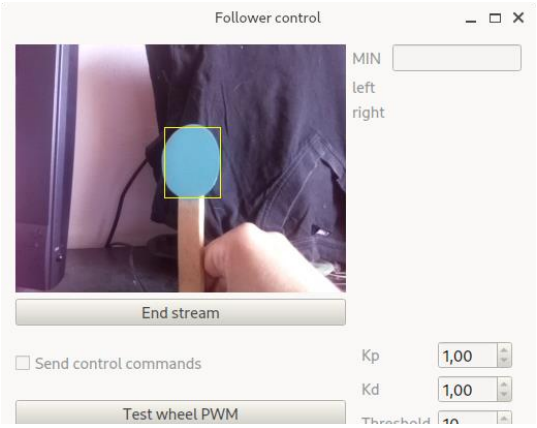


*Figure 6 Follower Main Window*

published by Phil Birch on the *imagetracking* website[23]. Surprisingly, a Label component from the Qt Widgets sub-framework offers an API to display images from raw binary data.

```
class CvLabel(QLabel):
    @Slot(ndarray)
    def set_img(self, cv_img):
        pxm = self.convert_cv_qt(cv_img)
        self.setPixmap(pxm)
```

The control process is automated and does not require user intervention. Using the signal-slot mechanism inherent to Qt the n-dimensional array is passed onto the Worker object responsible for implementing the necessary image processing and feature extraction procedure. Each frame is supplied asynchronously as the object's slot is called. The worker, in turn, emits a signal with the control values to be transmitted remotely, once the visual procedure is finished.

```
class VisionWorker(QtCore.QObject):
    display_frame = QtCore.Signal(np.ndarray)
    pwm = QtCore.Signal(int, int)

    @QtCore.Slot(np.ndarray)
    def process_frame(self, img):
        # process 'img' frame.. (source code snipped)
```

This approach using worker objects for the vision-based control opens up future development of the application into a more flexible framework for testing different procedures and algorithms. A Strategy design pattern can be implemented in the structure of the application, to allow dynamically choosing the procedure used stright from the user interface.

## 5. Conclusions and future directions

The implementation structure and the application architectures described in this paper achieve a de-coupling between the high-level control strategy and the mobile robot agents executing it. The advantages of this platform are faster development-testing iterations and easier debugging of visual procedures and controllers defined, as they are implemented in a plug-in fashion within the desktop application.

This approach enables testing of various control strategies without changes to the source code for the robot application, or even hardware changes. It allows a cleaner separation of concerns and isolates the development of the physical prototype from the tasks it is expected to perform.

Another advantage gained by off-loading the feature extraction and control components to a desktop workstation is an increased processing power provided by the host computing platform.

The architecture proposed in this paper also bears with it some disadvantages or trade-offs, namely it provides less mobility for the system and less agent autonomy. This setup is ideal in a networked environment and also involves the use of a workstation. Mobile robots that are part of the system are completely remote-controlled and dependant on network connectivity. To increase safety for the mobile robots, a set of failsafe measures must be implemented in the robot, independent of control commands and with higher priority.

There are multiple possible future directions for the system. Developments can be done on both hardware and software level. The mobile robot prototypes can be designed for greater efficiency, having the robot application implemented on a microcontroller running a real-time OS or even as a firmware.

The desktop application can be implemented in a compiled language such as C++ and may expose its user interface via the QML engine.

Regarding network communications, a custom serial protocol can be developed for remote control of the mobile robots, allowing other front-ends such as a mobile app to connect to the robots.

Last but not least, the communications technology can be adapted to Bluetooth or other wireless networks to increase mobility and allow field tests to be done in a much easier manner.

**Acknowledgement**

This research has been funded by the University of Oradea, within the Grants Competition "Scientific Research of Excellence Related to Priority Areas with Capitalization through Technology Transfer: INO - TRANSFER - UO", Project No. 326/21.12.2021.

**References**

[1] P. Corke, *Robotics, Vision and Control - Fundamental algorithms in MATLAB*, 2nd ed. Springer International Publishing AG, 2017.

[2] M. CORNEA and V. O. MIHALCA, “A REVIEW OF SWARMING UNMANNED AERIAL VEHICLES,” *Ann. ORADEA Univ. Fascicle Manag. Technol. Eng.*, vol. Volume XXV, no. 3, 2016, doi: 10.15660/AUOFMTE.2016-3.3270.

[3] R. C. Țarcă, “Virtual and Remote Control Lab Experiment Using Matlab,” *Ann. ORADEA Univ. Fascicle Manag. Technol. Eng.*, vol. XIX (IX), no. 3, pp. 78–81, 2010, doi: 10.15660/auofmte.2010-3.2017.

[4] I. Pasc, L. Csokmai, F. Popențiu-Vlădicescu, and R. C. Țarcă, “Augmented reality used for robot remote control in educational laboratories,” *Appl. Mech. Mater.*, vol. 658, pp. 672–677, 2014, doi: 10.4028/www.scientific.net/AMM.658.672.

[5] V. O. Mihalca, D. M. Anton, R. C. Țarcă, and Ş. Yıldırım, “Simulating Simple Tasks for a Kinematic Model of Differential-drive Mobile Robots,” 2023, doi: 10.1109/EMES58375.2023.10171646.

