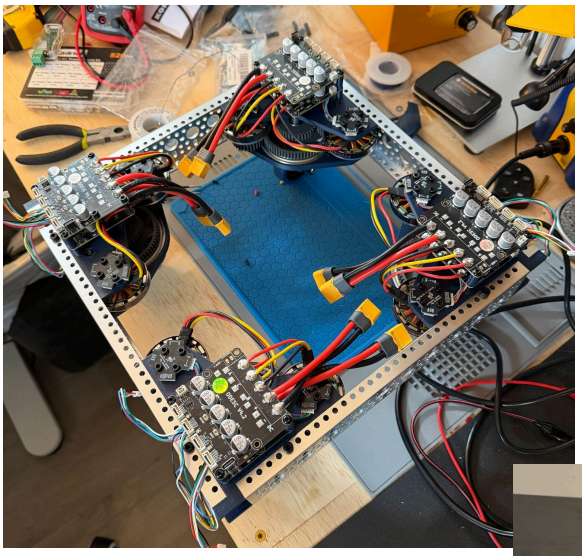
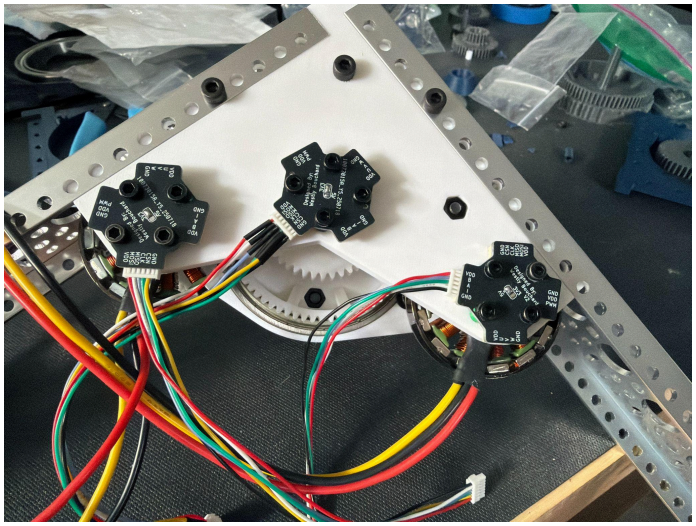
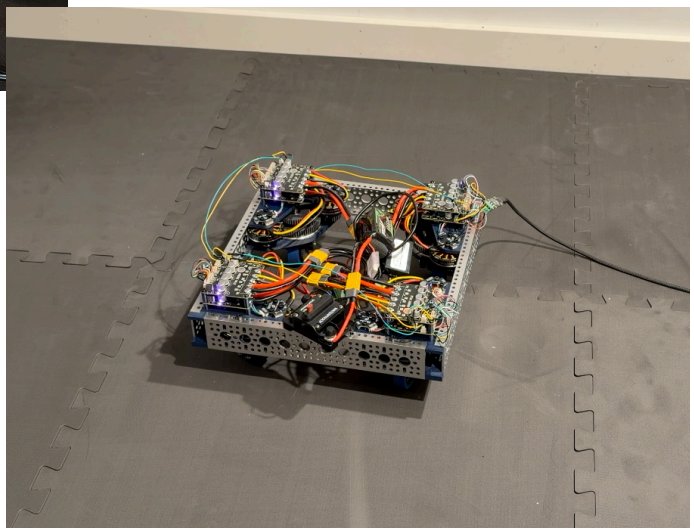
Systems Integration and Functionality Testing for the Rival Robotics Competition







[Demo Video Link]

Westly Bouchard (westly_bouchard@mines.edu)
Course: Prototyping for Concept Validation (EDNS 498-A)
Instructor: Dr. Michael Sheppard
Date: November 5, 2025

Background and Relevance

The Rival Robotics Competition (Rival) is a yearly robotics challenge that tasks teams to construct a robot to play a unique game. The 2025 Rival game, titled Doomsday, is played by collecting badminton birdies and scoring them by placing them on platforms of various heights. This game is played on a 12ft by 12ft field and so, a core component of any competitive robot is a drivebase that can transport a scoring mechanism between the birdie collection points, and the scoring areas.

