Integrating ROS with the KIPR Wombat

Itmam Alam, Akos Papp

Technical Secondary College,
Department of Computer Science,
2700 Wiener Neustadt, Austria
E-mail of corresponding author: alam.itmam@student.htlwrn.ac.at

***Abstract*—This paper explores the integration of the Robot Operating System (ROS) with the Wombat Controller, aiming to enhance the controller's capabilities using ROS's modular architecture and extensive library support. The paper delves into the potential advantages, drawbacks, and challenges encountered during the project, offering insights into the complexities of navigation with ROS 2 in the context of Botball competitions. Additionally, it presents findings from an experiment evaluating the performance of the iRobot Create 3 with the navigation system of our library JoeX, demonstrating promising results in terms of efficiency and speed. By addressing usability challenges and introducing libraries like JoeX, this paper aims to enhance robotics competitions while broadening the accessibility of ROS 2 among Botball teams.**

***Index Terms*—ROS 2, mobile robotics, robotic system, autonomous navigation**

I. Introduction

The world of robotics is constantly changing, and with each day, we find innovative solutions that enhance the capabilities of robotic systems. One such innovation that has significantly impacted the field of robotics since its development is the Robot Operating System, commonly referred to as ROS [1]. The Robot Operating System is currently in its second iteration (ROS 2), and for our project, we will specifically utilize the ROS 2 Humble distribution. Throughout this paper, we will refer to ROS 2 as ROS for simplicity.

ROS has emerged as a pioneering framework for the development and control of robot systems. It serves as a flexible and open-source platform that eases the seamless integration of hardware components, software modules, and various other components. Its modular architecture, extensive library support, and a global community of developers have made it the preferred choice for researchers and engineers. ROS allows users to leverage its functionalities to build advanced robotic applications with reduced time and resources.

However, despite the immense potential of ROS, its complexity often poses a challenge for implementation. As a result, no team at Botball has yet reached the goal of integrating ROS with their robot systems. Moreover, there appears to be a lack of knowledge about ROS within the Botball tournament community.

In this paper, we aim to address these challenges by providing an overview of everything that is possible with ROS and showcasing our successful integration of ROS with the KIPR Wombat, a versatile robot controller developed by KIPR and used in the Botball tournament. Our project "JoeX" introduces a wide range of possibilities, enabling control over diverse hardware components and incorporating navigation technologies unprecedented in this competition.

In the subsequent sections, we will discuss the functionalities and advantages of ROS, explore the library we developed for using ROS with the Wombat, outline potential use cases, and address the challenges and considerations that may arise. Ultimately, we hope this paper serves as a catalyst for future developments in this field, inspiring new ideas and approaches in the intersection of ROS and robots.

II. Understanding ROS

ROS, the Robot Operating System, is a flexible framework designed to simplify and standardize robotics development. It provides a comprehensive suite of tools and libraries that facilitate the creation of complex robot systems. By offering a middleware for communication between different components, ROS enables seamless integration and interoperability among various hardware and software modules. In the following sections we will explain the core components of ROS and mention some of the libraries that were used.

A. Architecture

ROS is at its core a message broker, where nodes establish direct connections with one another, enabling peer-to-peer communication within the framework [2]. This communication infrastructure uses TCP/IP for transmission, facilitating reliable data exchange between nodes.

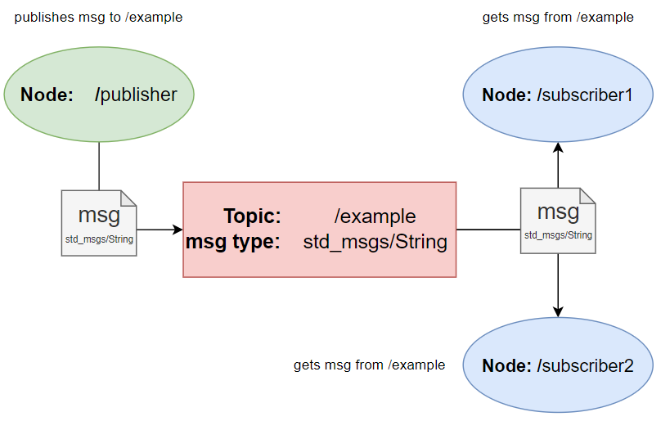


Fig. 1. Publisher and subscriber model in ROS

Nodes are individual programs within a ROS application, each dedicated to performing a specific task. It's a best practice for nodes to focus on a single, well-defined task to ensure modularity and reduce the complexity within the system. Each node has a unique name similar to UNIX paths (e.g.: /node-name) and is implemented using a designated ROS client library like rclcpp or rclpy [3].

