# Laying the Foundation for Scoring FTC Games Using Computer Vision

Utku Melemetci

August 2022

## Abstract

FTC games require a lot of volunteer personnel to score. I present foundational tools to automate scoring. Line detection and color thresholding is used to detect the field. Color thresholding and Hough circle transforms are used to detect the alliance and shared shipping hubs. Game elements are detected using color thresholding and clusters are separated using a connected object separation method. Image normalization techniques are applied throughout. In the end, the field and shipping hub detectors performed well, however the game element detector did not perform well enough for use in the real world and will likely need serious modifications. I also present potential avenues for improvement.

## Contents

# 1 Introduction

One of my favorite pastimes throughout the school year is being a part of my school's robotics team.* We compete in the FIRST Tech Challenge. FIRST (For Inspiration and Recognition of Science and Technology) is a global non-profit that runs numerous robotics competitions, the FIRST Tech Challenge (FTC) being one of them. In FTC, teams compete at events to move up the event hierarchy. Ranking at an event is based on the team's win to loss ratio. Each match consists of four robots, two per "alliance." Alliances are determined randomly and teams on an alliance are to work together to maximize their score.

In a match, teams earn points by completing tasks. Although the game and therefore the tasks change each year, common tasks include moving "game elements," usually plastic pieces of some kind, from one location to another. In the 2021-2022 season, the game was called Freight Frenzy. One of the main goals in Freight Frenzy is to move plastic cubes, wiffle balls, and rubber ducks (collectively called "freight") to one of several locations, including the alliance shipping hub, the storage unit, and the shared shipping hub. A labeled diagram is provided below.

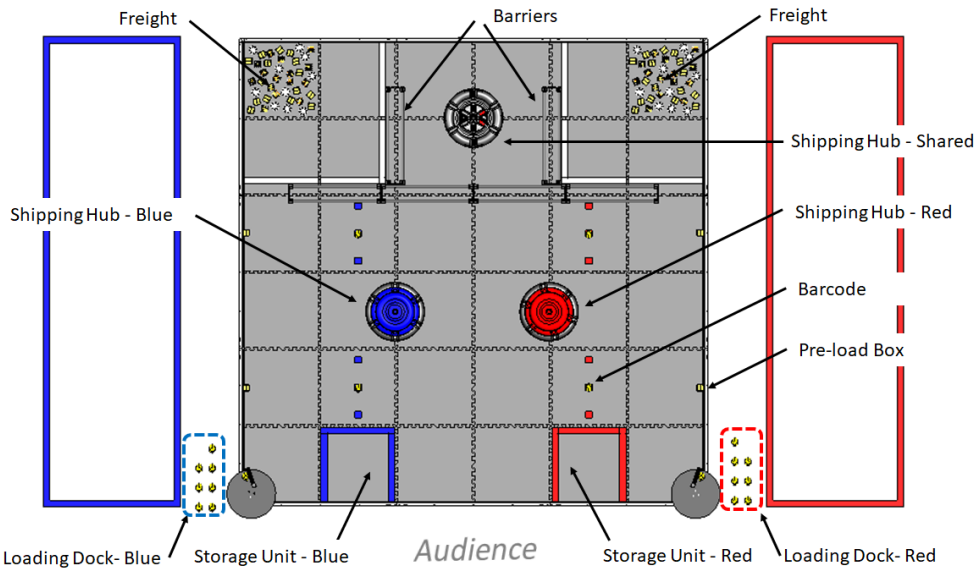



Figure 1: Bird's eye view of the field (FTC Organizers, 2021)

*All code used for this project is available at https://github.com/Yey007/ftc-score

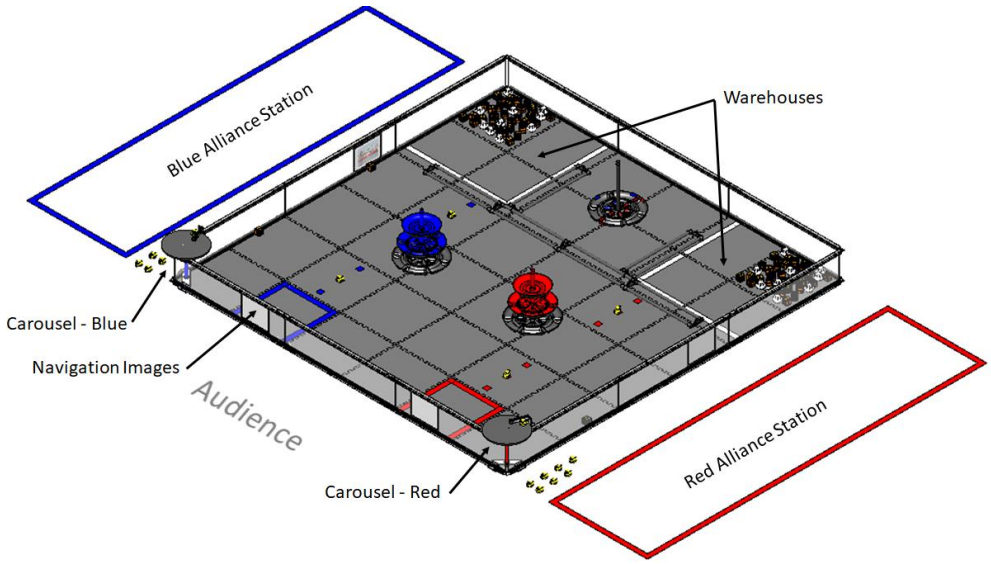



Figure 2: Isometric view of the field (FTC Organizers, 2021)

There are other tasks in Freight Frenzy, but moving freight is the primary one and, crucially, the hardest for scorekeepers and judges to keep track of amidst the chaos of the game. Up to three or four judges and scorekeepers are usually required for each match. If the necessary number of volunteer judges and scorekeepers can be reduced, lower tier qualifiers that otherwise would have struggled to find enough manpower can have an easier time running, and FTC can be made accessible to more areas and people. Additionally, scoring errors, which occur at even the highest tier FTC events, can be minimized.