Figure 1: Doomsday game piece

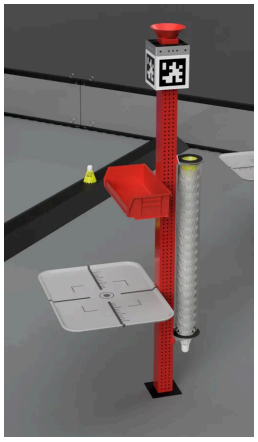


Figure 2: Scoring Station

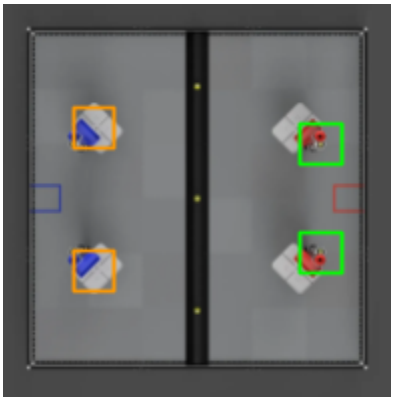


Figure 3: Pickup (green) and scoring (orange) locations

The assembly, integration, and testing of this drivebase is the focus of this report. Prior to this sprint, significant prototyping and validation work had taken place with regards to the individual components, sensors, and subsystems through the construction and testing of a single swerve module; the primary, repeated element of the drivebase.

Of note for this report is that this control system relies on a central computer to manage system state, receive input from the operator, send corresponding movement commands to motor controllers, and to gather and report telemetry data. Up until this point, the development and testing of this system had taken place on my desktop computer as this allowed for easy modification and debugging of the various system functions.

However, attaching a full size desktop computer to this small, 12”x12” robot frame is laughably impractical, and so the constraints of this competition required that the control system also be able to run on a small single board computer (SBC), in this case a raspberry pi.

Functional Requirements

In order for this system to properly support competitive gameplay in the Rival competition, it must fulfil the following core functionality requirements:

1. The system must be able to boot, start the primary control logic, and shutdown without intervention from the operator. Rather than manually establishing communication with hardware, starting the control system, and connecting the controller over the command line, the system must be able to perform these tasks on its own, as a raspberry pi does not have a screen or keyboard with which one could issue these commands.
2. The system must be able to reliably translate movement inputs from the operator to motor commands in real time. The concern here is that the raspberry pi is much less powerful than the development computer previously used, and so it is important to validate that the system can continue to operate as expected under these constraints.
3. The assembled drivebase must intrude minimally into the interior of the robot frame, as this space is needed for the scoring assembly as well as other elements of the control system. This is important because none of the electronics or wiring was modeled when the design was created in CAD, as that focused purely on the mechanical linkages of the system.

Materials and Assembly Process

The drive base is primarily constructed out of off the shelf aluminum extrusion from goBUILDA, and 3D printed parts. It also utilizes standard metric fasteners, bearings, and wiring. The control system is composed primarily of:

- Raspberry Pi 5 running Ubuntu Linux
- CAN to USB adapter (allows the pi to send and receive data over the CAN bus)
- ODesc V4.2 motor controllers (ODrive knockoff boards, running a slightly modified version of the open source ODrive Firmware)
- AS5047p magnetic encoders (on custom breakout boards)
- 5010 360KV drone motors
- 24V (6s) LiPo Battery

The primary challenge associated with the assembly process was certainly the wiring and placement of the electronic components. The manufacturing and assembly of the swerve modules was fairly straightforward thanks to the mountain of previous prototyping work done.

Summary of Testing

The original concept for this project utilized large connectors in between the power distribution board, motor controllers, and motors. However, the initial assembly with this design showed that the connectors and extra lengths of wire required for modularity simply took up too much space, and would not have met the functional requirements defined above.

