2026 Team Description Paper: The Bots

Henry Bryant and Mathew MacDougall

thebots.robocup@gmail.com
github.com/sfunderbots
cad.onshape.com/documents/f50de4495ca6813b0d6d1926

Abstract. This paper details the design and development of the systems of The Bots, a Small Size League team intending to compete in RoboCup 2026 in Incheon, South Korea. Adhering to our modular low-cost design, we introduce a solenoid active damping method using a PWM modulated kicker to receive a ball at 5m/s, and a new logging and replay software feature.

Keywords: RoboCup 2026 · Small Size League · Robotic Soccer · Solenoid Active Damping · Logging · Replay

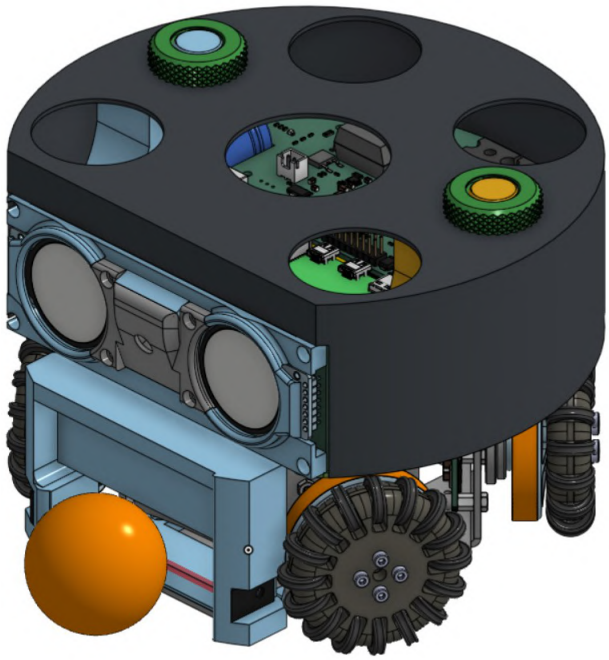


Fig. 1: CAD of Generation 3 Robot.

1 Introduction

The Bots is an interdisciplinary team composed of both university graduates who were formerly part of UBC Thunderbots (Canada), and graduates recruited from other Universities in the USA. Established in 2022, and having placed second in Robocup 2025 in Salvador, Brazil, the team is now pursuing its second competitive action within the Small Size League, seeking qualification for RoboCup 2026 in Incheon, South Korea. In a continual effort to maintain a low development cost model, The Bots has improved ball reception capabilities using active damping without adding extra parts or additional cost. This paper outlines The Bots' progress in developing the mechanical, electrical, and software systems of these robots to compete in RoboCup 2026.

2 Electro-Mechanical

2.1 Solenoid Active Damper

A common problem that many teams have faced is receiving passes at higher speeds. There are two main failure scenarios: (1), the ball hits the receiver but bounces off, or (2) the ball is completely missed.

Several mechanical solutions have been explored to address scenario (1), allowing the dribbler to retain control of the ball. One approach, as described by Twente[6] and RoboJackets[2], uses a foam damping structure to reduce the total reflection distance before the dribbler is able to control the ball. A more sophisticated approach, initially used by TIGERs[7] and improved upon by Thunderbots[9], uses a 3D printed compliant mechanism to support free lateral movement of the dribbler when in contact with the ball. However, these solutions rarely work efficiently at speeds larger than 3.5m/s, as the damping mechanism is unable to absorb sufficient ball momentum for the dribbler to retain control after collision; for example, Twente reported this limitation on their dribbler in 2022[6].

More complex methods have also been used, such as Zjunlict[5] that use a bidirectional spring-damping system but instead with 3 points of contact on the ball to improve the ball on dribbler dynamics, which can receive a ball at 8.5m/s.

Inspired by Zjunlict having the ability to stop the ball at 8.5m/s, we have explored using the kicker solenoid as an active damper. By holding the kicker extended with minimal force, an incoming ball creates a partially inelastic collision on impact. By applying PWM to the high voltage IGBT, we can apply just enough power to overcome the force of the return spring and hold the plunger out. When the golf ball (weighing 45 grams) strikes the plunger (weighing 60 grams), the plunger is driven back to its retracted position, dissipating most of the ball's initial impact energy in the process. Any remaining energy can be absorbed by the rotating dribbler.

Initial testing was performed by having one robot kick the ball at another from a short distance away, observing how the ball bounced back off the dribbler.

Video footage was captured and rectified using known distance measurements and field tape lines as a reference. This enabled accurate calculation of the ball's pre-collision and post-collision velocity. Figure 2.1 shows the rectified frame used for position tracking; the extended plunger (indicated by an arrow) is shown in the receiving position.

