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

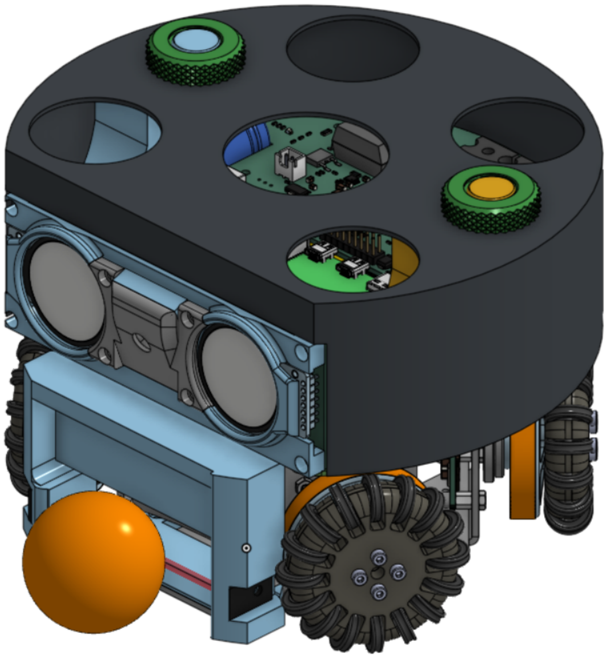


Fig. 1: CAD of Generation 3 Robot.

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 likely scenarios when a receiver is unable to successfully control the ball: (1), the ball hits the receiver but cannot maintain control and bounces off, or (2) the ball is missed entirely. Some solutions for (1) that teams have tried can either be reduced to improving the foam damping structure[4][2], or implementing a 3D printed compliant mechanism to support bidirectional movement of the dribbler[7][5][3] which aim to reduce the reflection after the ball hits the dribbler assembly.

However, these solutions rarely work efficiently at speeds larger than 3.5m/s[4], because the damping systems are unable to absorb enough of the ball's momentum for the dribbler mechanism to retain control of the ball after collision. Importantly, it should be noted that we are only considering the standard two-point dribbler, unlike Zjunlict's bidirectional 3 point dribbler that can receive a ball at 8.5m/s[3].

We explored using the kicker solenoid as an active damper. The kicker head is extended and held in an outward position by PWM'ing the IGBT. We apply just enough power to overcome the force of the return spring and hold the plunger out. The plunger weighs 60 grams unloaded. When the golf ball, weight 45 grams strikes the plunger, 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 just be absorbed by the rotating dribbler.

Initial testing was performed by having one robot kick the ball at another from a short distance away, and observing how the ball would bounce back off the dribbler. Video footage was captured and rectified using known distance measurements and known parallel lines using python in the frame to accurately calculate the ball's collision speed and the speed after the collisions. The following figure shows the rectified frame used for calculating positions, and shows the extended plunger waiting to receive a pass (figure 2.1):

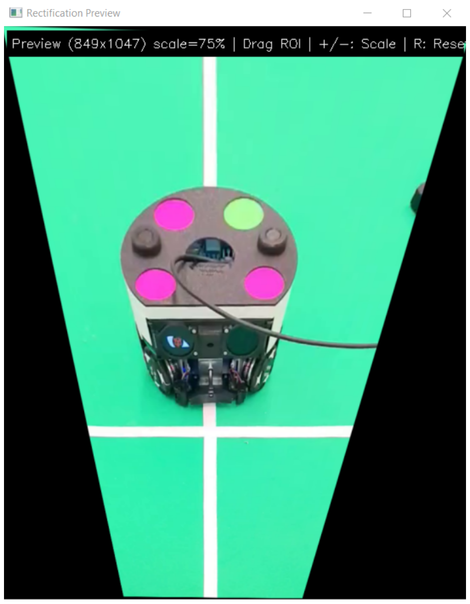
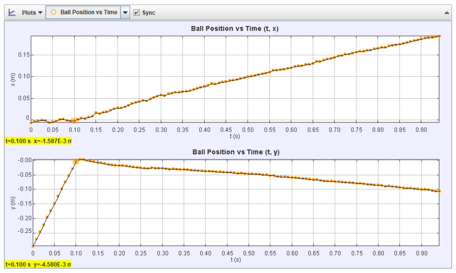