Topics serve as communication channels within a ROS application, facilitating interaction between nodes. They are named buses through which nodes exchange messages. Nodes can publish messages to topics to share information or subscribe to topics to receive data from other nodes as shown in figure 1.

Services in ROS establish a communication pattern based on request and response interactions between nodes. Within this model, a client sends a request to a server and awaits a specific computation or action, after which the server provides a response.

Messages are described and defined in *.msg* files. It is best practice to put them in the *msg/* directory of a ROS package [4]. Messages look like this:

```
float32 current_temp
string location
```

Fig. 2. Example of a ROS message

The first word is a data type the second is the field name. In the example above, a message structured like this could be published from a node that measures temperatures. Another node can then subscribe to the topic and read the incoming temperature values.

The use of these communication mechanisms allows for asynchronous and decentralized data exchange, enabling efficient and parallel processing of tasks within the robot system.

B. Toolset and Libraries

ROS comes equipped with a diverse array of tools and libraries, because it is primarily designed to be modular, which allows developers to easily use and integrate third-party components. For instance, the ROS 2 Navigation Stack provides capabilities for path planning, obstacle avoidance, and localization. ROS also offers libraries for sensor data processing, motion control, and perception, such as the Point Cloud Library (PCL) for 3D data processing [5].

The **Nav2** stack serves as a key navigation stack within ROS 2, providing developers with powerful tools for building complex robotic navigation systems. Leveraging ROS 2 as its core middleware, Nav2 integrates crucial components such as action servers, lifecycle nodes, and behavior trees to provide a robust navigation system. Action servers enable the control of long-running tasks, facilitating asynchronous communication with feedback during execution. Lifecycle nodes ensure deterministic behavior during system startup and shutdown, contributing to reliable and predictable operations. The use of behavior trees [6] offers a structured and human-understandable framework for defining complex robotic behaviors.

The stack's design emphasizes modularity and ease of development, exemplified by the integration of modular node plugins.

The **tf2** library stands as a fundamental component within ROS 2, providing developers with a tool for managing coordinate frames in robotic systems. Tf2 enables users to monitor and track multiple coordinate frames over time, maintaining a relationship between them. This allows users to perform seamless transformations of points, vectors, and other entities between any two coordinate frames at specified points in time. ROS specified several naming conventions for these frames, e.g. 'map', 'odom' and 'base-link' [7]. One of the key strengths of tf2 is that information regarding the coordinate frames of a robot is accessible to all ROS 2 components on any computer within the system.

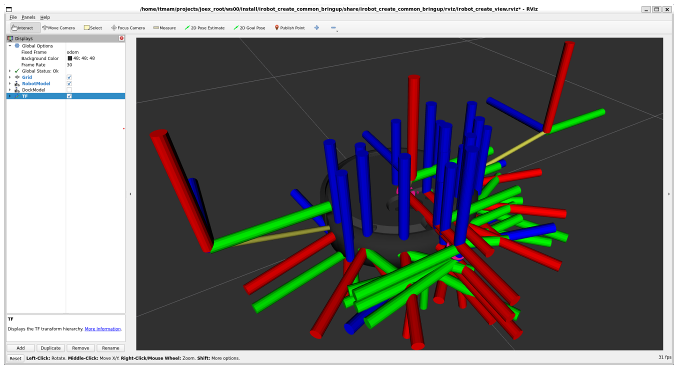


Fig. 3. Transforms of the iRobot Create 3

C. Simulation Tools

Simulation tools play a vital role in the development and testing of robotic systems. They provide a controlled virtual environment for evaluating and refining various functionalities, ensuring the robustness of robot behaviors.

Gazebo plays a crucial role in simulating robots in ROS. It provides a rich set of tools, allowing developers to simulate various aspects of robots. From modeling physical interactions to simulating sensor outputs, Gazebo enables a comprehensive understanding of how robots operate in diverse scenarios. Robots can navigate and interact within the simulated environment, allowing developers to assess the performance and reliability of navigation algorithms in a controlled setting.