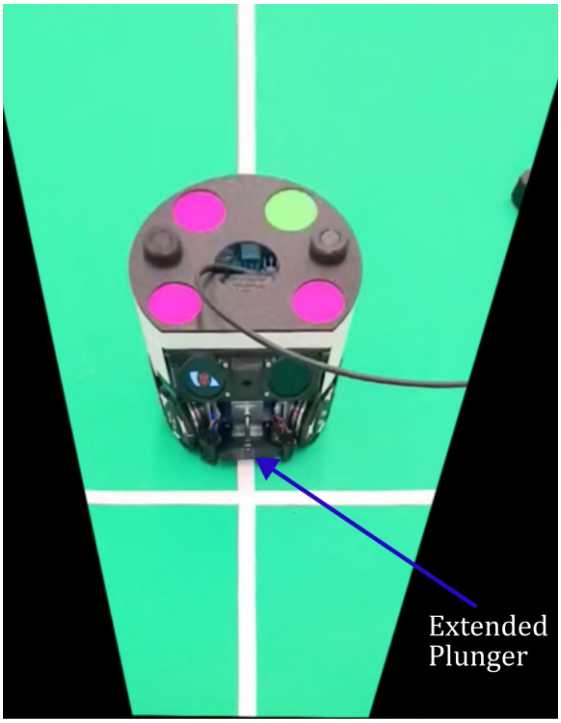
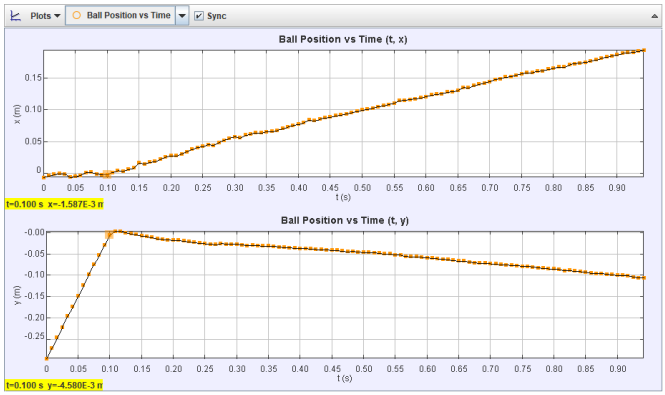


Fig. 2.1: Rectified Frame used for ball tracking, with kicker extended using PWM

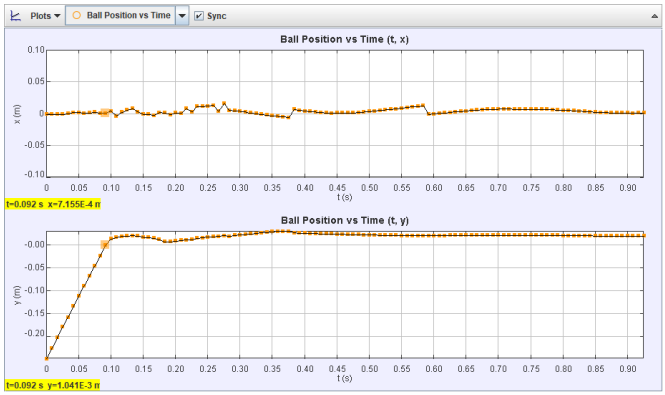
Preliminary tests showed that it was possible to catch and significantly damp a ball without a dribbler up to 5m/s. Ball tracking was performed using Tracker[8], an open sourced physics tracking tool, calibrated using the distance from the extended plunger to the solenoid (45mm), as there were no suitable reference features in the plane of motion of the ball.

The tracked ball positions (using a circular orange template) were plotted[1], as shown in Figure 2.2, and the resulting velocities were calculated and summarized in Table 1:

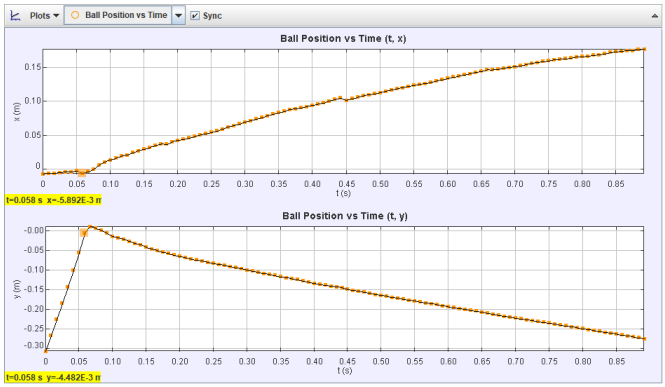
[1] Measurements are approximate; the primary source of error is the calibration dimensions used for rectification.