Rectification Preview
Preview (849x1047) scale=75% | Drag ROI | +/−: Scale | R: Rese

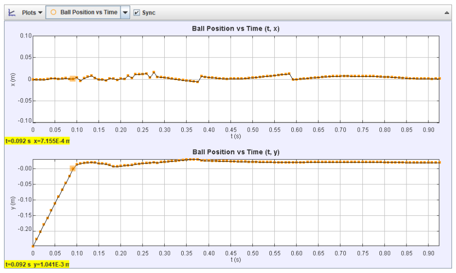
Fig. 2.1: Rectified Frame used for ball tracking, with kicker extended using PWM

Preliminary tests showed that it was possible to catch and significantly damp a ball without a dribbler up to 5m/s[1]. The tracking of the ball was performed using Tracker[6], an open sourced physics tracking tool, calibrated using the distance from the extended plunger to the solenoid (45mm) as there were no useful markings in the footage in line with the z plane of the ball. The autotracked ball (using a circular orange template) was then plotted, shown in figure 2.2, then the resulting velocities calculated in Table 1:

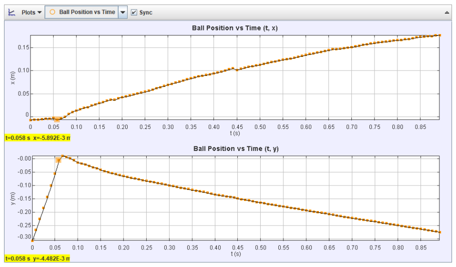
[1] These measurements are approximate, using the rectified perspective described earlier, with the only major issue being the the error on the calibration dimensions.



(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, noting the approximate change in position (dx and dy) and change in time (dt) we can calculate the absolute velocities (where v_{in} denotes pre-collision and v_{out} denotes post-collision) using the total change in position regardless of direction:

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

It should be noted that during tests 1 and 3, the kicker re-extended after collision, adding energy back to the ball after the initial damping collision. These tests did not achieve the balance of forces stated earlier, but still produced a resulting reflection velocity of under 10% of the input velocity (hence, the active damping successfully damped over 90% of the energy). We've identified MicroPython to be one of the reasons for the inconsistent performance (see improvements section below).

However, through testing this prototype, we believe active damping can provide significant improvement to ball receiving capabilities in a way that is accessible to many SSL teams. With a damped output velocity as shown in test 2 (where the ball bounced in the air briefly and then rolled forward) 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[2]. As a start, we used approximately $15\times$ the pulsewidth
3. Send a PWM frequency to the kicker with a duty cycle that corresponds to the threshold that balances the spring (retraction) force of the kicker with the electrical force generated from the solenoid. This was done experimentally, but could be justified with formulas if necessary.[3]

Tests were conducted with our solenoid[4] and setup with various frequencies, where we recorded the threshold needed to keep the kicker extended as follows[5]:

Frequency	Period	Duty	Pulse Length
f	T_s	D	t_{on}
100 Hz	10ms	1.4%	$10ms \times 1.4\% = 140\mu s$
1kHz	1ms	3.5%	$1ms \times 3.5\% = 35\mu s$
5kHz	$200\mu s$	15%	$200\mu s \times 15\% = 30\mu s$
10kHz	$100\mu s$	27%	$100\mu s \times 27\% = 27\mu s$
20kHz	$50\mu s$	52%	$50\mu s \times 52\% = 26\mu s$
30kHz	$33\mu s$	82%	$33\mu s \times 82\% = 27\mu s$