For this reason, I propose foundational tools for automating the scoring of FTC games. Efforts throughout this paper will be focused on scoring freight in the shared shipping hub due to generally more favorable lighting and occlusion conditions, but the alliance hubs are also detected and tracked. It should be made clear that the goal of this paper is not to provide a complete solution for scoring Freight Frenzy or any other FTC game, but rather to provide foundational components and a proof of concept. Many components, such as the field detector and preprocessing techniques are written in a game-invariant fashion as to facilitate future research. I discuss the preprocessing techniques and detection methods used in sections three and four, and the strengths and weaknesses of said methods in section five. I also present avenues for research in future work.

There are four main components to the program presented: data collection, field detection, shipping hub detection, and game element detection. I collected data from two livestreams, one being the Wisconsin Knack Attack Qualifier livestream (FTC Wisconsin, 2021) and the other being the Oregon State Championship livestream (RevAmped Robotics, 2021). This paper contains still and processed images from these livestreams. All images of matches are from these livestreams. The field detector uses line detection to find the edges of the field. It then computes roughly where the corners of the field should be by looking at the intersection of the segments. The corners are then used to apply a perspective transform. This is not always fully accurate, so a color-based refinement method is used to draw a bounding rectangle around the grey section of the field.

The shipping hub detector is comprised of two smaller algorithms: the alliance shipping hub detector and the shared shipping hub detector. The alliance hub detector uses the distinctive color of the alliance hubs to draw bounding boxes, while the shared shipping hub detector relies on the circular shape of the shared shipping hub.

The game element detector uses thresholding techniques to find yellow cubes. Clusters of these cubes are recursively split apart using a separation method based on convexity defects. The method finds concave points on the outline of the cluster of cubes to determine a line that separates one cube from another.

In general, I find that the field and hub detectors work well even under less-than-ideal conditions with occlusions and lighting differences. I will still discuss a number of failure cases where improvements can be made. The game element detector is a slightly different story. Though it performs well with small clusters, it has a lot of trouble separating larger clusters and clusters where convexity defects are not clear. I suggest potential fixes and improvements to all three parts in future work.

## 2 Notation and Assumptions

Some points concerning the notation used and assumptions made throughout the paper are described here to improve clarity.

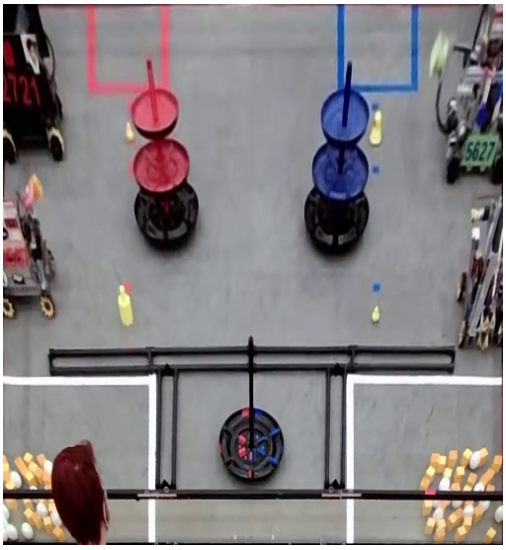
- Anything `in this font` is either code or closely related to code (filenames, version numbers, etc.)
- The OpenCV Version used is `4.5.4.60`
- Images are in BGR with a range of [0, 255] per channel unless a conversion is stated.
- HSV images have hue channels with a range of [0, 180), saturation channels with a range of [0, 255], and value channels with a range of [0, 255]
- Dimensions are always in $width \times height$ or $x \times y$
- When finding contours, `cv2.RETR_EXTERNAL` is always used to only find external contours

## 3 Related Work

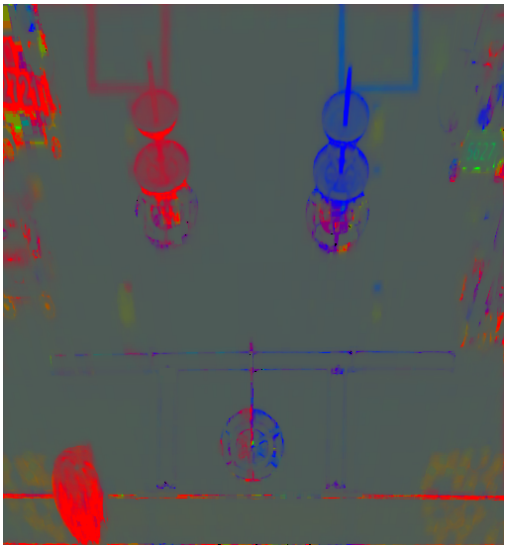
Little research has been completed specifically in the area of scoring FTC games. However, many generic algorithms and techniques can still be applied. One key area of research was image normalization. FTC events, although commonly held in gyms, have no standard environment. From livestream to livestream, lighting, color tint, and other factors can vary wildly. Therefore, I explored many normalization algorithms to deal with different conditions.