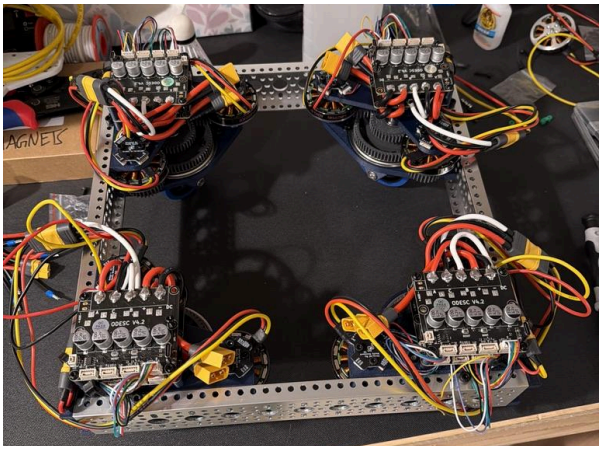


Figure 4: Bulky connectors and wiring

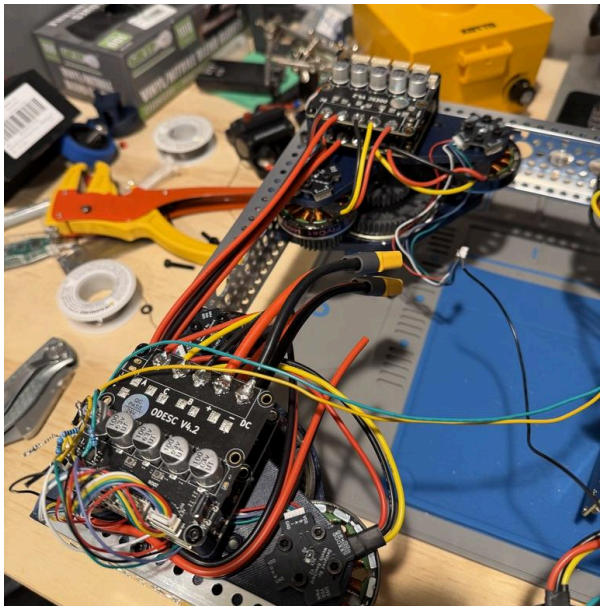


Figure 5: Rewiring process, removing connectors and improving routing

Over the next two iterations, the connectors between the motors and drivers, and the drivers and distribution board were removed in favor of direct solder joints. Additionally, the power wiring to the motor controllers was re-routed to run along the perimeter of the robot frame, to keep the interior clear.

Once the assembly and re-wiring process were complete, my focus moved to the control system testing. Because the Pi runs Linux, I was able to utilize the systemd service manager to automatically run setup commands and launch the control system software when the pi boots up. The system was even able to await a connection from the xbox controller before starting the control system, an important safety feature that prevents garbage movement data from being sent to the motor controllers.

However, significant problems were discovered while testing the system functions that would allow for a graceful shutdown of the control system. These are important because simply cutting power to the Pi risks corrupting its operating system and so properly shutting it down before cutting power is good practice for system longevity. The control system does have this functionality, however running the “sudo shutdown now” command requires the calling process to have superuser privileges, something that can only be obtained after a password is entered by the user. Because the only method of interaction with the robot currently is the xbox controller, there is no way to execute this privileged command, and so this part of the system is nonfunctional.

Finally, an analysis of the control system resource footprint was conducted while the system was running. This analysis utilized the atop command line tool, which provides comprehensive resource monitoring and data logging capabilities. The analysis found that the control system utilizes approximately 20% of the CPU and 50% of the ram. Additionally, the output of the ROS2

controller manager was monitored during testing and no errors related to realtime performance or control loop stability were observed. This confirmed that the control system was running in real time without issue, and that there was performance headroom for more features to be built in the future.

Discussion of Fidelity and Reflection

In terms of the tested functionality this was certainly a mid fidelity prototype. While the parts were close to what would actually be run in a production environment, they were not polished and did not have the features of a high fidelity system. For the physical components, high fidelity versions would certainly have dedicated paths for wire routing and well placed cutouts for zip ties and other retention mechanisms. Similarly, high fidelity, production code would be much more robust, and provide comprehensive logging and telemetry features as well as integrations with industry standard ROS packages such as MoveIt or Nav2. These features would be good targets for a future iteration.

The fidelity of this prototype certainly imposed limitations on the testing process. Specifically, the lack of comprehensive telemetry systems necessitated the use of command line resource monitoring and simple visual / qualitative detection of control system faults or potential errors. With high quality logging, these tests could be expanded and strengthened to allow these functionality claims to be made with more confidence.

References

Not a whole lot here. I did use the following medium article to learn about writing systemd services to start programs when a computer boots:

https://medium.com/@benmorel/creating-a-linux-service-with-systemd-611b5c8b91d6