Applied Machine Learning in Sensor Calibration - A Clustering Technique

Abigail Liu
Los Altos Community Team 0399
abbyddliu@gmail.com

Aaron Xie
Los Altos Community Team 0399
aaronxie2009@gmail.com

Oliver Jiang
Los Altos Community Team 0399
oliverjiangthepootato@gmail.com

***Abstract*—The reliability of autonomous robots depends largely on their ability to interpret sensors correctly in real time. Similarly, one very important aspect in Botball is being able to analyze the robot's surroundings based on sensor data, which are often noisy and inconsistent due to environmental variability. Sensor calibration is a common approach for generating accurate and reliable sensor readings. However, many of the current calibration techniques we use are not effective enough. In this paper, we evaluate several calibration algorithms and proposed a new clustering-based algorithm, conducting experiments to show that the proposed new algorithm is effective in calibrating sensor data. We also proposed enhancements to this algorithm to handle generic and extreme use cases. In this paper, we use the tophat sensor's identifying black tape as an example for experiments and analysis. However, this solution can be used in any sensor calibration and in any situation that requires clustering such as color, size, distance, etc.**

I. Introduction

The success of autonomous robotics depends on how well a robot perceives and understands its environment. Sensors are a crucial part in allowing robots to learn its position, speed, operation state, etc. The robots then make intelligent decisions based on the data from the sensors. There are many real world examples of robots relying heavily on sensors for operation, such as the self-driving car Waymo[1] and Amazon's Kiva, an Autonomous Mobile Robot (AMR)[2]. However, physical sensors almost never read perfect data. Reading errors can be caused by various factors: environmental impacts such as lighting, humidity, and temperature, manufacturing defects/variations, wearing-out of the sensor, measurement noise, the set-up of the surroundings of the robot, etc[3]. Therefore, it is impossible for sensor reading noise and bias to be eliminated, and the errors will cause the robot to make incorrect decisions if not addressed properly. A widely used approach to solve this problem is sensor calibration, which is common in industries involving autonomous robots such as surgical, self-driving, manufactory, aerospace, and other robots from various areas. The process of "calibration" involves systematic data processing to fix errors, which results in accurate and reliable sensor readings.

When it comes to Botball, there are many places we need to calibrate the sensor data and decide on a threshold. The most common use cases are:

- Tophat sensor: When do we see a black line or a white area?
- ET sensor: When do we see an object?
- Camera: Which color do we read? Is the object large or small?
- Linear Slide: When used to convert analog signals into discrete signals, which discrete signal do we see?

A reliable technique to calibrate sensor readings can benefit all of these scenarios. Among these use cases, identifying the black line is the most critical and frequent scenario in Botball because it is the most effective way for a robot to follow a driving path and identify its location using the lines. In the North California 2025 regional tournament, our team LACT 0399 had a failed seeding run because one of our robots could not find a black line and got stuck, blocking the other robot and causing us to score 7 points. This accident motivated us to investigate this problem. Though we know that this accident was a corner case, one out of thousands of runs, robot reliability is crucial in many ways-both in Botball and professional industries. A single failure could cause a loss both financially and in human lives. We aim to improve the reliability of our robots as much as possible so that in real life we can achieve higher safety and security. In the rest of this paper, we will focus on the process of calibrating the tophat sensor to identify "BLACK", in addition to analyzing various techniques and identifying the optimal solution. As mentioned earlier, the same technique can be applied to the ET sensor, camera, linear slide, and other sensors as well.

II. Experiment Setup

First, we tried to analyze the accident. One thing many of our team members noticed was the lighting, as it was not as bright as the lighting on the table we used during practice. However, the way a tophat sensor works is that it emits infrared (IR) light to generate readings, which is not affected by lighting[4]. Therefore, lighting should not be an issue and will not be a part of this experiment.

The next possibility is the sample data used to calibrate the robot. The samples may not be evenly distributed between "BLACK" and "WHITE". When the samples are extremely skewed, it could affect how the threshold of "BLACK" is determined. If the threshold for "BLACK" is set too high, the robot may not read as high of a value as the threshold during the run, which causes it to miss the black line.

