Fallback Strategies for Handling Sensor Failures in Botball

Thomas Schweiger*, Maja Schorf
Höhere Technische Bundeslehr- und Versuchsanstalt Wiener Neustadt
Higher Technical Federal Teaching and Research Institute Wiener Neustadt
Department of Computer Science
2700 Wiener Neustadt, Austria
*Corresponding author's email: thomas.schweiger.47@gmail.com

***Abstract*—When creating a code for a competition like Botball, the hardware is expected to function as designed. These expectations come with certain risks. These risks include faultiness and failure of distinct parts, which can have a major impact on the strategies used. Sensors play a huge role because failure of these sensors can lead to devastating setbacks when performing tasks such as sorting by color or identifying objects with a camera. This paper aims to bring analyzation and documentation of the efficiency a robot has when it has to resort to a fall strategy due to a sensor failing closer. Fallback strategies decrease the number of robots that stop operating when a sensor fails, this paper wants to bring forth the implementation and accuracy of them, as well as increase the importance of fallback strategies when defining strategies and developing consistent code.**

***Index Terms*—Fallback Strategies, Sensors, Gyro, Motor Ticks, Error Handling, Reliability**

I. Introduction

The ideal robot consists of a duality of consistent code and working hardware [1] (see pages 1-20). When the hardware does not comply, the robot might stop working entirely. An example would be that, when following a line and the sensor malfunctions, the robot continues in one direction, which can be a curve or a straight line. This paper presents a solution and the benefits of fallback strategies that handle sensor failure. Through an experience report from ECER teams, unexpected errors or mishaps ranging from code executing in an incorrect order to parts of the robot falling apart or breaking entirely were deduced, which creates the problem that both hardware and software cannot work without the other. Stable hardware is of minimal use with poorly written code, same as an immaculate code not able to operate on failing hardware, the latter issue can be kept in check by adding slower and less efficient fallback code as a sort of safety net into the code. By giving individual code units a time to live, teams can maximize the time their robots have in a Botball run to score points by resorting to more simplistic code which executes consistently without relying on hardware too much.

Fallback Strategies add rationality to the robot's decision-making process. The best way to discover the most optimal usage of strategies is to identify the crucial sensors and find efficient backup code for potential failure and to see how it impacts the robots overall performance. Research Question: This study investigates how fallback strategies influence the reliability and task performance of mobile robots when critical sensors fail during execution. It is hypothesized that properly designed fallback strategies can maintain acceptable task accuracy while increasing the overall robustness of robot behavior in the presence of sensor failures.

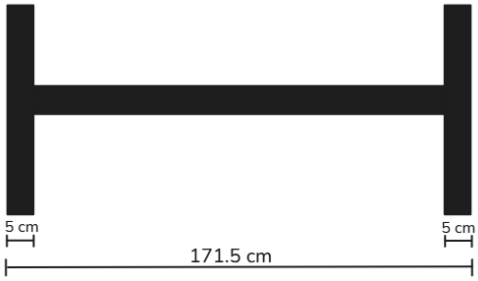


Fig. 1. Top down, scaled up view of the experiment environment with measurements.

II. Experimental Setup

This study goes into detail on the quality that fallback strategies have. The robot will attempt multiple tasks, normally performed with sensors, using fallback strategies. Although not every sensor has the same measurable values, the results of each completed task will be split into categories such as:

A. Measuring the location offset, which is the distance in centimeters (cm) between the actual end-position and the goal-position determined by the baseline coding, which is code that uses a sensor, to determine the accuracy and consistency of fallback strategies.

B. Measuring the time offset, which is calculated by subtracting the execution time the baseline coding has from the fallback code, to determine the lost time upon task completion. Time is measured in seconds.

The testing environment on which the experiments will be held consists of white flooring with black tape stripes forming lines that the robot can follow as seen in figure 1.

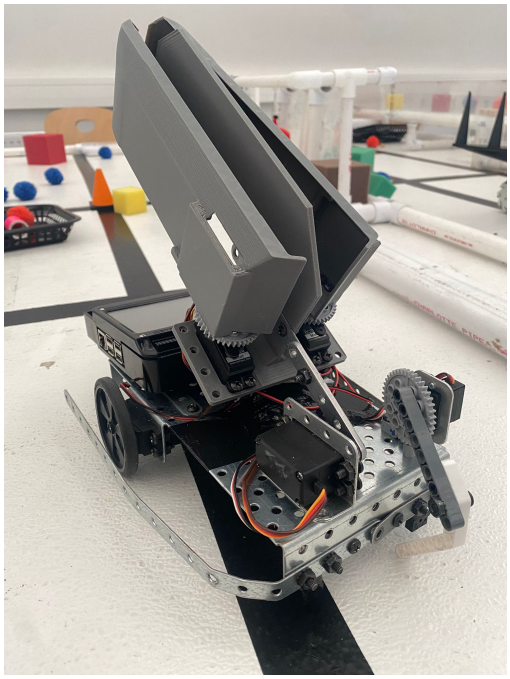


Fig. 2. Image of the robot used.

