Extending the Wombat Controller with an Arduino Microcontroller

Jovan Vujičić
Netzwerktechnik
HTL Villach
Villach, Austria
jovan.vujicic@edu.htl-villach.at

Franziska Ebner
Netzwerktechnik
HTL Villach
Villach, Austria
franziska.ebner@edu.htl-villach.at

Abstract*—In this paper we will explain how we integrated an Arduino microcontroller into our Wombat controller. The main reason we did that is to expand the number of usable input and output ports for our robots. The combination of the modular and very robust Wombat with an Arduino microcontroller allows us to overcome the hardware limitations of just the Wombat or just a standard microcontroller. By setting up a dependable communication protocol and designing a for our purposes useful hardware setup, we can use more sensors, servos, and motors on our robots. The result is an efficient and not too expensive solution for projects that require a higher number of controllable channels.

I. Introduction

Robotic systems have evolved rapidly in recent years, and with this development comes a growing need for control architectures that can adapt to increasingly complex hardware setups. Many projects today rely on a wide range of sensors and peripheral modules, all of which must be coordinated reliably and in real time. While dedicated controllers such as the Wombat offer a solid and durable foundation for mobile robots, they are not designed to handle an unlimited number of input and output channels. This becomes a limiting factor as soon as a project requires more than the built-in hardware can support. On the other hand, microcontroller platforms like the Arduino provide a large number of accessible I/O pins and a flexible programming environment, but they lack the robustness and structural reliability needed for more demanding or long-running robotic applications.

This gap between flexibility and durability is a common challenge in robotics. Developers often find themselves forced to choose between a stable but limited controller, or a versatile but less reliable microcontroller. In many cases, neither option alone is ideal. This paper addresses that problem by combining both systems into a single, integrated solution. By connecting an Arduino microcontroller to the Wombat controller, we aim to merge the strengths of each platform: the Wombat's dependable hardware and modular design with the Arduino's expandability and ease of customization.

To make this integration practical, we designed a communication protocol that ensures stable data exchange between the two devices and developed a hardware setup tailored to the needs of our robots. This allows us to add additional sensors, servos, and motors without overloading the Wombat or compromising system performance. The result is a cost-effective and scalable approach that enables more complex robotic configurations without requiring specialized or expensive hardware.

Our work demonstrates that combining established controller platforms can be a viable alternative to replacing them entirely. By extending the Wombat with an Arduino, we create a flexible and reliable control system that supports a significantly higher number of channels, making it suitable for a wide range of robotics projects.

II. State of the Art / Literature Riview

In many robotics projects, the built-in controller of a robot is not enough when the system needs to handle a large number of sensors, motors, or servos. Controllers like the Wombat are known for being reliable and easy to use, but they offer only a limited number of input and output ports. Because of this, many teams expand their robots by adding a second controller, usually a microcontroller such as an Arduino. Arduino boards are very common in educational and hobby robotics because they provide many I/O pins, are simple to program, and come with a wide range of libraries that support different sensors and actuators. [1]

There are already systems that follow this idea of extending a main controller with an additional microcontroller. One example is the *Serial Wombat* project, which provides small expansion chips that communicate with Arduino or Raspberry Pi devices through a serial connection [2]. These chips are designed to add more inputs and outputs and come with their own Arduino libraries that make the communication easier. This shows that using a serial link to expand a controller is a well-established method. [3]

There are also examples of the Wombat being combined with other platforms. For instance, researchers have connected the Wombat to the Robot Operating System (ROS) to add more software features and hardware options. [4] This demonstrates that the Wombat is often used together with additional systems when a project requires more flexibility than the built-in hardware can provide.

Most of these hybrid setups rely on USB or serial communication. To make this work smoothly, both devices need a clear communication protocol so they can understand each other's commands. Many projects report that they had to write their own libraries or message structures because the default firmware did not support all the functions they needed.

Overall, the current state of the art shows that combining a stable main controller with a flexible microcontroller is a common and effective way to expand a robot's capabilities. However, there is very little documentation about connecting an Arduino directly to the Wombat. Our project builds on the ideas found in existing systems and applies them to the Wombat by creating a custom communication protocol and hardware setup that allows the two controllers to work together reliably.

III. Concept / Design

The basic idea behind our system is to use a master-slave setup so we can expand the limited number of ports on the KIPR controller and also run two processes at the same time without slowing the robot down. In our design, the KIPR controller acts as the master, because it is the main brain of the robot and runs the main program. The Arduino Mega2560, together with the L293D motor driver shield, takes the role of the slave. [5] From here on, we simply call it "the Arduino." We chose the Arduino Mega because it has many pins and works well with shields, and we noticed during testing that it can control servos more precisely than the Wombat alone.