A. Experiment configuration

We plan to conduct experiments for all the calibration techniques discussed in this paper. The experiment is set up in the following configuration:

- Data sample size: 200
- Repeat the following experiment for 10 times:
  - Repeat following steps 100 times:
    1) Calibration run: Run through one black line on the white surface and calibrate the tophat sensor, then drive back to the base line.
    2) Test run: Run through the same black line again, reporting whether a black line is seen.
  - Calculate the success rate of identifying BLACK and output the sample data logs.

Each experiment contains 100 repetitions, and each repetition contains a calibration run and a test run. We then repeat the experiment 10 times. We do this instead of running the experiment 1000 times because we want to output all of the data before our robot runs out of battery. We need the sample data for analysis. In fact, we observed similar results from all 10 experiments, so we randomly chose one experiment data set to analyze in the rest of this paper.

The code snippet is shown below:

```
    double cnt_black = 0;
    double cnt_runs = 100;
    for (int i = 0; i < cnt_runs; i++) {
        //calibrate_tophats();
        calibrate_tophats_w_clustering();

        // drive back to base
        drive(-1500, -1500);
        msleep(2000);
        drive(0, 0);

        int saw_black = 0;
        int read_rate = 10;

        // Drive for 2 seconds, read tophat every
        // 10ms and check whether it's BLACK
        drive(1500, 1500);
        double start_time = seconds();
        while (fabs(seconds() - start_time) < 2) {
            if (seeBlackAlg1() > 0) {
                saw_black++;
            }
            msleep(read_rate);
        }
        drive(0, 0);
        if (saw_black) {
            cnt_black = cnt_black + 1.0;
        }

        // drive back to base
        drive(-1500, -1500);
        msleep(2000);
        drive(0, 0);
    }

    printf("saw black %f\n", cnt_black/cnt_runs);
```

Listing 1: Experiment code

B. Metrics

We will evaluate all the algorithms in the next two metrics and compare their effectiveness.

- Success rate
- Graph showing the identified threshold among all sample data

III. Calibration Algorithms and Experiment Results

A. Calibration algorithm 1: 90%ile sample

Our previous method (which we used for the 2025 NC regional tournament) for calibration involved collecting raw sensor readings over a fixed interval and sorting the data. Then we would take the 90th percentile of the data to eliminate the outliers in the data to calculate the threshold to determine whether the sensor has read a "BLACK" value. Below is a code snippet to determine whether we see BLACK:

```
int seeBlackAlg1() {
    if (analog(LEFT_TOPHAT) >= BLACK_VAL) {
        return 1;
    }
    else {
        return 0;
    }
}
```

Listing 2: Algorithm 1 of determining BLACK

Fig. 1 below visualizes how the BLACK threshold is set.

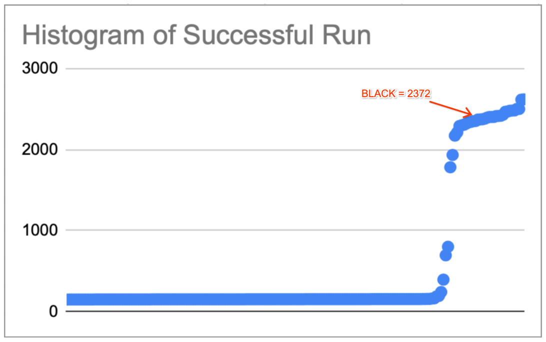


Fig. 1: Sample data of a successful run.

This algorithm is simple and intuitive, and works the majority of the time. However, the threshold set by this algorithm is vulnerable to sample data noise. Out of our 100 test runs, the threshold shows a large fluctuation (in Fig. 2).

When the sample data set is significantly skewed (which is what happened during the NC regional tournament), the threshold may be too high or too low and could cause the robot to miss a black line or read "BLACK" when there is not actually a black line.

As shown in Fig. 3, this algorithm had a success rate of 92% in this experiment.