[6] C. Vancea, V. O. Mihalca, and T. R. Toia-Creț, “Mathematical and Programmatic Design in Matlab: A Rewarding Experience,” *J. Comput. Sci. Control Syst.*, vol. 4, no. 1, pp. 203–207, 2011.

[7] K. S. A. Khuwaja, B. S. Chowdhry, K. F. Khuwaja, V. O. Mihalca, and R. C. Țarcă, “Virtual Reality Based Visualization and Training of a Quadcopter by using RC Remote Control Transmitter,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 444, no. 5, 2018, doi: 10.1088/1757-899X/444/5/052008.

[8] K. S. A. Khuwaja, B. Arif Ali, V. O. Mihalca, and R. C. Țarcă, “Automatic fuel tank monitoring, tracking & theft detection system,” in *MATEC Web of Conferences*, 2018, vol. 184, p. 02011, doi: 10.1051/matecconf/201818402011.

[9] Ş. Yıldırım and E. Arslan, “ODE (Open Dynamics Engine) based stability control algorithm for six legged robot,” *Meas. J. Int. Meas. Confed.*, vol. 124, no. May 2017, pp. 367–377, 2018, doi: 10.1016/j.measurement.2018.03.057.

[10] A. I. Pop, N. Pop, R. Ţarcă, C. Lung, and S. Sabou, “Wheeled Mobile Robot H.I.L. Interface: Quadrature Encoders Emulation With A Low-Cost Dual-Core Microcontroller,” in *2023 17th International Conference on Engineering of Modern Electric Systems (EMES)*, Jun. 2023, pp. 1–4, doi: 10.1109/EMES58375.2023.10171736.

[11] O. Araar and N. Aouf, “Visual servoing of a Quadrotor UAV for autonomous power lines inspection,” *2014 22nd Mediterr. Conf. Control Autom. MED 2014*, no. November 2014, pp. 1418–1424, 2014, doi: 10.1109/MED.2014.6961575.

[12] F. Khattar, F. Luthon, B. Larroque, and F. Dornaika, “Visual localization and servoing for drone use in indoor remote laboratory environment,” *Mach. Vis. Appl.*, vol. 32, no. 1, pp. 1–13, 2021, doi: 10.1007/s00138-020-01161-7.

[13] Ş. Yıldırım and S. Sertaç, “Design of a Mobile Robot to Work in Hospitals and Trajectory Planning Using Proposed Neural Networks Predictors,” in *International Conference on Reliable Systems Engineering (ICoRSE) - 2021*, 2022, pp. 32–45, doi: 10.1007/978-3-030-83368-8_4.

[14] X. Feng and S. A. Velinsky, “Distributed control of a multiple tethered mobile robot system for highway maintenance and construction,” *Comput. Civ. Infrastruct. Eng.*, vol. 12, no. 6, pp. 383–392, 1997, doi: 10.1111/0885-9507.00071.

[15] E. Prassler, A. Ritter, C. Schaeffer, and P. Fiorini, “A short history of cleaning robots,” *Auton. Robots*, vol. 9, no. 3, pp. 211–226, 2000, doi: 10.1023/A:1008974515925.

[16] D. M. Anton, R. C. Milaş, T. Ţicărat, V. O. Mihalca, R. C. Ţarcă, and F. I. Birouaş, “Experimental Testing of UV-C Light Sources Used Disinfection Mobile Robots,” in *2023 17th International Conference on Engineering of Modern Electric Systems (EMES)*, Jun. 2023, pp. 1–4, doi: 10.1109/EMES58375.2023.10171692.

[17] Ş. Yıldırım and B. Ulu, “Deep Learning Based Apples Counting for Yield Forecast Using Proposed Flying Robotic System,” *Sensors*, vol. 23, no. 13, 2023, doi: 10.3390/s23136171.

[18] D. Noje, R. C. Ţarcă, N. Pop, A. O. Moldovan, and O. G. Moldovan, “Automatic System Based on Riesz MV-algebras, for Predictive Maintenance of Bearings of Industrial Equipment Using Temperature Sensors,” in *ICCCC 2022: Intelligent Methods Systems and Applications in Computing, Communications and Control*, 2022, pp. 3–19, doi: 10.1007/978-3-031-16684-6_1.

[19] D. Noje, I. Dzitac, N. Pop, and R. C. Ţarcă, “IoT Devices Signals Processing Based on Shepard Local Approximation Operators Defined in Riesz MV-Algebras,” *Informatica*, vol. 31, no. 1, pp. 131–142, 2020, doi: 10.15388/20-INFOR395.

[20] D. Noje, R. C. Ţarcă, I. Dzitac, and N. Pop, “IoT Devices Signals Processing based on Multi-dimensional Shepard Local Approximation Operators in Riesz MV-algebras,” *Int. J. Comput. Commun. Control*, vol. 14, no. 1, pp. 56–62, 2019, [Online]. Available: https://www.univagora.ro/jour/index.php/ijccc/article/view/3490.

[21] M. Grinberg, “Video Streaming with Flask,” 2014. https://blog.miguelgrinberg.com/post/video-streaming-with-flask (accessed Sep. 27, 2023).

[22] D. Jones, “picamera - Picamera 1.13 Documentation,” 2016. https://picamera.readthedocs.io/en/release-1.13/ (accessed Sep. 02, 2022).

[23] P. Birch, “Displaying Opencv images in PyQT,” 2020. https://imagetracking.org.uk/2020/12/displaying-opencv-images-in-pyqt/ (accessed Oct. 06, 2023).