(a) Test 1: 8.8% Reflection Velocity



(b) Test 2: 0% Reflection Velocity



(c) Test 3: 8.3% Reflection Velocity

Fig. 2.2: Plots of Ball position vs Time for Active Damping Tests

Then, using the measured changes in position (dx, dy) and time (dt), the velocity magnitude can be calculated. Here, v_{in} denotes pre-collision and v_{out} denotes post-collision, computed from the total absolute displacement:

Test no.	dx_{in}	dy_{in}	dt_{in}	v_{in}	dx_{out}	dy_{out}	dt_{out}	v_{out}
1	0.01 m	0.3 m	0.1 s	3 m/s	0.2 m	-0.1 m	0.85 s	0.26 m/s
2	0 m	0.25 m	0.092 s	2.7 m/s	0 m	0 m	0.1 s	0 m/s
3	0 m	0.31 m	0.058 s	5.3 m/s	0.18 m	-0.27 m	0.83 s	0.4 m/s

Table 1: Active Damping Test Position and Velocity Table

During tests 1 and 3, the kicker re-extended after collision, adding energy back to the ball and preventing the balance of forces described earlier. Despite this, the resulting reflection velocity remained under 10% of the input velocity, meaning the active damping successfully absorbed over 90% of the ball's kinetic energy. Some of the inconsistencies in performance were traced to the MicroPython control implementation such as timer resolution (see improvements section).

Based on these results, we believe active damping can provide significant improvement to ball receiving capabilities accessible to many SSL teams. For example, in test 2[2], the damped output velocity suggests a simple dribbler should be able to receive and control the ball at much higher velocities.

To achieve the function of PWM activated damping, the workflow can be described as follows.

1. Produce a soft kick, which is enough to fully extend the kicker with almost zero ball speed
2. Wait for the kick pulse to finish actuating[3]. As a start, we used approximately 15× the pulsewidth
3. Send a PWM signal at a duty cycle that balances the solenoid force against the return spring, determined experimentally.[4]

Tests were conducted with our robot using an OTS 12V Push Pull solenoid and setup with various frequencies, where we recorded the duty cycle threshold needed to keep the kicker extended as follows:

Frequency	Period	Duty	Pulse Length[5]
f	T_s	D	t_{on}
100 Hz	10 ms	1.4%	140 μs
1 kHz	1 ms	3.5%	35 μs
5 kHz	200 μs	15%	30 μs
10 kHz	100 μs	27%	27 μs
20 kHz	50 μs	52%	26 μs
30 kHz	33 μs	82%	27 μs

Table 2: Thresholds to keep kicker extended at various frequencies.

[2] Where the ball briefly bounced in the air before rolling forward.
[3] Mechanical actuation is significantly delayed from the electrical response.
[4] Exceeding this threshold causes re-extension after impact, increasing contact time.

The pulse width values calculated in Table 2 represent the duration (t_{on}) for each cycle relative to the total period (T_s) from our setup (these values are experimental and may not transfer to your setup). The effective power generated by the pulse can be approximated by $f \times t_{\text{on}}$; for example, a 140 μs pulse repeated 100 times every second gives a relative value of 0.014/s. In theory, this value should roughly stay constant (within reason). However, we observed a minimum pulse width threshold required to produce any response from the kicker, which may be attributable to MicroPython timer limitations, solenoid characteristics, or both. All test frequencies produced audible tones, but at higher frequencies, such as 30kHz, the applied pulse width (relative power calculated as 0.81/s) resulted in a significantly louder tone, indicating a much larger power output.

Improvements The difference between observed tests can be attributed largely to MicroPython's 1 μs timer resolution. With this resolution, small deviations in the delivered pulse width could sometimes produce an average pulse width above the threshold (tests 1 and 3), on the threshold (test 2), or below the threshold (tests not shown, as the kicker was not held after the initial pulse).

Due to the power consumption of each pulse, it is also beneficial to increase the frequency to spread out the overall power dissipation in the solenoid over time which leads to smaller spikes of current and thus heat generation in the current traces. A frequency of 10kHz was chosen (with an associated duty cycle of 27%) as a practical middle ground, as higher frequencies produced a noticeably higher power output, while ensuring a duty cycle above 5%, which generally becomes impractical to drive at such a low or high duty cycle.

While this limitation is a consequence of MicroPython's 1 μs timer resolution, using a proper PIO implementation of PWM (such as how the kick pulse code is generated), which can operate at the 125MHz system clock achieving sub-microsecond pulse resolution, would eliminate this variability entirely. This has since been implemented, though the finalized firmware has not yet been published to our repository. Future work includes verifying the pulse widths with oscilloscope measurements to validate the set values calculated above.

Power Draw Discussion In these tests, the voltage only ever dropped a maximum of about 60V during a two second period, resulting in an approximate 30V/s decay of HV. This means that with an accurately timed active damping, we could immediately kick at full speed (without charging) if necessary, as it is estimated that a full speed kick discharges only about 100V of our 210V capacitor banks. However it should also be noted that a re-charge only takes around 0.3s, easily within the time AI needs to reposition to align a kick.