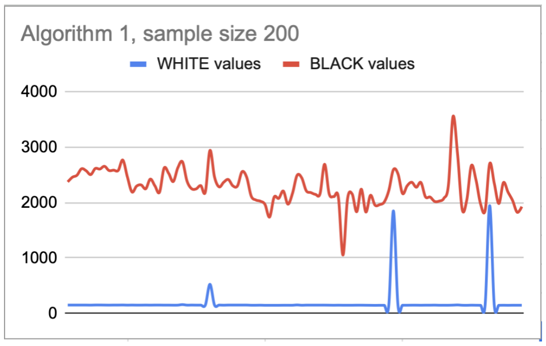
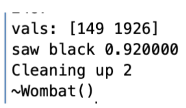


Fig. 2: Algorithm 1: calibrated values for WHITE and BLACK.



```
vals: [149 1926]
saw black 0.920000
Cleaning up 2
~Wombat()
```

Fig. 3: Experiment result for Algorithm 1: succcess rate

B. *Calibration algorithm 2: median of the 80%ile data range*

The second calibration method that we came up with after the regional tournament was to use the middle 80% of the data. We would first sort all the sample data in ascending order, define white and black as the 10th percentile and the 90th percentile respectively, then average "WHITE" and "BLACK" to calculate the threshold, as shown in the following code.

```
int seeBlackAlg2() {
    int read = analog(LEFT_TOPHAT);
    if (fabs(read - WHITE_VAL) >= fabs(read -
    BLACK_VAL)) {
        return 1;
    }
    else {
        return 0;
    }
}
```

Listing 3: Algorithm 2 of determining BLACK

Fig. 4 shows how Algorithm 2 sets the threshold. In the same experiment run, Algorithm 2 sets the threshold to 1262 instead of 2372 used by Algorithm 1.

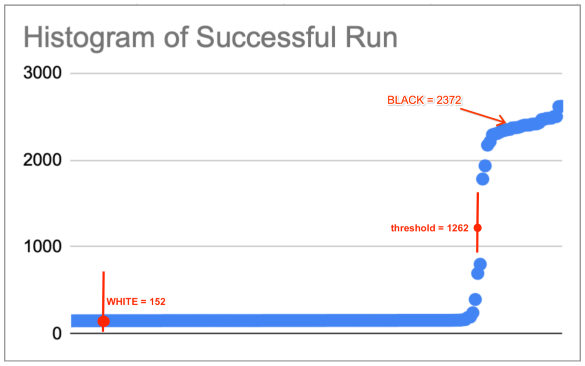


Fig. 4: Algorithm 2 chooses a different threshold in the same sample.

Compared to the previous algorithm, this algorithm is more accurate, since averaging the two values lowers the threshold and gives us more room for error. Fig. 5 demonstrates how Algorithm 2 is more effective than Algorithm 1 and avoided some super noisy data samples. However, Fig. 5 shows when the sample data is extremely skewed, this algorithm still has the same problem as the previous algorithm, as an error in the sample data may cause an unproportionate threshold value. This algorithm had a 98% success rate in our experiment.

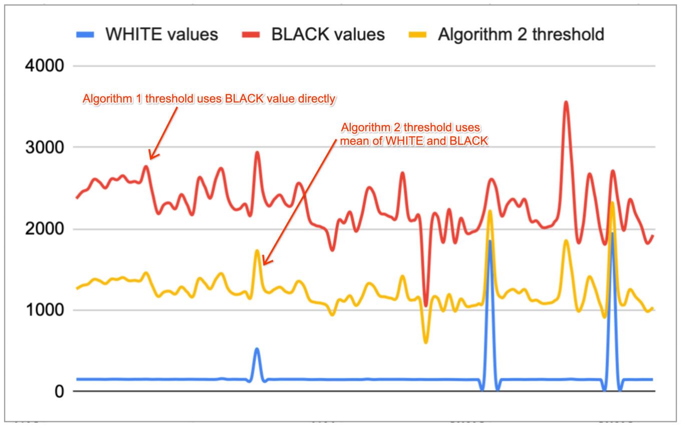


Fig. 5: Algorithm 2: using mean of WHITE and BLACK vs using BLACK only.