To get exact values, any obstacles along the robot's path will be removed. Although this setup cannot be fully transmitted into a competition scenario, the goal is to isolate the impact of sensor failure and fallback-logic.

III. Methodology

There are a variety of different sensors used for different purposes, from line or reflection sensors to distance based sensors [1, 2] (see pages 175-230 in reference 1). These sensors also have individual ways of malfunctioning [3]. The experiments will look at a variety of sensors and what happens when a total failure occurs. A total failure, in this case, means either the returned values are constant, losing signal or not having one in the first place. This is done by either deactivating the sensor in the code or giving the sensor variable a constant value [4, 5], creating an artificial jam. The robot used for the experiments can be seen in figure 2. The sensors used for the experiments are the built in gyro as well as the Kipr tophat reflectance-sensor, which will, from now on, be referred to as a "line-sensor" for simplicity.

Sensor failure detection: Fallback strategies are only useful if applied when absolutely necessary. Therefore, implementing ways to detect such failure is a crucial part of the applying fallback strategies properly. However, since every sensor has its unique way of failing so the decision of how to detect sensor failure varies from one sensor to another such as incorrect values over a certain time period, comparing returned values with set constant values, checking if certain values have passed a specific breakpoints

To create more realistic values, which can be used for competition, the experiments will be split into two phases

Phase 1: Testing with total sensor failure to extract raw values and determine the overall impact of sensors. In addition, show how important fallback strategies are in specific scenarios.

Phase 2: Repeat the tests with more realistic sensor failure, such as incorrect values or obstacles that manipulate its abilities to receive values that are more likely to occur during competitions, such as Botball.

IV. 1st Experiment

The gyro is a core-part of a robot's ability to turn [6], however, it is not a perfect sensor and comes with an own set of problems. For this experiment, sensor failure is defined as returning a constant value to artificially create a gyro jam. The extracted values consist of the accuracy measured in percent, and the angular offset measured in degrees.

A. Procedure:

Phase 1: Due to slight slipperiness of the surface and the already existing potential inaccuracy of the gyro, 10 runs will be performed.

Phase 2: In this phase the robot will turn using the built in tick system inside the motors. Now, said motor ticks are individual to each robot, so in this scenario, every time the robot turns it will do so with 350 ticks. Similarly to phase 1, this method will be used in 10 runs as well.

During both phases, the constructed code will be designed to make the robot turn 90 degrees 4 consecutive times creating a circular motion with 4 brief stops, one at each quarter. After both phases are completed, the extracted values of both phases will be compared with each other to determine the safer and more consistent option. In this experiment, instead of comparing to a reference run, the extracted values will be compared to 360 degrees.

B. Anticipated outcomes and anomalies:

During phase 1 it is expected to extract a wider array of values due to the gyro's slight unpredictability. In phase 2 the anticipation focuses on a potential pattern or at least multiple occurrences of the same value, however, there is a chance that the values will be further off compared to the ones from phase 1 if the code is not adjusted properly.

C. Extracted Values and Calculation:

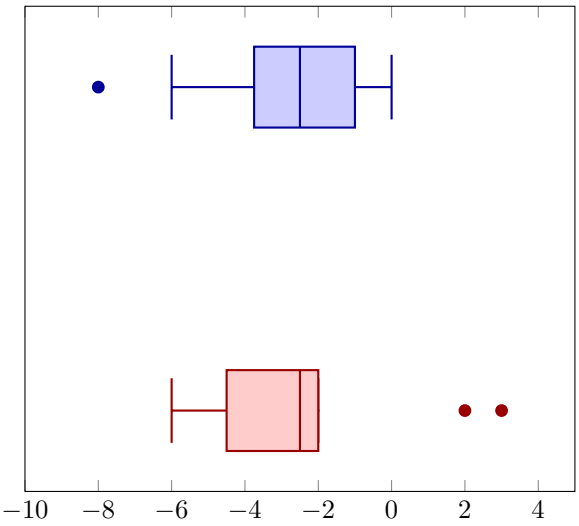
Phase 1: During the experimental runs in this phase, the gyro

showed what was declared as ”fatigue”, because toward the last runs the gyro’s accuracy dropped in comparison to the earlier runs. The software the code runs on is not made to run for longer periods of time. It is suspected that the discovered fatigue came from runtime, for further examination, another 10 runs had been performed except that after the 5th run, the software was restarted. Since the robot cannot be restarted while running in competitions, the exact values that were extracted are not included in this study. However, the average offset with a restart of the robot equals out to -1.6 degrees, this makes a simple restart of the software a solid solution to gyro fatigue. The average accuracy is 99.33 percent, this was calculated using the formula displayed in 3. The average angular offset is actually -2.8 but since the robot cannot have an accuracy above 100 percent the average was made positive for the sake of correct percent calculation.