[5] Pulse length calculated as $t_{\text{on}} = T_s \times D$.

3 Software

3.1 New Logging Framework

Many teams have previously highlighted the value of being able to log and replay detailed visualizations of their games and AI logic [7]. However when building our own logging and replay system, we realized that some useful visualizations are too performance intensive to generate and log in the real-time environment of an SSL game. In particular, we discovered that generating a debug heatmap [12] for each of the 7 cost components of our pass evaluation was adding a total of 10ms to each tick of our AI runtime. These 10ms were due to the time taken to generate the heatmaps themselves rather than logging them to disk, since our initial logging prototypes deferred the message serialization and file IO to a background thread, removing them from the critical path. Logging 7 heatmaps each with a grid of $46 * 31 = 1426$ 64-bit floating point values, results in $7\ heatmaps * 8\ bytes * 1426\ values = 79,856\ bytes$ of data to log each tick. Assuming our AI runs at the target rate of 60Hz, this debug data adds 4.57Mb/s of data to our logs in addition to the latency penalty which makes hitting this performance target nearly impossible. Conservatively assuming our AI needs to run for 1 hour for an SSL game, these debug logs alone would account for an additional 16Gb of (uncompressed) disk space. While compression strategies are available to reduce the space usage, it is still a significant increase and indicates a naive solution that directly logs any and all information we may want to visualize in a replay is clearly not a feasible solution.

Definition 1. *A node is an individual component of the software stack that runs in an independent thread and communicates with other nodes via message-passing. For example, the Perception node that is responsible for vision filtering or the Gameplay Logic node that is responsible for gameplay decisions.*

Our solution was to formulate each node in our system as a Mealy Machine [13], where the node defines a stateless "tick" function, and the output depends both on the input and current state of the node. The tick function also evolves the state.

$Output_n, State_{n+1} = tick(State_n, Input_n)$

This requires each node to explicitly centralize all its Input and State data into individual structs, as well as for the tick logic to be deterministic so that given the same Input and State the same output is produced. For example, this requires either removing any randomness from the logic or storing any random state in the State struct.

Once this is in place, it is simple to add logging for each individual node. Before running each tick the Input, State, tick index, and current time are stored in a "ReproducibleInput" struct. The sequence of ReproducibleInputs is serialized and logged to disk as the node's logfile. The end result is a logfile that has all the

necessary information to deterministically re-run any tick that occurred during the log.

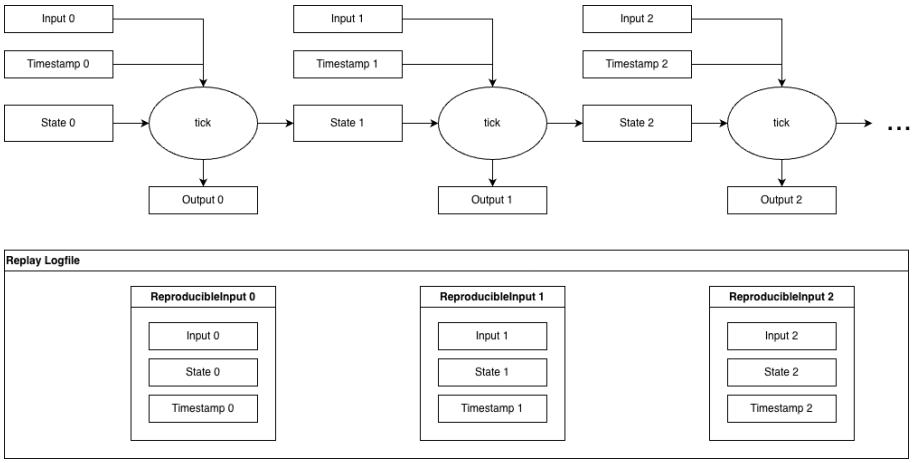


Fig. 3.1: Control flow of node tick and data logging

One caveat is that the logging is done per-node. There is no centralized logger listening to all messages in the system, so if a node is not running it will not generate a log. In practice the main challenge this presents is when we want to run the software without the GUI, for example during a simulated integration test. If the GUI is not run, then a GUI log will not be generated and we will not be able to visualize the log as a replay. Our solution to this problem is to always run a minimal version of the GUI that only performs the necessary state updates to generate a log without rendering any part of the GUI.

Compared to a centralized logging solution, the per-node logging can also result in additional storage usage since the same data may be logged by multiple nodes. For example, if two nodes use the same input data then two copies will be logged, while a centralized logger would only log a single copy.