Comprehensive Colour Image Normalization (Finlayson et al., 1998) was attractive for a few reasons. Most other color normalization methods focus on correcting either differences in images of identical scenes due to lighting color or the positioning of lighting (or lighting geometry), but usually not both. Normalizing for lighting color ensures that two identical objects or scenes, each under a different color of light, look identical after normalization. Normalizing for lighting geometry ensures that two identical objects or scenes, only under lights in different positions (e.g. one lighted from above and one from the side) look identical after normalization. In contrast to most methods, this method is "comprehensive" and corrects for both lighting color and geometry. Furthermore, it is incredibly simple. The algorithm iteratively normalizes for lighting color and then lighting geometry using existing methods until the image converges. The authors prove mathematically that the image must converge and that the converged image is unique, i.e. that the result is unique to that scene and that the same scene under any lighting conditions will produce the same result. They verify their proof experimentally and find that convergence usually takes four to five iterations.

In my own experiments, I found that overall, the method delivered what was promised. Although the resultant image lacked a lot of details, making comprehensive normalization unsuitable as a generic normalization step to be run on every frame, the algorithm successfully unified the gray of the field to almost a single color, making it suitable for use in field identification.

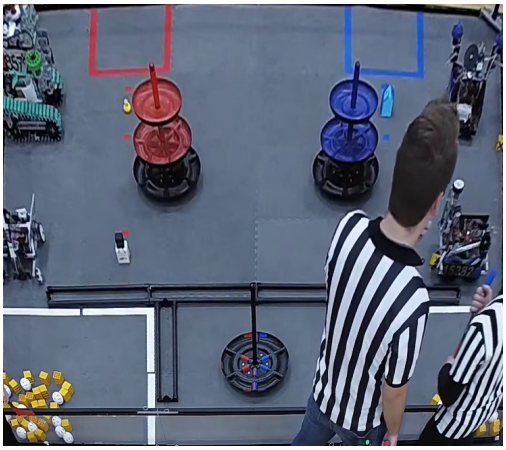


(a) Before comprehensive normalization



(b) After comprehensive normalization

Figure 3: Frame from an Oregon video before and after comprehensive normalization



(a) Before comprehensive normalization

(b) After comprehensive normalization

Figure 4: Frame from a Wisconsin video before and after comprehensive normalization

Another paper I utilized was "ALCN: Adaptive Local Contrast Normalization" by Rad et al., 2020. Though their approach to normalization was neural network based and likely too complex for the purposes of this paper, they also provided a survey of previous methods which was incredibly valuable. Local Contrast Normalization (LCN), one of the techniques ALCN borrowed from, was simple to implement given the formula and performed well. Although LCN leaves the image without much detail, much like the comprehensive normalization method, it does an admirable job highlighting and unifying bright colors like the red and blue used for the alliance shipping hubs as seen in Figure 5. Therefore, LCN was utilized in detecting these hubs.

To keep track of the hubs after detecting them, I experimented with the object tracking algorithms provided by OpenCV. CSRT, called CSR-DCF (Discriminative Correlation Filters with Channel and Spatial Reliability) in the original paper, is one such tracker. As the name suggests, the tracker builds on correlation filters.

A known object's position can be "estimated as the location of the maximum of correlation response," (Lukežič et al., 2018) where the correlation response is determined by the sum of the correlations between a number of feature channels and target templates. Correlation is computed using the correlation operator, an operator much like the convolution operator. The feature channels can be many things, from a greyscale version of the image to histogram of gradients (HoG) channels. Their algorithm prescribes a set of features to use. Since the object is known, or rather provided to the tracker on initialization, the targets for these channels can easily be computed. The improvements the authors make on this base algorithm are twofold. First, they generate a spatial reliability map to determine the places most suitable for tracking to extract those features. Second, they estimate reliability scores for each of those channels based on their maximum correlation outputs, where high values indicate high reliability, and how strongly channels "vote for" or correlate with a single location, where correlating strongly with fewer locations indicates high reliability. CSRT (or CSR-DCF) was chosen because it ran quickly, had relatively good accuracy and was practical to use as it was already available in OpenCV.

Figure 5: Frame from an Oregon video normalized with LCN

Tracking was not necessary for game elements like the yellow cubes, but normalization was beneficial in detecting them. For this task, the normalization by standardization approach described in Rad et al., 2020 was used. Standardization was applied on a per channel basis. The yellow was not particularly highlighted like the blues and reds using LCN, but the blue tint was more or less removed from one of the livestreams, which made thresholding easier. Simple thresholding, however, was not enough to count the number of cubes in an area since they tend to clump up and appear as a solid blob. To deal with this, an algorithm similar to that described by Jing et al., 2015 is implemented. The paper is based around object detection in a competition similar to FTC called VEX.

The authors propose two methods: one for object detection and one for connected object separation. The object detection is of little interest as I am not dealing with transparent objects as the authors were. The connected object separation algorithm, however, is far more interesting. Convexity defects are used to detect where objects connect and thus where to split them. The example of colored spheres from the 2013 VEX game is used. The algorithm can work with two or more connected objects, although it appears to be designed to work best when objects are touching each other in a line.

Figure 6: An example of connected objects separation from Jing et al., 2015. The second image shows convexity defects being found, and the third image shows the objects being separated.

A similar, but not identical algorithm is used in this paper. The new algorithm uses convexity defects as well, but performs filtering on the found defects based on distance from the convex hull. Additionally, it is recursive, splitting objects at the "good" defects closest to each other until no more good defects are left. Not performing filtering on the ratio of the rectangle like the original work also allows it to split clusters of objects, not just those in a line. As I will demonstrate, however, this method may not be fully suited for this task.