The idea is that the Wombat handles the high-level logic, like deciding what the robot should do next, while the Arduino takes care of the low-level tasks, like moving motors or reading sensors. This way, the workload is split between the two controllers, and the robot can do more things at once without running into hardware limits.

Communication between the two devices happens through the serial port at 9600 baud. The KIPR controller sends commands to the Arduino in a simple text-based format. For example, if motor 1 needs to move at speed 300, the Wombat sends the command "M:1:300". The Arduino receives this command, understands what it means, and then moves the correct motor at the correct speed. We designed the commands to be easy to read and easy to debug, because that helped us a lot during development.

The Arduino can also send information back to the Wombat. For example, if a sensor is connected to the Arduino, the Wombat can request the value by sending a command like "S:2". The Arduino then reads sensor 2 and sends the value back through the serial connection. The Wombat can then use this value in its program, for example to avoid obstacles or follow a line. This two-way communication makes the whole system much more flexible.

We also had to think about how to organize the hardware. The motor driver shield sits on top of the Arduino and allows us to control multiple DC motors and servos. To ensure stable operation and prevent power drops when driving multiple motors, we integrated an independent external power supply specifically for the motor driver shield, using a KIPR Controller battery. This battery is mounted on the top of the counterweight. We connected all additional sensors directly to the Arduino pins. This keeps the wiring simple and avoids overloading the Wombat's ports. The Wombat only needs one USB cable to communicate with the Arduino, which makes the setup clean and easy to maintain.

Overall, the concept behind our design is to let each controller do what it is best at. The Wombat stays the reliable main controller, while the Arduino adds flexibility and extra ports. By combining both, we created a system that can handle more motors, servos, and sensors than either controller could manage on its own.

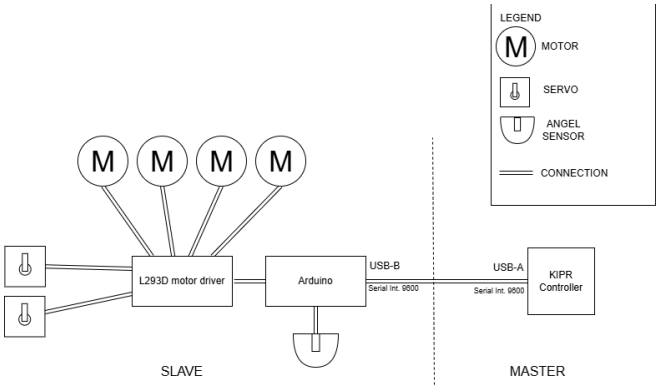

Fig. 1: System architecture of the master-slave communication between the KIPR Controller and Arduino Mega 2560 vias USB serial interface.

IV. Implementation

At the beginning, we connected the Arduino microcontroller to the robot using a standard USB-A cable. This was the easiest way to link the two systems, because the Wombat automatically recognizes the Arduino as a serial device. The serial interface then became the main communication channel between the two controllers. Every command, sensor value, or message travels through this connection. Whenever the robot needs to control something that is connected to the Arduino, it sends a command through the serial port. The Arduino reads the incoming text, checks what the command means, and then performs the action right away. This makes the system react quickly and keeps the communication simple.

We also made sure that the communication works in the opposite direction. If the Wombat needs a sensor value that is connected to the Arduino, it sends a request through the same serial interface. The Arduino then reads the sensor, prepares the data, and sends it back to the Wombat. This allows the robot to use many more sensors than before and still get the information fast enough for real-time decisions. Because of this two-way communication, the robot can understand its surroundings better and react more precisely, for example when following a line or avoiding obstacles.

While building the system, we ran into a problem. The firmware version on our Wombat did not include any built-in functions to directly communicate with an Arduino. At first, this made it difficult to send and receive data in a structured way. To solve this issue, we decided to write our own small software library. This library defines all the commands we use, such as how to control motors, how to read sensors, and how to format the messages. To provide a better understanding of our software architecture, the core function of our custom library include:

- **serial_begin(port, baudrate):** Initializes the serial connection to the specified port at the given speed.

- **serial_send(command):** Transmits a formatted text command to the Arduino.
- **serial_read_string(buffer, max_length):** Reads incoming data into a local buffer while preventing memory overflows.
- **serial_end():** Safely closes the serial port and releases the connection.

It also makes sure that both the Wombat and the Arduino understand the same structure, so there are no misunderstandings between the two devices.

We also added some basic error handling to our library. For example, if the Arduino receives a command that does not match the expected format, it ignores it instead of crashing. This was important because during testing we sometimes sent wrong commands by accident. With the error handling in place, the system became much more stable.