On the other hand, the per-node logging makes accurately reconstructing and replaying logs significantly easier than with a centralized logging solution. Since each node handles the logging of its own Input and State before each tick, the log is from the perspective of the node. Each ReproducibleInput contains the exact data that was used for the tick without the need for additional bookkeeping to determine which individual input messages were available at the time the tick was run, which would be required with a centralized logger. Our conclusion was that this simplicity is worth the extra storage usage and always running a minimal version of certain nodes.

With this system in place, basic visualization of a log works as follows:

1. The GUI node loads the desired logfile
2. As the replay time advances, the GUI searches the logfile for the ReproducibleInput with the timestamp closest to the current replay time
3. The GUI tick function is re-ran with the Input and State from the ReproducibleInput, and the new state and outputs are rendered in the GUI. If the tick function is deterministic, this should be an exact reproduction of the original GUI tick and should display the exact same information.

3.2 Enhanced Debugging Enabled by this Logging Architecture

Because the logging system allows us to deterministically re-run any tick exactly as it happened, this opens up several new debugging capabilities.

The first is that we can re-generate any debug data that would have been too performance or data intensive to log during a live game. For example, the passing cost debug heatmaps mentioned in the previous section. First we place the debug heatmap generation logic behind a debug flag that defaults to false so that it does not run during games. In the GUI state we store the tick index of the most recently consumed AI outputs. When loading a GUI replay, this allows us to easily lookup the ReproducibleInput for the AI tick that generated the AI outputs being visualized in the replayed GUI tick.

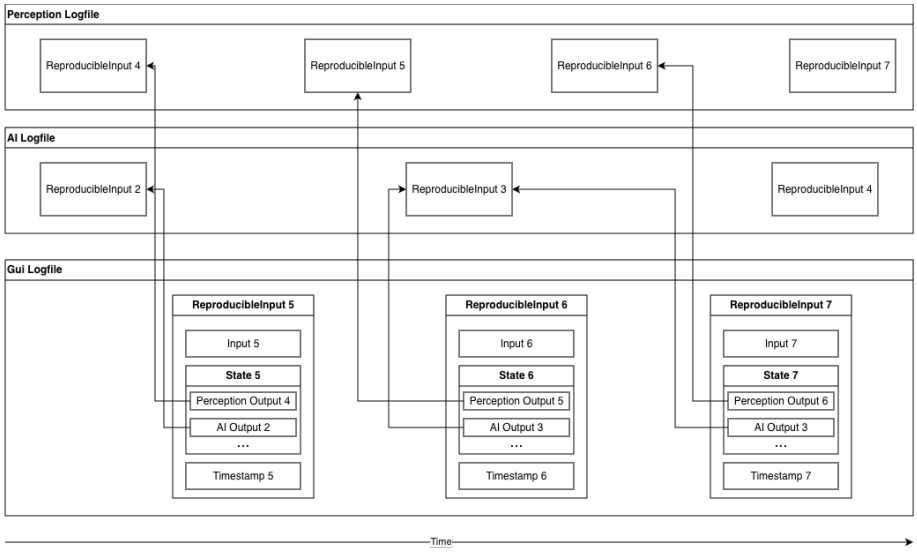


Fig. 3.2: Data flow diagram showing how the GUI tracks which outputs were visualized in each frame

When we want to visualize the debug heatmaps, we set the debug flag to true and use the AI's ReproducibleInput to re-run the AI tick. This produces a new AI Output, now containing the debug heatmaps, that can be passed back to the GUI for visualization. This allows us to generate and visualize the debug heatmaps without having logged them in the first place.

The second capability unlocked by our logging system is the ability to re-run ticks with a debugger such as gdb. A debugger can be attached to the live node process when re-running a logged tick as described above, or the entire application can be run under a debugger when replaying a log. This offers developers the incredibly powerful ability to both reproduce exact behavior and retroactively debug any issues observed during a log using modern debugger features.

3.3 Known Limitations

The primary limitation of our new logging solution is the lack of backwards compatibility. This is primarily due to using the Rust Serde library with the bitcode[11] format for serialization and deserialization of log messages, instead of formats such as Protobuf that support backwards compatibility. We chose Serde+bitcode due to its ease of integration with our stack, as it does not require manually maintaining a schema like Protobuf. The primary purpose of the logs are for immediate debugging and not long-term records, so backwards-compatibility is not a concern. Logs should generally be loaded on the same sha that produced them in order to ensure the nodes and behavior are reproduced exactly.

For similar reasons to the above, the logging system has limited ability to recover partially-corrupted logs. Bitcode is not able to deserialize partially corrupted data, so we do not have the option to recover partial logs. We have mitigated this problem by chunking the logs into smaller files, so that if an individual chunk becomes corrupted the other chunks can still be loaded. We do not expect this to be a common problem and therefore do not think it is worth the additional complexity of solving at this time.

We are extremely excited to use this new logging system at RoboCup 2026 and expect it will enable us to promptly debug issues without compromising on the computational performance of our AI software.