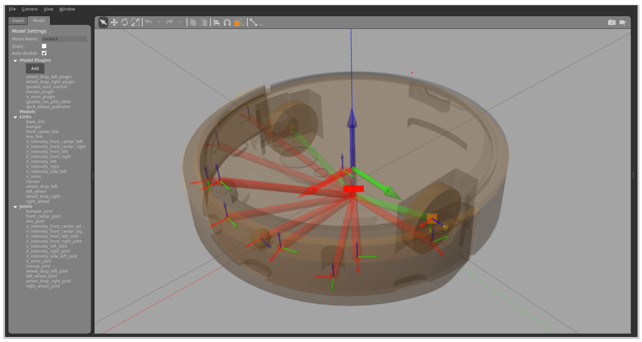
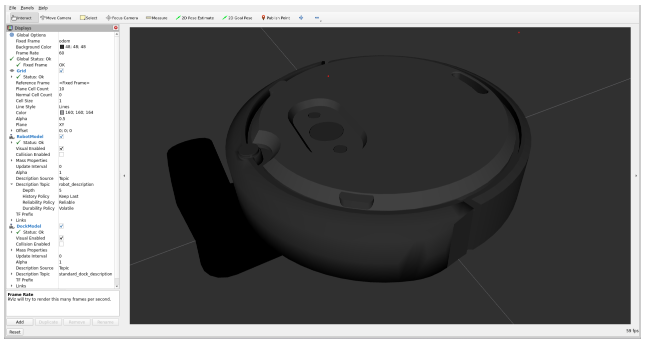


Fig. 4. iRobot Create 3 simulation in Gazebo

RViz2 emerges as a powerful visualization tool, offering insights into the inner workings and data streams of robotic systems. This includes sensor data, odometry, and the robot's perception of its surroundings. The real-time visualization enhances developers' understanding of how the robot perceives and interprets its environment. RViz2 seamlessly integrates with Gazebo, creating a unified development and debugging environment. This integration ensures that developers can visualize the simulated robot's data within RViz2 while testing in the Gazebo simulation environment.



```
Global Options
  Fixed Frame             odom
  Background Color        48; 48; 48
  Frame Rate              60
✓ Global Status: Ok
  ✓ Fixed Frame           OK
Grid                      ✓
  ✓ Status: Ok
  Reference Frame         <Fixed Frame>
  Plane Cell Count        10
  Normal Cell Count       0
  Cell Size               1
  Line Style              Lines
  Color                   160; 160; 164
  Alpha                   0.5
  Plane                   XY
  Offset                  0; 0; 0
RobotModel                ✓
  ✓ Status: Ok
  Visual Enabled          ✓
  Collision Enabled
  Mass Properties
  Update Interval         0
  Alpha                   1
  Description Source      Topic
  Description Topic       robot_description
    Depth                 5
    History Policy        Keep Last
    Reliability Policy    Reliable
    Durability Policy     Volatile
  TF Prefix
  Links
DockModel                 ✓
  ✓ Status: Ok
  Visual Enabled          ✓
  Collision Enabled
  Mass Properties
  Update Interval         0
  Alpha                   1
  Description Source      Topic
  Description Topic       standard_dock_description
  TF Prefix
  Links

Frame Rate
RViz will try to render this many frames per second.
```

Fig. 5. iRobot Create 3 visualization in RViz2

III. Robot Communication

This section explores ROS's role in facilitating communication within the Wombat platform throughout the Botball run phases: calibration, game start, and continuous run. Leveraging ROS's publisher-subscriber model via topics, services, and actions, the framework allows for dynamic data exchange and real-time communication among robots.

During calibration, ROS facilitates dynamic communication as robots can publish sensor calibration data on dedicated topics, allowing multiple robots to compare and correct values as needed.

At the start of a Botball game, ROS communication plays a pivotal role in synchronizing robotic behaviors. ROS topics can broadcast important game-related data like start signals and initial positions, ensuring synchronization among robots.

During a Botball run, real-time communication is essential for collaboration. ROS topics enable robots to exchange task updates, ensuring a shared game state understanding. Additionally, ROS facilitates dynamic service requests of robots, while ROS actions coordinate long-running tasks like complex navigation maneuvers.

These scenarios underscore the flexibility of ROS, presenting possible communication usages that cater to dynamic robot interactions.

IV. JoeX Library

For the integration of ROS with the Wombat platform, a Python library called JoeX was developed. Serving as a bridge between ROS and the Wombat, JoeX manages communication and control over essential components like motors, servos, and sensors. This was accomplished using the libwallaby library [8], developed by KIPR.