C. *Calibration algorithm 3: K-Means clustering*

The final calibration method we came up with is clustering, a classic Machine Learning technique. We would cluster all sample data into two groups, one representing WHITE and one representing BLACK. Then we would take the mean value of each cluster as the value for WHITE or BLACK, respectively. After identifying WHITE and BLACK, the algorithm for creating the threshold was the same as Algorithm 2(taking the mean of WHITE and BLACK). Compared to the percentile method, clustering is more accurate, as it is not affected by how data is distributed between the two groups(WHITE and BLACK), thus it can handle the skewed data better.

There are many existing clustering algorithms, such as K-Means[5], Hierarchical Clustering[6], Density-Based Spatial Clustering[7], Mean Shift Clustering[8], etc. Since we only have 1-dimensional data and two clusters, Ikotun's literature review[9] suggested that the K-Means algorithm would be the simplest and works the best for our use case.

The K-Means algorithm starts with two centroids and repeatedly assigns each data point to the closest centroid until convergence. The cluster with the larger mean becomes "BLACK" and the cluster with the smaller mean becomes "WHITE". The following code is the version of K-Means that we used in this experiment.

```
int K = 2;
while (changed && iter < MAX_ITER) {
    changed = 0;

    // Assign each data point to the closest
    centroid
    for (int i = 0; i < size; i++) {
        int dist0 = abs(data[i] - centroids[0]);
        int dist1 = abs(data[i] - centroids[1]);

        int new_label = (dist0 < dist1) ? 0 : 1;

        if (labels[i] != new_label) {
            labels[i] = new_label;
            changed = 1;
        }
    }

    // Recalculate centroids
    float sum[K] = {0.0};
    int count[K] = {0};

    for (int i = 0; i < size; i++) {
        sum[labels[i]] += data[i];
        count[labels[i]]++;
    }

    for (int k = 0; k < K; k++) {
        if (count[k] > 0) {
            centroids[k] = (int)(sum[k] / count[k]);
        }
    }

    iter++;
}
```

This method is much more accurate than the other two in small sample data sizes, where there is a greater chance of error. As shown in Fig. 6, both the percentile technique and the clustering technique generate similar curves in all three metrics: WHITE, BLACK and the threshold. In fact, the majority of time, all three algorithms work the same, though Algorithm 3 has smoother curves in all metrics compared to the other two algorithms. That means that Algorithm 3 is more stable and reliable, in addition to handling several extreme cases very well, achieving a 100% success rate in our experiments.

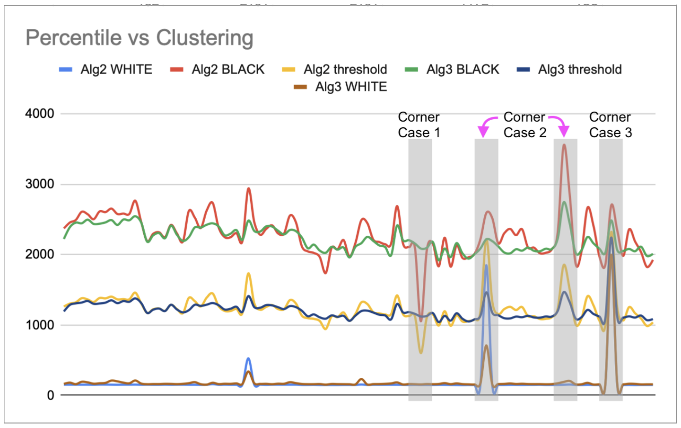


Fig. 6: Algorithm 3: use K-Means to cluster WHITE and BLACK, then use the mean as threshold.

Taking a closer look at Fig. 6, we can see that in several corner cases where the sample data was significantly skewed, Algorithm 3 out performed the other two algorithms. We will analyze these corner cases in details in the next section.