Phase 2: In this phase, there was no visible pattern of increasing or decreasing values, instead across all 10 runs, there are only 6 unique values with the other 4 being overlaps of those values. There were also two cases where the robot turned too far instead of not enough, indicating an overshooting of the target position. The accuracy of turning with motor ticks is calculated in the same way as in phase 1 and is found to be 99.77 percent with an average angular offset of -2.4 and an average of absolute angular offset of 3.4. To calculate the absolute average, all values were made positive.

The results indicate that motor tick-based turning achieves slightly higher average accuracy than gyro-based turning under the tested conditions. However, this minor improvement may not justify the additional calibration effort required to determine the relationship between motor ticks and turning angles for each robot, in addition to the absolute offset of phase 2 being bigger than in phase 1 making it slightly worse in a theoretical manner. In practice, the gyro remains a reliable solution when functioning correctly, while motor ticks primarily serve as a viable fallback in case of sensor malfunction.

Values of Phase 1 and 2 displayed as Boxplots



Note: The results seen in the graphs above are in degrees and measured in the amount the robot overshot, therefore, negative values mean that the robot did not turn far enough.

$$\left(\frac{\frac{x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10}{10}}{360} \right) \times 100$$

Fig. 3. Formula for calculating the percentage of the average offset in the first experiment.

V. 2ND EXPERIMENT

To get the robot onto the right course towards its designated task, it needs to align itself properly with respect to the Botball game table. This can be achieved via line-sensors and the black lines across the game-table. The extracted values include the time and location offsets of the robot’s ending position. A reference run will be performed, which will be done using two intact sensors to align at a line.

A. Procedure:

During both phases, the robot will drive from one vertical line to the other and aligning at it. There will be 9 runs in both phases with one extra reference run to compare and calculate with.

Phase 1: In this phase, sensor failure is defined as one of two or both sensors being declared as ”dead”, since this creates multiple possibilities, the 9 runs will be split into 3 parts in following order: both sensors dead, left sensor dead, right sensor dead. The robot will drive its path as intended. After a set distance it will stop regardless of finding a line or not to replicate the robot just stopping at a line because no more aligning is needed. For this procedure to work, the motor ticks were utilized again. In this case the robot would stop after 4720 motor ticks had passed. To receive the amount

of motor ticks, the robot drives the distance between both lines, which comes out as 161.5 cm, and after stopping, it will display the amount of motor ticks that had passed after drives. Another big factor for motor tick extraction is the weight of the robot. The robot used for these experiments came out with a weight of around 1 kilogram. To account for natural drift and slipperiness, a few more ticks were added to the output and declared as ”safety ticks”.

Phase 2: During this phase, the sensors will return incorrect values, in this case, the sensors will catch the horizontal line and start aligning. To prevent this from happening, when the sensors find a line and start aligning, the robot will check the amount of degrees it had to turn to align. For this experiment, it was declared that if the robot had to turn 15 or more degrees for alignment, it would add 4 to said degrees and turn that amount in the opposite direction to get back onto the projected path. For research purposes, the robot always started with a slight angle to guarantee the activation of the fallback code.

B. Anticipated outcomes and anomalies:

During phase 1 the time the robot takes to finish aligning is expected to roughly be the same as the reference run while the end position offset have more volatile results. In phase 2 the time offset is anticipated to be further off, due to correcting the robot's path multiple times during a single run, while the ending position should be closer to that of the reference run.

C. Extracted Values and Calculation:

The reference run finished with a time of 7.5 seconds and 3.5 cm of black tape still visible, which is the primary way to determine the location offset.

Phase 1: In this phase, the robot never took more than 1 additional second to end the run (with the longest run, which is run number 7, taking 0.8 seconds longer than the reference run). The downside is that the robot was always at least about 1 centimeter of the target position (with run number 5 being the most accurate with -0.9 centimeter away from the reference run), this is calculated by subtracting the ending position of the reference run from the ending position of each run. The average time it took the robot to finish a run was 8 seconds, which creates an average time offset of 0.5 seconds. For the calculation of the location offset, all negative values were made positive to create the average offset from one side, otherwise the average would be distorted and less realistic. On average, 2.5 centimeters of black tape were still visible in a run, this turns into an offset of -1 centimeter (in this case, negative means that the robot would have had to drive 1 centimeter backward in order to hit the target position). It was also discovered that if only one sensor fails, the other had issues with the horizontal line from time to time, causing the robot to take longer to finish the run.