A. Structure

JoeX serves as a crucial intermediary in enabling seamless communication between ROS and the Wombat controller. Developed using the ROS client library for Python, it offers an interface for controlling all components used in the Botball tournament. Additionally, JoeX plays a vital role in managing the navigation aspects of the platform.

The libwallaby serves as a foundational library for controlling components such as servos, motors, and various sensors with the Wombat. We adapted libwallaby to be compatible with any ROS node, enhancing its versatility and interoperability within the ROS ecosystem. Our system architecture includes a C++ node responsible for handling major calculations related to navigation and driving. This node communicates with the JoeX Python library using ROS topics, services, and actions. This approach addresses performance concerns associated with Python due to the overhead of interpretation during runtime, which can result in slower execution compared to compiled languages like C++.

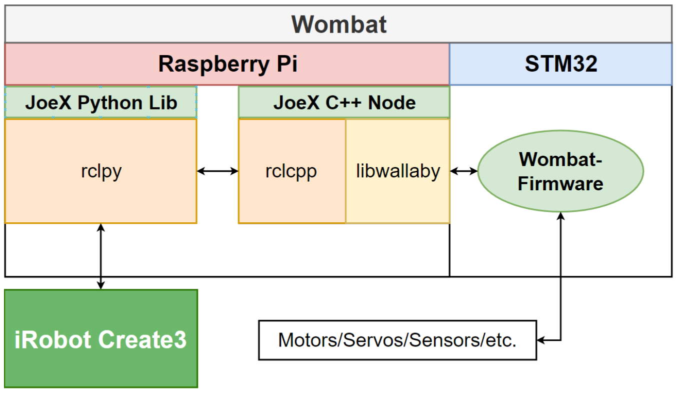


Fig. 6. System architecture of JoeX

By leveraging this architecture, we eliminate the need for compiling code when programming the robots, allowing for immediate implementation of changes without waiting for compilation. Our programming approach streamlines the development process and maximizes efficiency, with Python code interfacing seamlessly with JoeX, which in turn communicates with the C++ node in the background.

B. Configuration

The library parses a YAML [9] file containing all configuration values of the robot, ranging from port specifications of robot components to PID values of motors. Users have the flexibility to configure various aspects of the robot's functionality, including mapping specific ports of the wombat to designated names for streamlined programming, as shown in figure 7.

C. iRobot Create 3

The iRobot Create 3 emerges as a prominent addition to the supported components of JoeX. It can be utilized in this year's Botball tournament, providing participants with an advanced robotics platform for competition purposes. The robot is equipped with a multi-zone bumper featuring seven pairs

```
servos:
  - name: arm
    port: 0
    start_position: 0.2617994
    update_rate: 100
    ticks_per_rad: 732.311681902
    max_velocity: 600
motors:
  - name: gripper
    port: 0
    ticks_per_rad: 1000
    p: 0.1
    i: 0.01
    i_max: 5
    d: 0.1
```

Fig. 7. Example YAML configuration file used by JoeX

of infrared proximity sensors at the front, enabling obstacle detection. Additionally, four cliff sensors located at the bottom ensure the robot remains on stable terrain. The integration of two wheels with current sensors and encoders enables precise motion control and odometry estimation. Furthermore, an optical odometry sensor, combined with the IMU, provides accurate pose estimation by fusing data from wheel encoders and IMU readings [10].

The Create 3 supports odometry and IMU sensor-fusion out of the box, ensuring exceptional accuracy compared to the older Create 2. Moreover, the Create 3 handles all the calculations for driving itself, which reduces the load on the Raspberry Pi of the Wombat.

With its integration via JoeX, we can leverage the rich capabilities of the Create 3 through ROS. In addition, the platform's support for ROS enables the use of autonomous behaviors, enhancing the versatility of the robot.