# 4 Using Computer Vision to Score FTC Games

## 4.1 Data Collection

I downloaded two livestreams of two FTC events from YouTube using the open-source tool youtube-dl.[†] One was the Oregon State Championship, held on Sunday, March 13th, 2022, and the other was the Wisconsin Knack Attack Qualifier, held on Saturday, January 29th, 2022. The videos were selected based on availability, but coincidentally had some nice properties. The Oregon video was quite easy to work with, while the Wisconsin video had more challenges like lens distortion, frequent obstructions, and blue tinting. Therefore, algorithms were easy to develop in Oregon's more controlled environment and could be made robust in Wisconsin's chaotic one. The highest quality (1920 × 1080p) format with no audio available was selected, which, for both videos, was format code 248. After downloading them, I converted the videos from webm to mp4 using another open-source tool, FFMPEG.

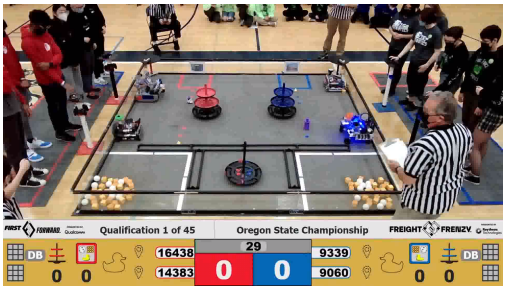



Figure 7: Oregon FTC broadcast

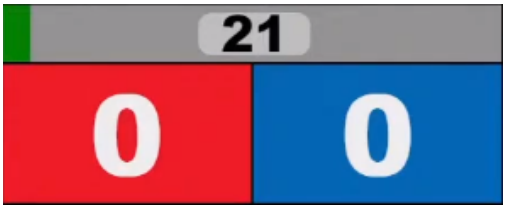



Figure 8: Oregon overlay progress bar

Splitting the many hour-long videos into clips of individual matches was necessary to work with the videos. At 45+ matches per video, splitting all the videos manually would take at least an hour or two. Thus, the only logical solution was to spend five hours automating the task instead. This processing was done using OpenCV and the Tesseract OCR Engine using the Python wrapper PyTesseract. At all FTC events, each match has an overlay at the bottom of the screen with a timer and progress bar. The color of the very start of the progress bar was monitored for a change, which is a good heuristic for detecting the start of a match. To cut down on false positives, I used Tesseract to read the text of the timer. If the text read "29," the frame was recorded as the start of a match. There was no need to record the end of matches as each match is known to last exactly 2 minutes and 38 seconds (30 second Autonomous period, 8 second waiting period, 2-minute Tele-Operated period).

I then used FFMPEG's Python wrapper to extract clips from the stream based on the recorded frames. The overlay at the bottom is cropped out in the main program.

## 4.2 Field Detection

Field detection is a relatively simple two-step process.[‡] It runs only once at the start of the match. The first step is based on line detection and aims to generate a rough perspective transform to get an overhead image of the field. The second step is color based and uses the grey color of the field to further refine the selection.

### 4.2.1 Line Detection and Perspective Transform

Before line detection, some preprocessing is done on the frame. One frame is processed at a time, though multiple frames may pass before detection succeeds as the field cannot be identified under circumstances of extreme obstruction. First, a $5 \times 5$ Gaussian blur with $\sigma_x = 1$ is applied. The edges of the field wall are black, so `cv2.inRange` is used to threshold for dark values. The lower bound is set to `(0, 0, 0)` and the upper bound to `(80, 80, 80)`. Of course, this range encompasses much outside just the field walls, so line detection is necessary. To make the detector's job easier, some morphology is applied. First, an open operation with a $3 \times 3$ rectangular kernel is applied to get rid of noise. Then, two separate dilations are

[†]The code for this section can be found in `notebooks/data_curation_final.py`
[‡]The code for this section can be found in `ftcscore/detection/field.py`

applied. The first uses an $8 \times 2$ kernel designed to bolden the horizontal top and bottom field wall edges. The second uses a $2 \times 4$ kernel to bolden the diagonal right and left field wall edges. It is smaller in its primary dimension (4 pixels, rather than 8), because the right and left edges are never fully vertical, but the bottom and top edges are often very close to fully horizontal. At the end of preprocessing, the image looks like this:

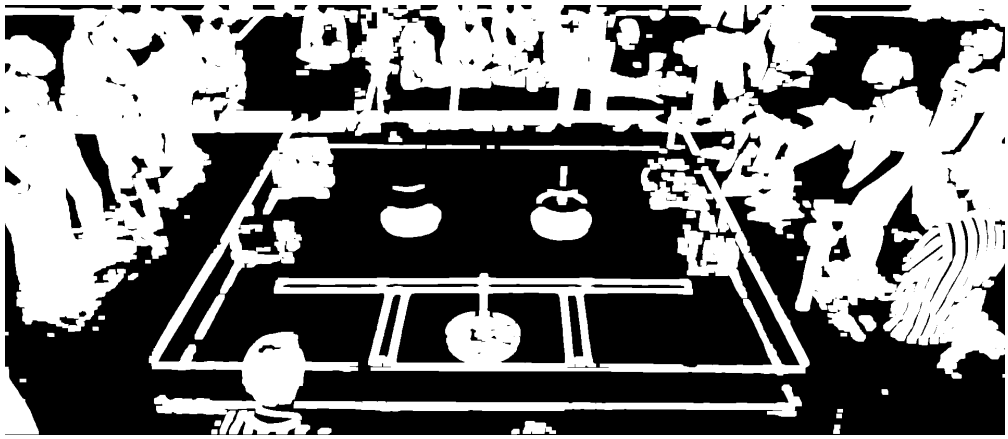


Figure 9: Preprocessed frame from the Oregon broadcast

Now line detection is possible. I tried both line detectors available in OpenCV, `HoughLines` and `LineSegmentDetector`. `LineSegmentDetector` provided better results without tuning, so I used it as a basis for further development. Through experimentation, I found a scale of one tenth (or 0.1) to be the optimal scale.