IV. Corner Case Analysis

A. *Corner Case 1: data is largely skewed towards WHITE*

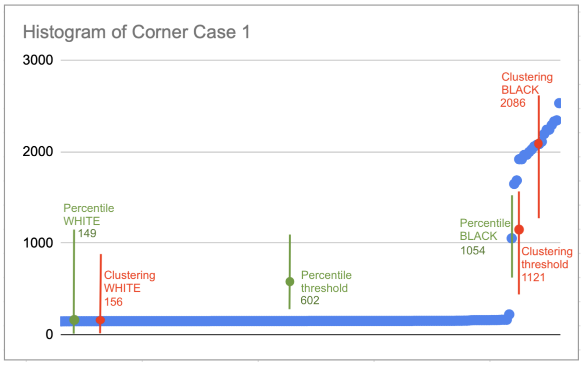
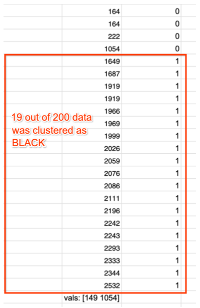


Fig. 7: Corner Case 1: data is skewed towards WHITE

The sample data in this case has 90% WHITE and 10% BLACK (Fig. 7). That happened to place the 90%ile data at a low reading of 1054, which caused the percentile threshold to become 602. This error could cause the robot to read a false BLACK on a grey area instead of a black line.

In contrast, the clustering technique was able to identify the 19 data samples of BLACK (Fig. 8), and only use these 19 data points to calculate the BLACK value. As a result, the clustering technique generated a reasonable threshold of 1121.

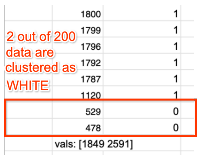


	164	0
	164	0
	222	0
	1054	0
	1649	1
	1687	1
	1919	1
	1919	1
	1966	1
19 out of 200 data was clustered as BLACK	1969	1
	1999	1
	2026	1
	2059	1
	2076	1
	2086	1
	2111	1
	2196	1
	2242	1
	2243	1
	2293	1
	2333	1
	2344	1
	2532	1
	vals: [149 1054]	

Fig. 8: Corner Case 1 data

B. *Corner Case 2: data is largely skewed towards BLACK*

The sample data set in this case was extremely skewed, with only two WHITE data points (Fig. 9).



	1800	1
	1799	1
2 out of 200 data are clustered as WHITE	1796	1
	1792	1
	1787	1
	1120	1
	529	0
	478	0
	vals: [1849 2591]	

Fig. 9: Corner Case 2 data

The percentile technique ignored the two WHITE data points because they fell out of the 10% percentile and were eliminated. That caused both the values of WHITE and BLACK to be evaluated to BLACK values, which further

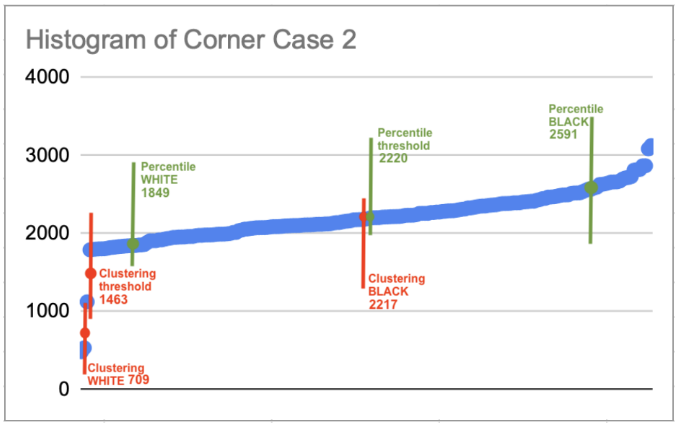


Fig. 10: Corner Case 2: data is skewed towards BLACK

caused the robot to output a very high threshold of 2591 for Algorithm 1 and 2220 for Algorithm 2.

Both Algorithm 1 and Algorithm 2 failed to find a black line during the experiment, while the clustering technique successfully isolated the WHITE data from the rest data and calculated a reasonable threshold of 1463 (Fig. 10).

C. Corner Case 3: invalid sample, only BLACK or only WHITE

We found one experiment run in which all algorithms failed to find a black line (Fig. 11). All of sample data points were BLACK, thus neither the percentile nor clustering methods could identify WHITE and generate a valid threshold. We decided that this was not a valid run, and should not be counted towards the success rate of each algorithm.