4 Electrical

4.1 Magnetic Shell-Detection Mechanism

In 2024[12], we proposed an idea for a limit switch or reed switch-based shell-detection mechanism to protect the user from unsafe high voltage by detecting the presence and absence of the robot's shell. While not directly relevant to robot performance metrics such as passing or shooting performance, high voltage safety is a fundamental design consideration for any robot handled at a competition. It is widely understood that any voltage over 60Vdc is potentially hazardous to humans[1], where lethality can be assumed based on the available current in the circuit. However, in capacitive based high voltage systems, such as SSL robots, this energy is stored instead of continuous. The energy stored in The Bots' chicker board (a 2000uF system at 210V worst case) is calculated as follows:

$$E = \frac{1}{2}CV^2 = \frac{1}{2}2000\mu F \times (210V)^2 = 44.1J \tag{1}$$

Understanding the lethality of stored capacitive energy has been explored by many researchers over the years, where Gordon et al. [4] explored the effects of capacitive electrical shock among other causes. As explained in this paper, early researchers found that while most shocks below 100J were found to not pose a fibrillation risk, if the shock was delivered "during the vulnerable part of the cardiac cycle (the T wave)" fibrillation could potentially happen - estimated between 25 and 50J. Importantly, lethality in capacitive systems is determined by both stored energy and discharge time, which is the minimum of either contact time or the RC constant dominated by skin resistance. Gordon et al. then classify 100-400V, 1-100J systems as a conservatively rated moderate hazard where fibrillation is unlikely but injury is possible, dominated by a slow energy transfer to the body.

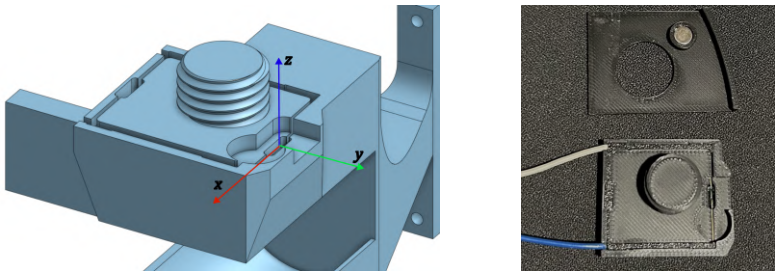
While the conservative moderate hazard classification offers some reassurance, the 44J on a fully charged chicker board falls in the 25-50J cardiac T-wave fibrillation window and could potentially pose a threat. An active shell-detection mechanism is therefore necessary to prevent accidental contact with a charged board, reducing the risk of injury and the possibility of fibrillation under adverse conditions. However, the main challenge with such a mechanism is if a robot was on the field when the system detected "No Shell," it would disable a robot's ability to kick.

While reed switches are known to exhibit contact chattering due to vibration[3], this was preferred over the failure mode of mechanical limit switches, which degrade due to mechanical wear from repeated physical actuation, likely accelerated by the high-vibration environment of an SSL robot. Decreasing the air gap between the magnet and the reed switch (MKA-07101 in our case) to zero is expected to decrease the chattering risk, which can be further supported by software debouncing.

Beyond chattering, a secondary failure mode must also be considered from our design in 2024[12]: if a switch connected to the RP2040 were to fail or the circuit is disconnected (open or short circuit failure), “No Shell” and “Disconnected” would both produce 0V. Given the risks involved with such a system, it did not seem logical to try to cover all edge cases in software alone, particularly accounting for on/off-field operation scenarios such as active gameplay and testing exceptions. To mitigate the risk, we chose to implement two solutions as follows:

- Have our AI logic control the Chicker board (using the new Chicker code structure[6]) based on the reed switch status[7] and gameplay status
- Implement a hardware fail-safe using a voltage divider to eliminate the disconnected circuit failure explained above, which removes the need for AI to cover most edge cases and instead only cover important fail-safes.

A shunt resistor, embedded on the left as shown in Figure 4.1, creates a voltage divider when connected and allows an ADC pin to discern ‘Disconnected’ (3.3V), ‘Open’ (1.65V), and ‘Closed’ (0V). The reed switch, embedded on the right, is installed first with soldered bent leads, after which the insulation is stripped carefully in the contact area to solder the shunt resistor second.



(a) Indentation Design (with coordinate system)

(b) Physical Implementation

Fig. 4.1: Reed Switch Implementation

We can determine the threshold of the reed switch through experimental testing and by evaluating the magnitude of the magnetic field over the reed (disregarding constants). Experimentally, passing the magnet over the reed gives approximate turn-on and turn-off distances of $x = |3mm|$ and $x = |15mm|$ from the reed center ($x = 0$), using the coordinate system shown in figure 4.1b above.