Phase 2: During this phase, each run took at least 0.8 seconds to finish, with run number 6 being the fastest. While it took the robot much longer to finish runs, it always kept its ending position within 1.1 centimeters to the target location (run number 6 being the farthest away). Overall in phase 2 the robot took an average of 9.3 seconds for one run with an average ending position of 3.8 centimeters away from the black line's border, translating into -0.3 centimeters offset and an accuracy of 91.43 percent.

When comparing both phases with each other, it becomes apparent that the fallback used in phase 2 is the superior one in terms of efficiency. Although the fallback in phase 1 was faster, inaccuracy can influence the path the robot takes in a fatal way. The effort to apply phase 2 fallback is minimal compared to the potential impact it can have when aligning.

	Results Phase 1		Results Phase 2	
Run-Nr.	Finished after	Ending Position	Finished after	Ending Position
1 *(B)	7.7 sec	1.7 cm	9.2 sec	3.7 cm
2 (B)	7.8 sec	0.7 cm	10 sec	3.4 cm
3 (B)	7.8 sec	0.4 cm	8.7 sec	4.1 cm
4 *(L)	8.1 sec	1.7 cm	10.3 sec	3.5 cm
5 (L)	7.9 sec	2.6 cm	9.4 sec	3.7 cm
6 (L)	8.0 sec	6.2 cm	8.3 sec	4.6 cm
7 *(R)	8.3 sec	-2.7 cm	9.9 sec	3 cm
8 (R)	8.2 sec	-5 cm	8.5 sec	4.4 cm
9 (R)	7.8 sec	2.3 cm	9.3 sec	4 cm

*The letters in the brackets indicate, which sensor(s) was/were declared dead.
B = both sensors; L = left Sensor; R = right sensor

Note: The results seen in the table above follow the measurement rule mentioned above. Negative values were received when the robot overshot the black line, and they are measured from the front of the bot to the closest beginning of said line.

VI. Conclusion

General Conclusion:
Knowing when the effort it takes to apply a fallback strategy is actually worth it is a difficult process [7, 8]. It takes proper testing and careful consideration to get the most out of fallback strategies [9]. Below are guidelines to assist in getting optimal usage out of fallback strategies.

- Define the required tasks clearly.
- Identify all expected sensor failures.
- Measure how often the task succeeds.
- Justify why the system should be implemented.

- Evaluate whether the solution is efficient.

Case-specific Conclusion:

This study demonstrates that fallback strategies can significantly improve the robustness of mobile robots when sensor failures occur. Although fallback methods introduce additional execution time, they can reduce positional errors and prevent complete task failure. The results highlight that the effectiveness of fallback strategies strongly depends on the specific task and sensor involved. Future work could investigate more advanced fault-detection mechanisms and adaptive fallback strategies that dynamically select alternative behaviors depending on the detected failure type.

The table below contains a brief summary of the results of both experiments performed for this study.

Task	Sensor used	Fallback useful?	Justification
turning	gyro	No	difference very minor
align at line	line-sensors	Yes	detection of false aligning

VII. Acknowledgment

The authors would like to thank Dr. Michael Stifter and Alexander Lampalzer for providing them with constructive criticism to help improve their work. Another big thanks goes to Mikail Ülger for doing preparatory work for the authors, as well as the members of robo4you for their support in publishing this paper.

References

[1] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer, 2016. https://link.springer.com/book/10.1007/978-3-319-32552-1.

[2] P. Corke, *Robotics, Vision and Control*. Springer, 2017. https://petercorke.com/books/robotics-vision-control-all-versions/.

[3] G.-H. Yang, "Fault detection and diagnosis in robotics," *IEEE Transactions on Robotics*, 2010. https://ieeexplore.ieee.org/document/5438811.

[4] J. Kim and S. Park, "Fail-safe strategies for autonomous robots," in *IEEE International Conference on Robotics and Automation*, 2012. https://ieeexplore.ieee.org/document/6225093.

[5] Y. Liu and H. Chen, "Robust navigation under sensor failure," *Autonomous Robots*, 2019. https://link.springer.com/article/10.1007/s10514-019-09824-1.

[6] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005. https://mitpress.mit.edu/9780262201629/probabilistic-robotics/.

[7] R. Arkin, *Behavior-Based Robotics*. MIT Press, 1998. https://mitpress.mit.edu/9780262011655/behavior-based-robotics/.

[8] R. J. Patton, "Issues of fault-tolerant control for dynamic systems," *IFAC Proceedings Volumes*, vol. 30, no. 18, pp. 219–228, 1997. https://www.sciencedirect.com/science/article/pii/S1474667017394415.

[9] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986. https://jmvidal.cse.sc.edu/lib/brooks86a.html.