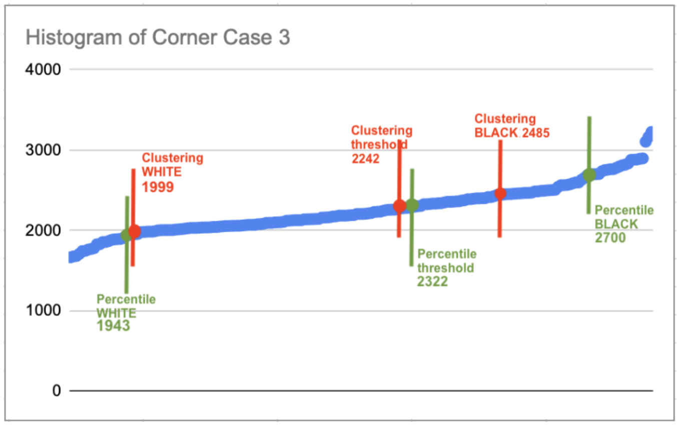


Fig. 11: Corner Case 3: invalid sample, only BLACK or only WHITE

V. Compare all three calibration techniques

A. Reliability

Based on the calibration techniques and their graphs, we can compare each algorithm to determine the most effective approach for identifying the threshold between WHITE and BLACK values.

Algorithm 1, which uses the 90th percentile of the BLACK values directly as the threshold, is the simplest and fastest method. However, its performance is highly sensitive to outliers in the BLACK data, and as shown in Fig. 10, this method selects a threshold that is too high and causes the calibration to fail. While it may perform well in ideal conditions, such as the successful run shown in Fig. 1, it becomes unreliable when sensor data is noisy or lacks a distinct difference between the two values.

Algorithm 2 is a refined version of Algorithm 1, as it averages the mean values of WHITE and BLACK data to determine the threshold. This method allows for a much larger margin of error, as the threshold it chooses is much more evenly spread between the two values. However, this method still assumes a relatively clean input and can be skewed if either WHITE or BLACK values are inconsistent or contain significant noise.

Algorithm 3 is much more efficient, especially in data sets with lots of noise, since it clusters the values without relying on a pre-sorted data set. This reliability is demonstrated in Fig. 6, as there are no extreme variations in the data and the difference in the data values are all very distinct. Thus, Algorithm 3 is the most flexible and accurate of the tested approaches.

Table I shows the comparison among the three algorithms in terms of their success rate of identifying BLACK. Among them, all of the algorithms have a success rate over 90%. Algorithm 2 is substantially more accurate than algorithm 1, though it still cannot handle the corner cases. On the other hand, Algorithm 3 is much smoother and resistant to noisy data, which allows it to be more reliable and able to withstand the corner cases.

Besides the performance of identifying BLACK, there is another risk at opposite direction: false BLACK. When data is extremely skewed towards WHITE (corner case 1), both algorithm 1 and 2 read a false BLACK while algorithm 3 does not.

Algorithm	**Success Rate**	**Risk for False Black?**
Algorithm 1	92%	Yes
Algorithm 2	98%	Yes
Algorithm 3	100%	No

TABLE I: Comparison of 3 calibration algorithms

B. Complexity

The percentile technique (Algorithm 1 and 2) needs to sort all the sample data read by the tophat sensor. We used Quick Sort in our robot. The time complexity of a Quick Sort is $O(n \cdot Logn)$[10], where n is the data size. Cormen suggested in his book that the time complexity of Quick Sort would not change even when the data had already been sorted.

The time complexity of the K-Means algorithm is $O(n \cdot i)$, where i is the number of iterations until convergence. In our case, the robot uses tophat sensor to read data from WHITE to BLACK, then WHITE, or vice versa. There is a pattern in the sample data, which means our sample data is already semi-sorted. The K-Means algorithm converges much faster using semi-sorted data[11], typically within 5 iterations. This means that when n is large, the effective time complexity is $O(n)$, so the clustering technique also has better efficiency than the percentile technique in terms of running time.