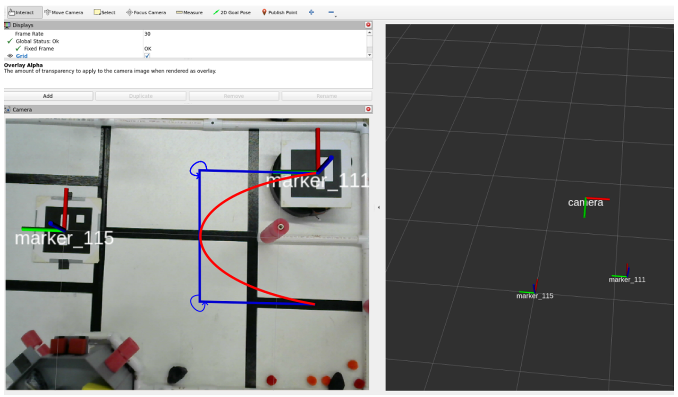
D. Navigation Algorithm

For navigating a robot using the Wombat, the Nav2 framework stands out as one of the most powerful navigation solutions available in ROS 2. With its customization options and advanced features such as collision avoidance and path planning, Nav2 offers many features for navigating complex environments. However, the abundance of nodes and features within Nav2 can increase system complexity. Despite its capabilities, many of the advanced features offered by Nav2 are not essential in the Botball competition, leading us to opt out of utilizing Nav2 in favor of a more efficient approach with JoeX.

Our system uses a polynomial curve for path planning, allowing navigation through multiple points with the option for directional control at the final point. By calculating the maximum velocity based on parameters such as curvature and robot specifications, we ensure precise and efficient movement along the planned path. This approach eliminates the need for tedious calibration processes, as parameters are measured once and applied universally.

E. Experiment

We conducted an experiment to evaluate the performance of the iRobot Create 3 in conjunction with the JoeX navigation system. Given that both JoeX and the built-in Create 3 navigation action performed equally well in terms of straight driving due to the same speed limit constraint, we shifted our attention to driving curves. It is important to note that when driving curves with the Create 3, the built-in navigate_to_position action generates a path that involves driving straight followed by turns.



Frame Rate 30
Global Status: Ok
Fixed Frame OK

Overlay Alpha
The amount of transparency to apply to the camera image when rendered as overlay.

Fig. 8. Test setup in RViz, left: camera stream, right: ArUco marker transforms

Figure 8 illustrates two distinct paths highlighted in blue and red. The blue path represents the trajectory generated by the navigate_to_position action, which involves driving straight, turning 90 degrees, driving straight again, making another 90-degree turn, and finally continuing straight. This method, while being accurate, is time-consuming due to the sequence of straight segments and turns required. In contrast, the red path represents the trajectory achieved by the polynomial curve. Unlike the navigate_to_position action, JoeX enables the robot to follow a continuous curve, eliminating the need for multiple segments and turns. This approach offers fast driving, enhancing overall efficiency during maneuvers.

For the test setup, a camera was positioned above a Botball game table capturing images, as shown in figure 8, that were published on a ROS topic. Additionally, essential camera parameters, such as an intrinsic matrix, were published via a camera_info topic for accurate marker detection using the ArUco OpenCV ROS node [11]. This node leverages the camera parameters to detect the position of ArUco markers [12] within the images. Subsequently, a transform from the camera to the detected ArUco marker is broadcasted by the ArUco node.

To measure the position of the robot accurately, we employed the tf2 library to calculate the transformation between two specific ArUco markers. This process allowed us to precisely determine the relative positions of these markers.

We conducted the curve drive tests for both JoeX and the Create 3 navigate_to_position action. It consists of driving a 120cm long curve forwards and backwards, five times in a row. Despite slight variations in accuracy, both approaches exhibited consistent performance, with deviations of approximately ±0.8cm from the start position across all trials. However, the significant contrast in completion times was evident, with JoeX completing the test in 66 seconds, while the built-in navigation required 139 seconds. This difference underscores the efficiency of the polynomial curve navigation, enabling faster traversal along the curve compared to the built-in driving method.

Our navigation algorithm offers significant improvements in accuracy, particularly in driving smooth and complex paths. Additionally, our system's reliance on odometry enables automatic correction, ensuring consistent performance even when external factors affect the robot's movement.

The consistent accuracy observed in both approaches also suggests reliable odometry performance of the robot, although minor discrepancies may arise due to factors such as camera precision.

V. Challenges

In this section, we delve into the downsides and difficulties we encountered during the implementation of our ROS node. These challenges highlight areas where ROS may present obstacles or limitations, impacting the efficiency and effectiveness of robotic systems.

A. Complexity of ROS

The comprehensive features and functionalities of ROS contribute to its complexity, posing challenges for users in terms of learning and efficiently utilizing the framework. A deep understanding of nodes, topics, services, and actions is essential for troubleshooting and debugging within a ROS system. Managing a project within this intricate ecosystem requires meticulous attention to detail and expertise, which can be daunting for novice users.