Optionally, one can apply a line segment merging algorithm such as the one proposed in Hamid and Khan, 2016 here. As the program later applies length filtering to the set of detected lines, merging line segments can help include lines cut off by obstructions or split due to lens distortion that would have otherwise been eliminated by the length filter, leading to more accurate detection. I implemented and applied LSM but found that it did not provide a significant performance benefit. The failure rate of the field detector was already quite low before implementing LSM, and LSM did not seem to cure any of the failed cases. LSM had more benefits when using line detection at a higher scale (more lines were merged), however it still did not provide a noticeable performance increase in actual recognition. Performance is further discussed in the analysis section.

After line detection, four suitable lines must be picked to generate the perspective transform. As mentioned above, length filtering is applied. Any line with a length $\leq 300$ pixels is thrown out. This helps avoid short, incorrectly detected lines. Similarly, slope filtering is applied to avoid lines detected from referee shirts or peoples' legs. Any line where $|slope| \geq 7$ is also thrown out. Finally, the four lines are chosen. Where $(x_1, y_1)$ and $(x_2, y_2)$ are the endpoints of a given line in the set of filtered lines, $y_1 + y_2$ and $x_1 + x_2$ are twice the midpoint's $y$ coordinate and twice the midpoint's $x$ coordinate respectively, and the lines are chosen as such:

- Top line: minimum of lines by $y_1 + y_2$ (highest midpoint)
- Bottom line: maximum of lines by $y_1 + y_2$ (lowest midpoint)
- Left line: minimum of lines by $x_1 + x_2$ (leftmost midpoint)
- Right line: maximum of lines by $x_1 + x_2$ (rightmost midpoint)

If any two lines are the same line, which can occur when less than four lines pass the filtering stage, a failure result is returned, and the process starts again in the next frame.

The intersections of these lines roughly represent the corners of the field. The intersections are calculated using NumPy and the code provided by Tsering, 2017 in their StackOverflow answer. If any of the intersection points are out of bounds for the image size, a failure result is returned and the process starts again next frame. If not, the algorithm has succeeded and the four points can be returned.

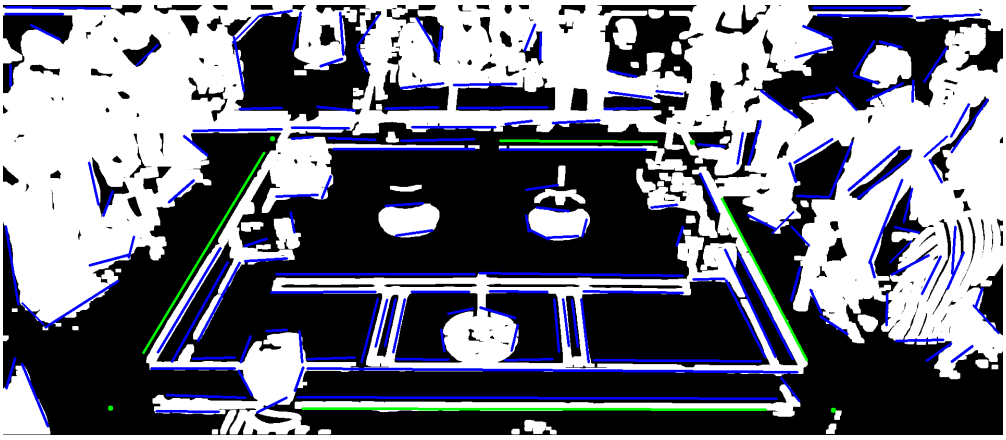


Figure 10: Detected line segments and intersection points from a frame from the Oregon broadcast. Blue segments represent detected lines, green segments represent those selected, and green points represent the intersections of the green segments

A perspective transformation can then be computed using those points. Specifically, a modified version of the code provided by Rosebrock, n.d. is used. The modification is in the output image size calculation. Instead of using the maximum horizontal distance between the points as the width and the maximum vertical distance between the points as the height, which can produce a rectangle, I use the minimum distance between all four points for the width and height since I know the field is a square and that the output should therefore be a square. The perspective transform is now complete and the refinement phase can begin.

Figure 11: A frame from the Oregon broadcast after the application of the computed perspective transform

#### 4.2.2 Color Based Refinement

As seen in the above image, the output from the perspective transform is not always perfect. If one wants distances in the image to be accurately mappable to real life distances, only the grey mat of the field should be visible. To accomplish this, the image is refined by finding grey parts of the image and cropping the image to fit those parts.

The first step in doing so is normalization. Greys can be especially hard to detect if some videos in the dataset are tinted, as was the case with the Wisconsin videos. To overcome this challenge, comprehensive normalization (Finlayson et al., 1998) is applied. Comprehensive normalization was chosen out of a myriad of normalization algorithms because it dealt with greys very well, unifying the field to a very narrow range of colors while not completely crushing it to black along with most of the image like Local Contrast Normalization.

After normalization, simple thresholding is used with a lower bound of `(78, 85, 70)` and an upper bound of `(95, 95, 88)`. An open operation is applied with a $30 \times 30$ rectangular kernel to remove noise and make the mask more cohesive. Next, contours are found and contours with an area $\leq 4000$ are removed. I determined this value experimentally. Although there are cases where due to obstructions, the threshold underselects and misses some areas that should have been included, it generally works quite well. Anything lower tends to overselect, including contours that belong to other grey objects in the scene. Finally, the selected contours are concatenated together into one big contour, and a bounding box is drawn around them.

Figure 12: Contours selected during the color refinement process in a frame from the Oregon broadcast. Selected contours are drawn in green

This bounding box is recorded as the part to crop out, and field detection is complete. Actually, another perspective transform based on the corners of the bounding box would likely be more accurate for robot tracking purposes, but since I am not concerned with that simply cropping the image is enough. The cropped version of the image is used for all further processing.