Table 2: Thresholds to keep kicker extended at various frequencies

Improvements With our current implementation using micro-second timers on the RP2040 in MicroPython, as you increase frequency, you also increase the PWM error which is dependent on the PWM resolution and the total period. With 30kHz, the error is 3%, which can significantly increase inaccuracy of the pulse length at higher frequencies, because the error accumulated over time is significant over 10,000+ cycles. This could either produce a result 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 extended 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. In theory, it would be possible to find an optimum between keeping PWM

[2] Note, the mechanical actuation is significantly delayed from the electrical response

[3] Note, if the duty cycle is set too high past the threshold, the forces will not be balanced and thus the kicker will re-extend after receiving the ball, effectively increasing the contact time and force applied to the ball, acting as a semi-solid wall instead of a damper

[4] OTS Solenoid from Amazon as used in our v2024 and v2025 robot[1]

[5] Note: these duty cycles may not transfer accordingly to your system

error low while also keeping the frequency as large as possible. The tests we ran used a frequency of 10kHz, and a duty cycle of 27%.

Lastly, it is good practice to use a duty cycle that is outside of 10% of the maximum duty range, because it becomes impractical to drive at such a low duty or such a high duty due to the error described earlier.

In these tests, the voltage only ever dropped a maximum of about 60V during a one to two second period, resulting in a 30-60V/s decay of HV. This means that with an accurately timed active damping, we could immediately kick at full speed (without charging) if necessary. 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.

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 [5]. 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 [9] 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 [10], 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.

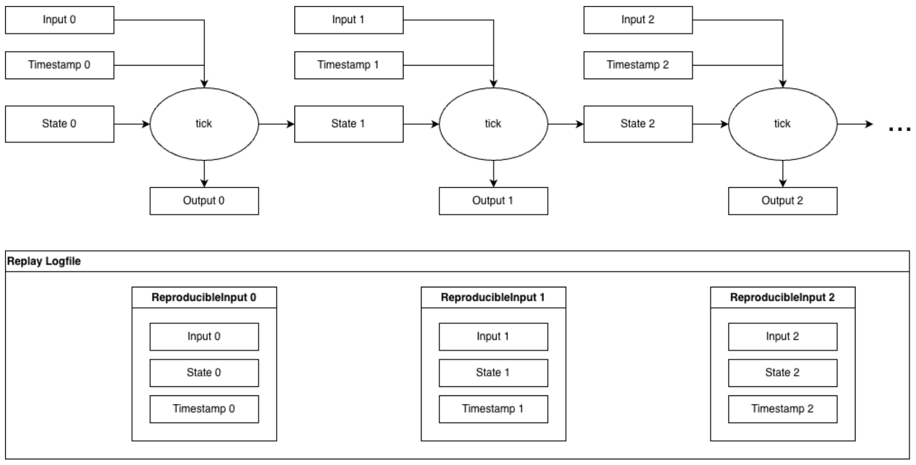


Fig. 3.1: Control flow of node tick and data logging

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.