While a reed switch output is binary, the field seen by the reed and its effective threshold is not; magnet misalignment or shell rotation[8] can shift the peak lo-

[6] Expects a Mode identifier to change between ‘Idle’ (discharged), ‘Autokick’ (Charge and wait for kick), ‘Damp’ (Tongue Damper), and ‘Charge’ (Re-charge after Damp)
[7] The reed switch can be ignored for `"Dont Care"` gameplay scenarios
[8] Due to assembly error or shell vibration

cation or magnitude[9], potentially causing the reed to fall below the switching threshold. To account for this, we calculate the normalized magnetic field along the reed; the exact field strength is not important - thus, verifying the peak of a properly placed magnet ensures the reed switch reliably closes despite potential misalignment.

As a reed switch only responds to the magnetic field perpendicular to its plane, we can mount the reed along the x-axis to measure B_z, which is the dominant field generated by magnet on the flat face[10]. Fixed constants from our setup are:

Description	Formula	Value
Magnet Radius	R	3 mm
Magnet Depth	L	3 mm
Reed Switch Thickness	T	2 mm
Reed Switch Length	l	6.5 mm
Center to Center Distance	$z_0 = L/2 + \text{gap} + T/2$	2.5 mm
Distance to Top Magnet Face	$z_{\text{top}} = z_0 - L/2$	1 mm
Distance to Bottom Magnet Face	$z_{\text{bottom}} = z_0 + L/2$	4 mm

Table 3: Magnet and Reed Switch Constants

To calculate $B_z(x)$, we will convert our cartesian coordinates to polar coordinates (r, θ). While the the magnetic field of a magnetized cylinder can be found using the surface integral (equation 2), the angular face integral for the circular magnet cannot be evaluated using elementary algebraic functions. Thus, the solution for the magnetic field is expressed in closed form using complete elliptic integrals $K(m)$ and $E(m)$[10]:

$$B_z(\mathbf{r}) = \frac{\mu_0}{4\pi} \iint_{\text{top + bottom faces}} \frac{(\mathbf{M} \cdot \hat{\boldsymbol{n}})(\mathbf{r} - \mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|^3} \mathrm{d}S \tag{2}$$

$$B_z(r, z) = \frac{\mu_0 M}{4\pi} \int_0^R \left[\left(\frac{(R - r')}{\sqrt{d(-r', z)}} - \frac{(R + r')}{\sqrt{d(r', z)}} \right) E(m) + zK(m) \right] \mathrm{d}r' \tag{3}$$

Where $d(r', z) = (R + r')^2 + z^2$ and $m = \dfrac{4Rr'}{d(r', z)}$

The complete elliptic integral special functions[10] are defined with $\phi = \frac{\pi}{2}$:

$$K(\phi, m) = F\left(\frac{\pi}{2}, m\right) = \int_0^{\frac{\pi}{2}} \frac{\mathrm{d}\phi'}{\sqrt{1 - m \sin^2(\phi')}} \tag{4}$$

[9] An x-axis rotation shifts the peak along x, while a y-axis rotation primarily changes the peak magnitude.

[10] Consequently, B_x and B_y are ignored.

$$E(\phi, m) = F(\frac{\pi}{2}, m) = \int_0^{\frac{\pi}{2}} \sqrt{1 - m\sin^2(\phi')}\mathrm{d}\phi' \tag{5}$$

Finally, we can represent the total magnetic field of the magnet using a difference of the top face z_{top} and the bottom face z_{bottom}:

$$B_z(r) = B_z(r, z_{top}) - B_z(r, z_{bottom}) \tag{6}$$

However, because the reed switch is seeing an average over $l = 6.5mm$ in length, we cannot assume that the point average field is correct. We need to localize this average over the integral of the reed switch length (i.e. $x - \frac{1}{2}$ and $x + \frac{1}{2}$), which is shown in the following figure as well as the equation as follows:

$$B_{\text{reed}}(x) = \int_{x-\frac{l}{2}}^{x+\frac{l}{2}} B_z(x')\mathrm{d}x' \tag{7}$$