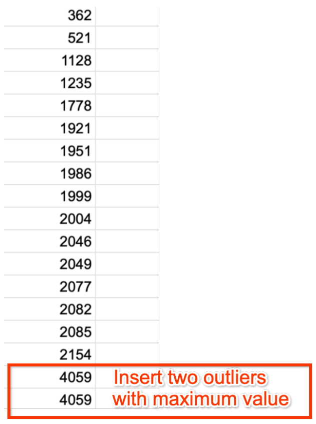
VI. Enhancements and Future Directions

In the above sections, we used our experiment results to analyze all three calibration techniques. However, there are cases that our experiment did not capture, but could possibly occur. We call them theoretical errors. One theoretical error is when the sensor reads faulty data, which could either be extremely high or extremely low. We do not worry much about extremely low data, because normally a WHITE reading is between 150-200, which is close to the minimum value 0 already. A faulty "0" reading will not skew the data that much. The other faulty possibility, an extremely high value, could cause two possible issues:

1) Re-clustering issue: The faulty high values become the BLACK cluster, pushing all the other valid data into the WHITE cluster.
2) The faulty high values are included in the BLACK cluster and raise the average value of the cluster, which further raises the threshold of BLACK.

A. *Re-clustering Issue*

It is hard to make the sensor generate faulty sensor readings. We then took some of our experimental sample data, and deliberately inserted some fake outliers. We want to test whether the outliers would cause the BLACK and WHITE be clustered together. Fig. 12 is an extreme example with low BLACK values, so that we have a large distance between the outliers and the BLACK samples. However, in all of our experiments, the outliers never caused the re-clustering issue, even in the extreme example Fig. 12.



362
521
1128
1235
1778
1921
1951
1986
1999
2004
2046
2049
2077
2082
2085
2154
4059
4059

Insert two outliers
with maximum value

Fig. 12: Insert 2 outliers

According to KIPR[12], the value range of tophat sensors is [0, 4059]. Thus, the current K-Means method we use can work for this scenario because the greatest possible value the sensor can read is not large enough to cause re-clustering. This is shown in Fig. 13 where we inserted outliers at the maximum sensor range.

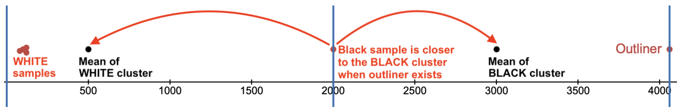


Fig. 13: Outliers cannot push BLACK data into WHITE cluster

Even though this is not an issue for the tophat sensor, it may become a greater issue for calibration of other sensors, where the maximum value is much greater than the typical high value, hence we still need to handle this issue. One way to solve this issue is to incorporate a data validation step in the process. We do this by restricting the value range of the data set(which we hardcode) such that all the outliers in the sample data are removed, as shown in the code below.

```c
if (read >= MIN_VALID && read <= MAX_VALID) {
    tophat[valid_count] = read;
    valid_count++;
}
```

Listing 4: Data validation

B. *Faulty High Threshold Issue*

When an extremely high faulty value(an outlier) is included within the BLACK cluster, it directly increases the cluster's average value, which will consequently raise this threshold.

In our new code(shown below), we calculate the value of "BLACK" by taking the median of the cluster instead of the mean. As shown in Fig. 14, using the mean of the data set results in a value of 2232, which is higher than all of the accurate black values. On the other hand, using the median gives us a reasonable value of 2046.

However, this new method has a drawback in that finding the median of the data set requires sorting of the sample data, which will increase the time complexity.

```c
// Determine BLACK_VAL and WHITE_VAL
int count_black = 0;

// Find the label for BLACK
int black = (centroids[1] > centroids[0]) ? 1 : 0;

// Count the BLACK sample data
for (int i = 0; i < size; i++) {
    if (label[i]) == black) count_black++;
}

// Sort all sample data
qsort(tophat, SAMPLE_SIZE, sizeof(int), compare);

// Calculate the indices of the median BLACK and
    median WHITE
int black_median = (SAMPLE_SIZE - count_black) +
    count_black / 2;
int white_median = (SAMPLE_SIZE - count_black) / 2;

BLACK_VAL = tophat[black_median];
WHITE_VAL = tophat[white_median];

printf("vals: [%d %d]\n", WHITE_VAL, BLACK_VAL);
```