B. Difficulty in installation

Installing ROS proved to be a significant hurdle, primarily due to limitations imposed by the hardware and the preferred operating system environment. While Ubuntu is the recommended OS for ROS installation, its resource-intensive nature and substantial storage requirements render it impractical for deployment on the Wombat. Moreover, containerization solutions like Docker present their own set of challenges, with trade-offs between resource efficiency and development convenience. The choice between an Ubuntu-based container with comprehensive development tools and the official minimal ROS image lacking essential utilities underscores the inherent complexities of ROS deployment and development. Balancing these considerations is crucial for optimizing the integration of ROS within robotic systems like the Wombat.

Notably, KIPR pursued these optimizations to enable the integration of the iRobot Create 3 into this year's Botball competition. Recognizing the importance of leveraging ROS for advanced robotics functionalities, KIPR sought to overcome the challenges associated with ROS deployment to harness the capabilities of the Create 3 platform. KIPR opted for containerization using a custom Ubuntu-based image with Podman for the Wombat platform.

Conclusion

While ROS remains a powerful framework for robotics development, its complexity poses significant challenges for many teams. Our implementation of ROS within the Wombat platform revealed the intricacies involved in using its features. However, our focus on easing ROS deployment, particularly through the development of the JoeX library, underscores the importance of overcoming these challenges. The performance evaluation conducted with the iRobot Create 3 and JoeX navigation system showcased promising results, with JoeX demonstrating faster and more accurate navigation compared to the built-in Create 3 driving action. Looking ahead, initiatives aimed at simplifying ROS 2 and making it more accessible can broaden its reach among Botball teams. By addressing usability challenges and leveraging innovations like JoeX, we can unlock the full potential of ROS and elevate robotics competitions to new levels.

Acknowledgment

The authors would like to thank Dr. Michael Stifter, the teachers of the HTBLuVA Wiener Neustadt, and the members of robo4you for their kind support in producing this publication. Special thanks go to Dr. Markus Bader from the TU Wien for helping us get into the world of the Robot Operating System.

References

[1] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng. "ROS: an open-source Robot Operating System", Jan. 2009. http://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf (accessed Mar. 4, 2024)

[2] Open Source Robotics Foundation. "ROS/Introduction". ROS Wiki. https://wiki.ros.org/ROS/Introduction#What_is_ROS.3F (accessed Dec. 12, 2023).

[3] Open Source Robotics Foundation. "Client libraries". ROS 2 Documentation: Humble. https://docs.ros.org/en/humble/Concepts/Basic/About-Client-Libraries.html (accessed Dec. 12, 2023).

[4] Open Source Robotics Foundation. "Interfaces". ROS 2 Documentation: Humble. https://docs.ros.org/en/humble/Concepts/Basic/About-Interfaces.html#messages (accessed Dec. 13, 2023).

[5] Open Source Robotics Foundation. "PCL". ROS Wiki. https://wiki.ros.org/pcl (accessed Dec. 22, 2023).

[6] Open Navigation. "Nav2 Behavior Trees". Nav2 Documentation. https://navigation.ros.org/behavior_trees/index.html (accessed Dec. 22, 2023).

[7] W. Meeussen. "REP 105 Coordinate Frames for Mobile Platforms". ROS Reps. https://www.ros.org/reps/rep-0105.html#id13 (accessed Jan. 2, 2024).

[8] B. McDorman, N. Zarman, Z. Sasser, T. Corbly and E. Harrington. "libwallaby Documentation". https://www.kipr.org/doc/index.html (accessed Jan. 24, 2024).

[9] O. Ben-Kiki, C. Evans, I. döt Net. "YAML Ain't Markup Language (YAML™) Version 1.2". (accessed Mar. 5, 2024)

[10] iRobot Corporation, "Overview - Create 3 Docs". Create 3 Docs. https://iroboteducation.github.io/create3_docs/hw/overview/ (accessed Jan. 31, 2024).

[11] B. Sowa, Fictionlab. "aruco_opencv". ROS Index https://index.ros.org/p/aruco_opencv/ (accessed Mar. 8, 2024)

[12] F. Romero-Ramirez, R. Muñoz-Salinas, R. Medina-Carnicer. "Speeded Up Detection of Squared Fiducial Markers", Image and Vision Computing, 2018. https://doi.org/10.1016/j.imavis.2018.05.004 (accessed Mar. 8, 2024)