### 4.3 Hub Detection

I propose two different algorithms to detect the two different types of hubs.[§] For the alliance specific hubs, I use color as the primary feature for detection, while for the shared hub, I use its circular nature.

[§]The code for this section can be found in `ftcscore/detection/game_specific/storage.py`

### 4.3.1 Alliance Shipping Hub Detection

The general strategy for detecting the alliance shipping hubs was to use color information and thresholding to detect the top level and the bottom level of the hub, which can be used to draw a bounding box. As a first step, Local Contrast Normalization (Rad et al., 2020) is applied to the image. I chose LCN because it highlighted the bright blues and reds of the alliance specific areas (See figure 5), making them easier to threshold. Using the normalized image, blue/red areas are selected based on which alliance hub is being detected. Then, a morphological open operation with a $16 \times 16$ elliptical kernel is applied to reduce noise and eliminate areas like the tape marking off the alliance storage unit (another possible destination for freight).

Contours are detected and filtered; only those with $area > 1000$ and $1.4 > aspect\ ratio > 0.6$ are kept. The area filter eliminates small contours that are likely misdetections, and the aspect ratio filter eliminates contours like those from the storage unit tape. Finally, the contour with the smallest $y$ value (closest to the top of the image) is chosen. Now that the top level of the hub is found, the bottom level can be detected as well.

A similar process is followed for detecting the bottom level. First, the image is thresholded for dark areas. Then, morphology is applied to reduce noise as well as close gaps and produce less hollow shapes; an open operation with a $4 \times 4$ elliptical kernel is used for the former and a close operation with $2 \times 2$ elliptical kernel is used for the latter. Contours are found, which are then filtered for those near the top level in the $x$ dimension. Out of the close ones, the one with the largest area is selected. Now a bounding box can be drawn around both contours, and detection is complete. However, as freight lands on the hub and the hub moves around, detection becomes a challenge. Therefore, I advise employing one of the traditional trackers available in OpenCV like CSRT or MIL. I found both to work quite well even under obstructions, though CSRT seems to be slightly faster.

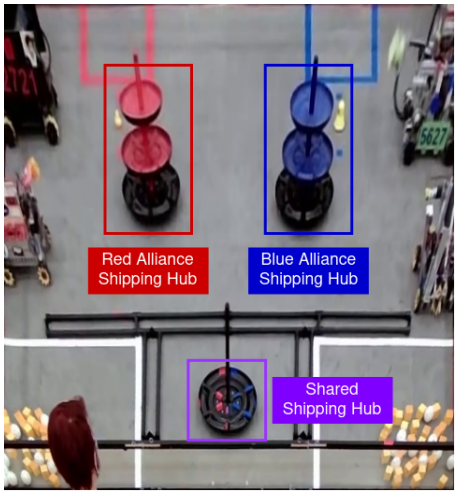



Figure 13: Labeled picture of the different hubs (RevAmped Robotics, 2021)

### 4.3.2 Shared Shipping Hub Detection

Because the shared shipping hub does not occlude itself, unlike the alliance shipping hubs, its geometry can be used to detect it. The image is first converted to greyscale as required by `HoughCircles`. An $11 \times 11$ Gaussian blur with $\sigma_x = 1$ is applied to improve the performance of `HoughCircles`. The parameters of `HoughCircles` were determined experimentally. If no circles are found, detection is attempted again next frame. The circles found are filtered for those in the bottom half and middle third of the image to avoid extraneous detections. If no circles are in that area, detection is retried next frame. Otherwise, the bounding box for the circle is computed. The box is later enlarged by 5 pixels in all directions because `HoughCircles` tended to underselect. Once again, using a tracker for future frames is advisable.

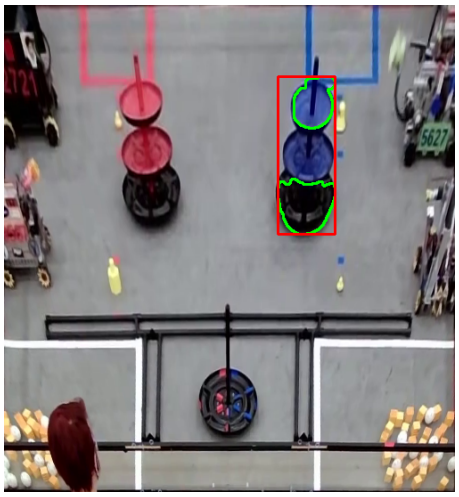


Figure 14: The blue alliance hub being detected (RevAmped Robotics, 2021)

Although I chose `HoughCircles`-based detection in the end, I also tried a color-based detection method. However, due to proximity to the field wall, the detected contour usually included the black edges of the wall, which sometimes led to trackers getting confused and expanding the track window to always include

that edge, even when the hub moved far away from it.