Listing 5: Code determining values of BLACK and WHITE

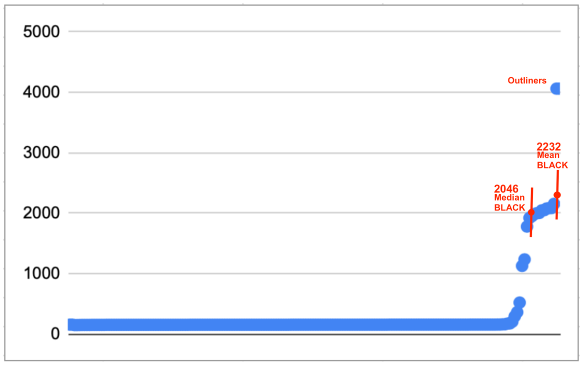


Fig. 14: Hyrbrid approach generates better BLACK threshold

C. *Other Enhancements*

A possible enhancement to the current K-Means clustering technique involves improving how the initial centroids are chosen. Our current K-Means algorithm sets initial centroids using the first two sample data. If we can set the initial centroids to the smallest and largest values in the dataset, the algorithm can converge much faster. Since the sensor data typically follows a WHITE-BLACK-WHITE pattern as the robot crosses a line, we can set the initial centroids to the values in the beginning and middle, respectively. This enhancement reduces the number of iterations needed to converge and increases the efficiency of the code.

VII. Conclusion

In this paper, we identified the issue behind errors in previous methods of calibration, experimented with three different methods, and proposed a simple machine learning-based calibration technique. Through various experiments, we showed that this K-Means clustering method outperformed other techniques, and can achieve 100% success rate of identifying “BLACK” from “WHITE”.

The clustering technique proposed in this paper can be applied in various analog sensors and use cases. For example, the K-Means algorithm can identify more than two clusters. It can also be used to calibrate the camera readings, such as identifying different colors or seeing whether the object we detect is a large building or a small building (depending on the number of pixels). In future work, we will apply this algorithm in other areas of Botball and evaluate its effectiveness.

To further improve the sensor calibration, we can consider using more advanced machine learning techniques such as linear regression to cluster the data. Unlike the other threshold-based methods, machine learning models can take more inputs (called features to the model), such as robot driving speed, and achieve more accurate calibration. This approach can allow for more flexibility that has the potential to significantly benefit the accuracy of our robots.

References

[1] S. Thrun, ”Toward robotic cars,” *Communications of the ACM*, vol. 53, no. 4, p. 99, Apr. 2010, doi: https://doi.org/10.1145/1721654.1721679.

[2] G. Fragapane, R.D. Koster, F. Sgarbossa, J. O. Strandhagen, ”Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda.” *European Journal of Operational Research*. 294. 10.1016/j.ejor.2021.01.019.

[3] T. Nguyen, Y. Shin, J. Kim, and D. Kim, ”Signal Detection for Ambient Backscatter Communication with OFDM Carriers,” *Sensors*, vol. 19, no. 3, p. 517, Jan. 2019, doi: https://doi.org/10.3390/s19030517.

[4] A. Xie and A. Liu, ”Sensing the World: A Deep Dive into the Different Methods of Interacting with the Environment Around Us”, The Global Conference on Educational Robotics (GCER), 2024.

[5] J. MacQueen. ”Some Methods for Classification and Analysis of Multivariate Observations.” *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[6] J. H. Ward Jr. ”Hierarchical Grouping to Optimize an Objective Function.” *Journal of the American Statistical Association*, 1963.

[7] M. Ester, H.P. Kriegel, J. Sander, X. Xu. ”A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.” *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996.

[8] Y. Cheng. ”Mean Shift, Mode Seeking, and Clustering.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1995.

[9] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija, and J. Heming, ”K-means Clustering Algorithms: a Comprehensive Review, Variants Analysis, and Advances in the Era of Big Data,” *Information Sciences*, vol. 622, no. 622, Dec. 2022.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. ”*Introduction to Algorithms*,” 3rd Edition. MIT Press, 2009.

[11] D. Arthur and S. Vassilvitskii, ”k-means++: The Advantages of Careful Seeding.” *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007.

[12] KISS Institute for Practical Robotics (KIPR), “Analog Sensors - Reflectance,” Professional Development Workshop 2025.