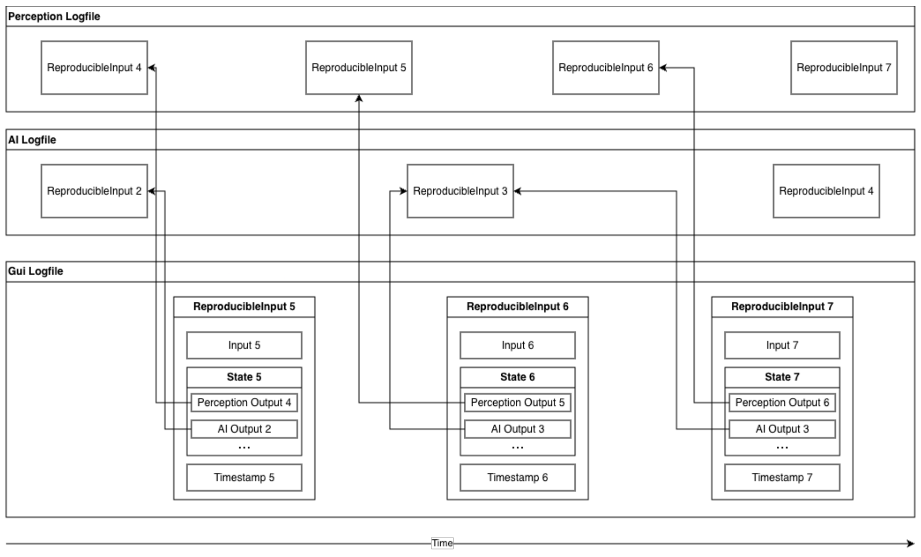


Fig. 3.2: Data flow diagram showing how the GUI tracks which outputs were visualized in each frame

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.

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[9], we proposed an idea for a limit switch or reed switch-based shell-detection mechanism. This mechanism would detect the presence and absence the robot's shell. With this information, we can decide whether it is safe to charge the high-voltage Chicker board or not, to prevent unsafe human exposure to HV electronics when the shell is off. The main challenge with a shell-detection mechanism is if a robot was on the field when the system detected "No Shell," it would disable a robot's ability to kick.

We decided to go with a reed switch to reduce the likelihood of accidentally turning on our HV electronics with the shell off. However, reed switches are more prone to vibration chatter, which can theoretically cause false positives, and in rare instances cause vibration damage. We are hopeful that the close proximity of the magnet (zero air gap) will help prevent this. Lastly, we need to mitigate a risk from our design in 2024[9]: 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.

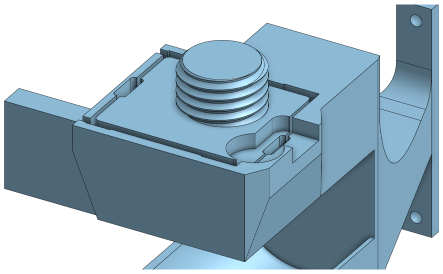
Given the risks involved with such a system, it did not seem logical to try to cover all edge cases for a "nice to have" system, particularly accounting for on/off-field operation, such as whether a game was active and exceptions for testing. To mitigate the risk, we chose to implement two solutions 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.

Adding a shunt to the switch, which is embedded on the left in the model shown in 4.1, creates a voltage divider when connected and allows an ADC pin to be able to discern 'Disconnected' (3.3V) from 'Closed' (0V) or 'Open' (1.65V). The reed switch is similarly embedded on the right, which can be installed first with soldered bent leads, and then the wire stripped carefully in the area to solder the resistor to be installed second.

[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


(a) Indentation Design

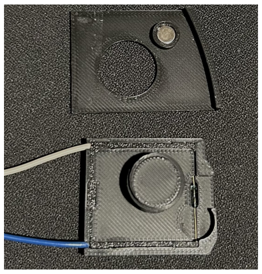

(b) Physical Implementation

Fig. 4.1: Reed Switch Implementation

Determining the threshold of the reed switch was done by passing the magnet over the reed approximating the turn on/off x-axis distances of $x = |3mm|$ and $x = |15mm|$. The placement of the magnet was determined by comparing the theoretical $B_z(x)$ magnetic field, where B_z is the field perpendicular to the reed (ignoring B_x and B_y). Fixed constants from our setup are:

Description	Parameters
Magnet Radius and Depth	$R = 3mm$, $L = 3mm$
Reed Switch Thickness and Length	$T = 2mm$, $l = 6.5mm$
Center to Center Distance	$z_0 = L/2 + gap + T/2 = 2.5mm$
Distance to top Magnet Face	$z_{top} = z_0 - L/2 = 1mm$
Distance to bottom Magnet Face	$z_{bottom} = z_0 + L/2 = 4mm$