### 4.3.3 Tracking

Because the hubs are technically mobile (though moving them during the game may incur penalties) they must be tracked. Tracking by detection is not reliable because as freight lands on the hubs, the detectable area reduces. Classical OpenCV trackers like CSRT can be employed in tracking hubs with great success, but some modifications can be made to make them faster and more resilient. To make the trackers faster, I suggest simply running them less often. The hubs do not move very rapidly, so waiting 3 or 5 frames between tracking updates does not decrease performance. Additionally, in case the track is lost, the existing detectors can be used to try and regain the track. The algorithm looks like this:

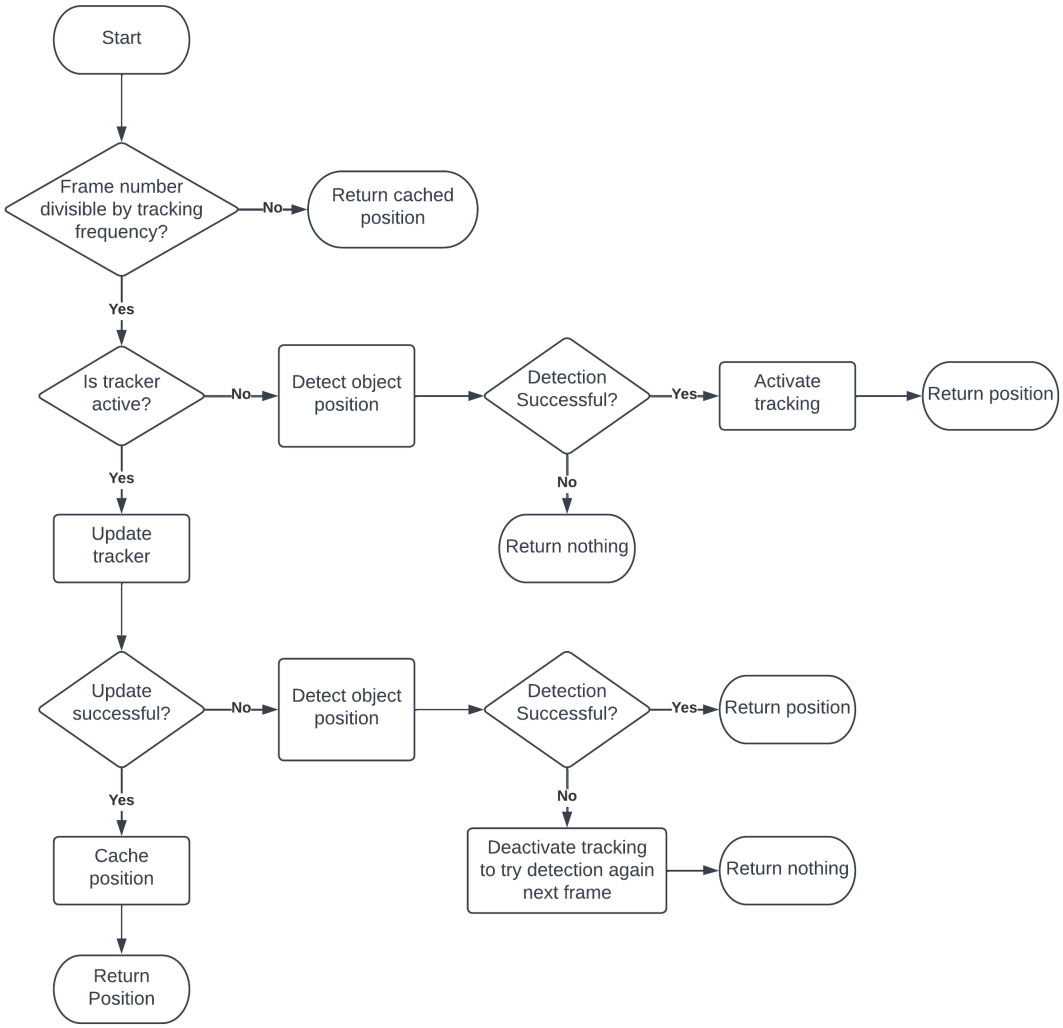
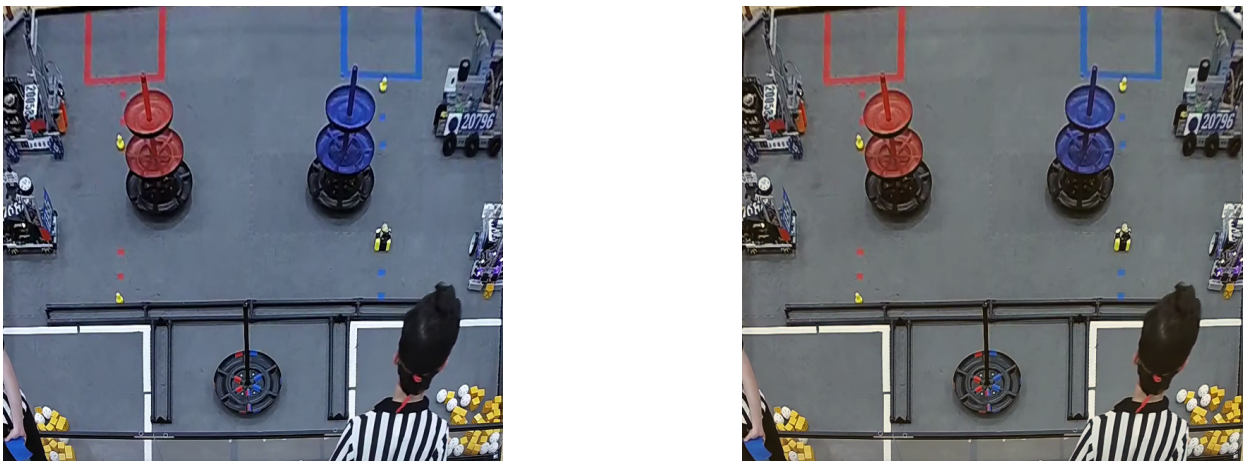



Figure 15: Flowchart of tracking algorithm.

## 4.4 Element Detection

Element detection consists of finding individual and clustered elements and then trying to split the clusters apart.[¶] For the sake of simplicity and due to time constraints, I focus on detecting cubes only. The first step is to apply normalization by standardization per channel as described by Rad et al., 2020. Other methods like LCN or comprehensive normalization tended to modify the image to a point where detection of cubes became near-impossible.