Another part of the implementation was testing different baud rates. We started with 9600 because it is very stable and easy to debug, but we also tried faster speeds. In the end, we stayed with 9600 because it was reliable enough for our purposes and did not cause any communication errors.

The implementation phase taught us a lot about how two controllers can communicate with each other. By writing our own library and testing different setups, we managed to create a stable and flexible system that allows the Wombat and the Arduino to work together smoothly.

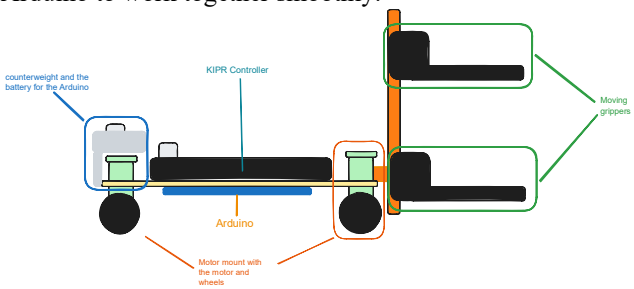

Fig. 2: Side view of the robot prototype. The stacked configuration of the KIPR Controller and the Arduino unit ensures a compact and modular design.

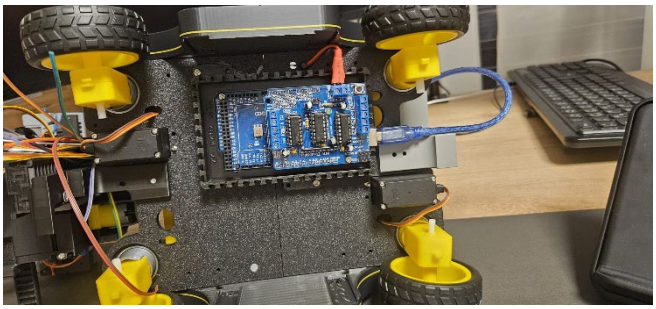

Fig. 3: Detailed view of the bottom side of the robot holding the Arduino.

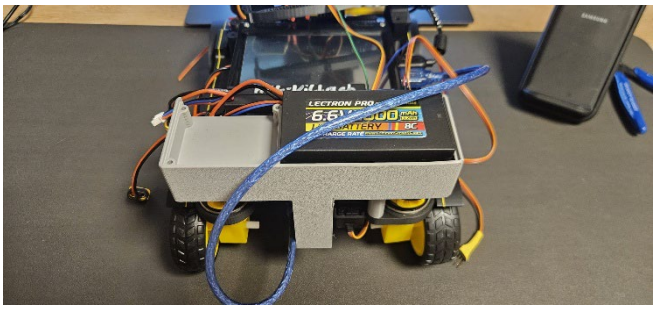

Fig. 4: Detailed view of the counterweight holding the battery for the Arduino.

V. Results / Conclusion

Through this project we were able to show that connecting an Arduino to the Wombat controller is a simple and effective way to get more input and output options for our robots. After setting up our own communication protocol and writing a small library, the two systems could talk to each other without problems. This made it possible for us to control more motors and servos and read extra sensors that the Wombat alone could not handle.

While testing, we noticed that the Arduino is especially good at tasks that need very precise timing, like controlling servos. The Wombat, on the other hand, stayed the main controller and took care of the bigger decisions and the robot's main program. Because both controllers shared the work, the robot reacted faster and could use more hardware at the same time.

Another thing we learned is that expanding a robot does not always have to be expensive. Instead of buying new hardware or replacing the Wombat, we managed to improve the system with an Arduino Mega and a motor driver shield, which are both affordable and easy to use. This makes our solution great for school projects or competitions where you don't have a huge budget.

We also realized how important it is to have a clear communication structure between two controllers. At first, it seemed complicated, but once we defined simple commands and stuck to them, everything worked very reliably. This experience helped us understand how different systems can work together if the communication is well planned.

In the end, our project shows that combining the Wombat with an Arduino is a practical and flexible way to overcome hardware limits. The setup can easily be reused or expanded for future robots, and it gives us many more possibilities for building more advanced and creative robotic systems.

References

[1] Arduino. 2024. Arduino Mega 2560 Rev3 Documentation [https://docs.arduino.cc/hardware/mega-2560]

[2] BroadwellConsultingInc. 2024. SerialWombatArdLib [https://broadwellconsultinginc.github.io/SerialWombat/]

[3] KISS Institute for Practical Robotics (KIPR). 2023. KIPR Robot Controller Documentation (libkipr / libwallaby [libkipr: libwallaby]
[4] ROS. 2009. ROS-Robot Operating System [ROS: Home]

[5] Adafruit Industries. 2024. Adafruit Motor Shield [Downloads | Adafruit Motor Shield | Adafruit Learning System]