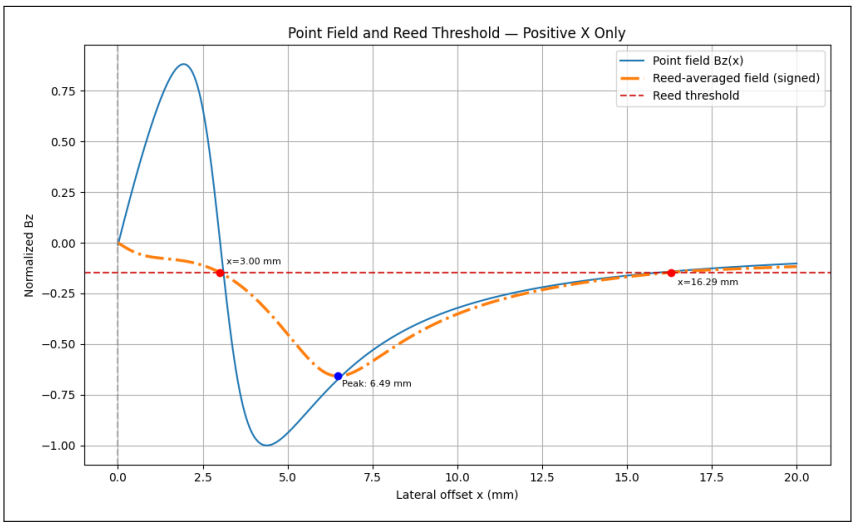


Fig. 4.2: Vertical Magnetic Field Calculation

As such, we see that the reed-averaged field $B_{\text{reed}}(x)$ around $x = 0mm$ is cancelled due to the inverted symmetry of the magnet field, and passes the experimentally determined threshold ($x = 3mm$) at $x = 16.29mm$, which is comparable to our measured $15mm$ threshold from earlier. If the magnet were to get off axis, the peak location and amplitude of the averaged waveform would change, potentially dropping below the threshold. Therefore, the optimal magnet placement is at $x = 6.49mm$.

5 Conclusion

In this paper we introduced our new solenoid active damping mechanism, magnetic shell detection system, and software logging framework. The solenoid active damping system offers a new and innovative way to receive fast-moving balls without a significant impact on robot cost or complexity, making it a viable option for lower-budget teams to improve their performance. The magnetic shell detection system provides an additional layer of safety to avoid humans making contact with the robot's high-voltage systems while they are live, which is a non-zero risk and has happened before in high-stress environments like RoboCup. Finally our logging framework describes a new approach and tradeoffs for logging, replaying, visualizing, and debugging the software game data at RoboCup, with a focus on precise replay and debuggability in favor of backwards compatibility and disk usage.

References

1. Use of conventional touch voltage limits - Application guide. Tech. Rep. IEC/TS 61201:2007, International Electrotechnical Commission, Geneva, Switzerland (2007)
2. An, G., Buchanan, J., Csukas, S., Iachonkov, B., Jones, J., Kamat, J., Mansour-Elsayed, A., Mycroft, A., Naeem, S., Strat, R., Stuckey, W.: RoboJackets 2016 Team Description Paper (2016)
3. Ayurzana, O., Tsagaanchuluun, S., Kim, H.: Analysis of Chattering Error on Reed Switch for AMR System of Water Supply. In: 2014 7th International Conference on Ubi-Media Computing and Workshops (2014). https://doi.org/10.1109/6916345
4. Gordon, L., Cartelli, L., Graham, N.: A Complete Electrical Shock Hazard Classification System and Its Application. IEEE Transactions on Industry Applications **54**(6) (2018). https://doi.org/10.1109/TIA.2018.2803768
5. Huang, Z., Chen, L., Li, J., Wang, Y., Chen, Z., Wen, L., Gu, J., Hu, P., Xiong, R.: ZJUNlict Extended Team Description Paper for RoboCup 2019 (2019)
6. Monat, C., Dankers, E., Skurule, K., Steenmeijer, L., Sijtsma, S., Diamantopoulos, S., Aggarwal, R., Smit, T., van Harten, A.: RoboTeam Twente Extended Team Description Paper for RoboCup 2022 (2022)
7. Ommer, N., Ryll, A., Geiger, M.: TIGERs Extended Team Description for RoboCup 2022 (2022)
8. Open Source Physics: Tracker Video Analysis and Modeling Tool. https://opensourcephysics.github.io/tracker-website/ (2020)
9. Senthilkumar, A., Sidhu, A., Balamurali, A., Sturn, D., Antoniuk, D., To, D., Muhstaq, F., Crema, F., Bryant, H., Rovner, H., Lew, J., Wakaba, K., Zareian, N., Levy, O., Khan, R., Cao, R., Nedjabat, R., Kong, T., Ajmal, S., Sylvia Ly, S., Zhou, Y.: 2023 Team Description Paper: UBC Thunderbots (2023)
10. Slanovc, F., Ortner, M., Moridi, M., Abert, C., Suess, D.: Full analytical solution for the magnetic field of uniformly magnetized cylinder tiles. Journal of Magnetism and Magnetic Materials **559**, 169482 (2022)
11. SoftbearStudios: bitcode. https://github.com/SoftbearStudios/bitcode
12. Veeraghanta, A., Bryant, H., Zheng, S., MacDougall, M., Lee, A., Guido, S., Bontkes, L., Van Dam, W.: 2024 Team Description Paper: The Bots (2024)

13. Wikipedia contributors: Mealy machine — Wikipedia, the free encyclopedia (2026), https://en.wikipedia.org/wiki/Mealy_machine, [Online; accessed 24-January-2026]