Table 3: Magnet Fixed 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 1), 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 $E(m)$ [8]:

$$B_z(\mathbf{r}) = \frac{\mu_0}{4\pi} \iint_{\text{top + bottom faces}} \frac{(\mathbf{M} \cdot \hat{\boldsymbol{n}})(\mathbf{r} - \mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|^3} \mathrm{d}S \tag{1}$$

$$B_z(r, z) = \frac{\mu_0 M}{4\pi} \int_0^R \left[\left(\frac{(R - r')}{\sqrt{d(-r', z)}} - \frac{(R + r')}{\sqrt{d(r', z)}} \right) E(m) + zK(m) \right] \mathrm{d}r' \tag{2}$$

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

The complete elliptic integral special functions [8] are defined with $\phi = \frac{\pi}{2}$:

$$K(\phi, m) = F(\frac{\pi}{2}, m) = \int_0^{\frac{\pi}{2}} \frac{\mathrm{d}\phi'}{\sqrt{1 - m \sin^2(\phi')}} \tag{3}$$

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

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}:

$$B_z(r) = B_z(r, z_{top}) - B_z(r, z_{bottom}) \tag{5}$$

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:

$$B_{\text{reed}}(x) = \int_{x-\frac{l}{2}}^{x+\frac{l}{2}} B_z(x')\mathrm{d}x' \tag{6}$$

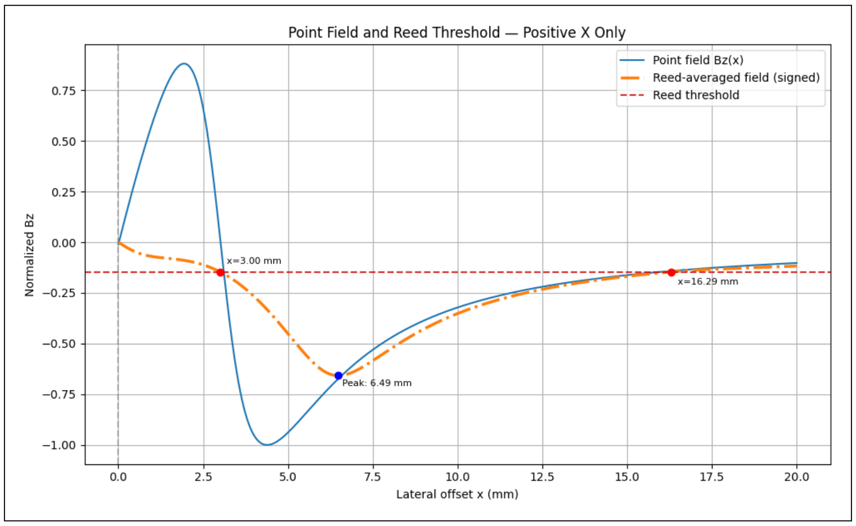


Fig. 4.2: Vertical Magnetic Field Calculation

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 magnitudes of the average waveform would shift, and the magnet field could go below the threshold if the magnet location was placed improperly. As such, we can conclude the best possible choice for the magnet location is at $x = 6.49mm$.

5 Conclusion

We aim to improve our solenoid active damping mechanism, our mechanical performance (including consistent wheel traction and kick angle), and refine our software before RoboCup 2026.

References

1. Amazon Vendor: Push Pull Solenoid DC 12V 35mm Long Stroke
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. 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)
4. 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)
5. Ommer, N., Ryll, A., Geiger, M.: TIGERs Extended Team Description for RoboCup 2022 (2022)
6. Open Source Physics: Tracker Video Analysis and Modeling Tool. https://opensourcephysics.github.io/tracker-website/ (2020)
7. 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)
8. 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)
9. Veeraghanta, A., Bryant, H., Zheng, S., MacDougall, M., Lee, A., Guido, S., Bontkes, L., Van Dam, W.: 2024 Team Description Paper: The Bots (2024)
10. Wikipedia contributors: Mealy machine — Wikipedia, the free encyclopedia (2026), https://en.wikipedia.org/wiki/Mealy_machine, [Online; accessed 24-January-2026]