(a) Before normalization

(b) After normalization

Figure 16: Frame from a Wisconsin video before and after normalization by standardization

Figure 17: Splitting a cluster by defects. Yellow points represent detected defects. Blue points represent chosen defects. The blue line represents the split line (RevAmped Robotics, 2021)

Applying no normalization, however, left color tints and other undesirable effects in, also making detection difficult. After normalization, the image is converted to HSV and thresholding is used to select regions where cubes are. Two different masks are used, one for low values like 0-30 and another for high values like 170-180, which correspond to similar colors because the yellow of the cubes is close to the part of the hue spectrum that wraps around. Unfortunately, other yellow objects such as GOBILDA wheels and motors used on some robots exist throughout the field. Luckily, limiting the search to just hubs (specifically the shared shipping hub, where the lighting is more neutral compared to the alliance shipping hubs) avoids most of the incorrect detection. At this point, the program can find contours. Approximation is disabled for `cv2.findContours` to receive more accurate results from `cv2.convexityDefects`, which will be used later. With simple approximation, `convexityDefects` missed some important defects.

The contours are split into two groups based on area: clusters, where $area \geq 130$, and individual cubes, where $130 > area > 40$. The cluster contours are then split into multiple parts based on their convexity defects with an algorithm similar to that proposed in Jing et al., 2015. After convexity defects for each cluster contour are found and a number of failure cases are handled, the two best defects to split the contour are found. The best defects are considered those closest to each other and with a

[¶]The code for this section can be found it `ftcscore/detection/game_specific/`

distance from the convex hull $> 150$. If no defects fitting this criterion are found, the splitting process ends. Otherwise, the contours are split at those two defect points, and the two new contours are split recursively. The contours resulting from each split are added to the list of individual cubes, which is then filtered again by area ($area > 40$) to avoid incorrect splits. Bounding boxes are generated and returned for every contour.

Using the described techniques, the program can successfully detect the field under difficult conditions and detect and track the shipping hubs. Though element detection is flawed, the program can still mask elements accurately and recognize clusters and individual elements. The results are further discussed in the next section.

# 5 Analysis

The algorithms described in the previous section are analyzed to expose strengths and weaknesses. While some parts of the program perform admirably, other parts do not fare so well.

## 5.1 Field Detection

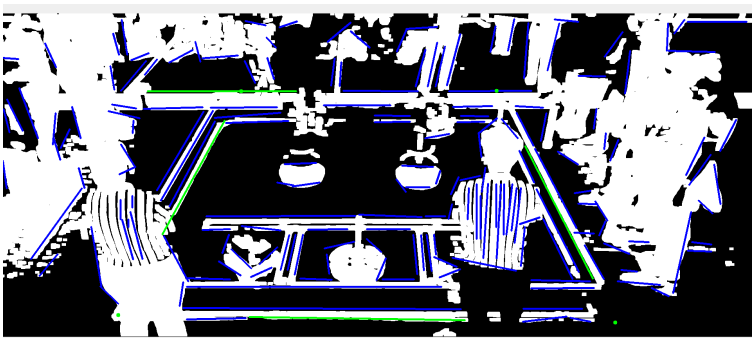
All 55 detected matches were evaluated. Of the 55, 36 were from Oregon and the other 19 were from Wisconsin, though it should be noted that processing of the Wisconsin livestream was cut short and more matches may be available. The field detection algorithm succeeded on 49 out of the 55 videos. Success was defined as correctly generating the perspective transform and bounding box for the field such that the grey part of the field takes up almost all the image. If the detection did not immediately succeed, but did after some obstructions moved, it was still considered a success. The field was eventually detected in all matches. Most detections happened instantly and almost all happened before the end of the autonomous period.

Of the six videos that failed, four were from Oregon and two were from Wisconsin. I investigated all failures. Two of the Oregon videos failed due to over-selection in the color refinement process. Some of the images on the field walls were detected as grey and the contours were determined to be big enough, so they were included in the final bounding box.

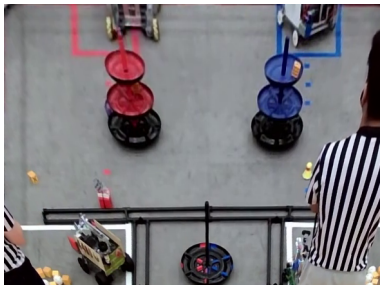
Figure 18: The top wall of the field being included due to failed refinement (RevAmped Robotics, 2021)

Another Oregon video failed due to obstruction of field walls by referees and incorrect thresholding. The upper bound for selecting black was too low and excluded some parts of the edges of the field wall, leading to the line for that edge being split and not passing filtering. Ultimately, this led to an incorrect perspective calculation. The color refinement process then also failed by not including the bottom wall in the bounding

rectangle. Though the final image was usable, it would not be accurate for position tracking on the 2D plane of the field. Perhaps this is one area where line merging algorithms could be helpful. As seen in the figure below, due to incorrect thresholding, the top edge of the top field wall no longer looks like an edge, and the bottom edge has a missing section. At least for the bottom edge, line merging could mend the broken line. One can also see, however, that a judge was obstructing the right wall, so even if the top wall was detected the field detection might have still failed.



(a) The middle of the top wall is not included (RevAmped Robotics, 2021)



(b) In addition to the bad perspective calculation, the bottom wall is cut off by the color refinement process (RevAmped Robotics, 2021)

Figure 19: Failure of the field detector

Both Wisconsin videos failed for similar reasons. In both cases, either robots that did not move during the autonomous portion or referees occluded one of the side field walls, and the line on a referee's shirt was chosen instead of the field wall because the slope filter did not catch that line.

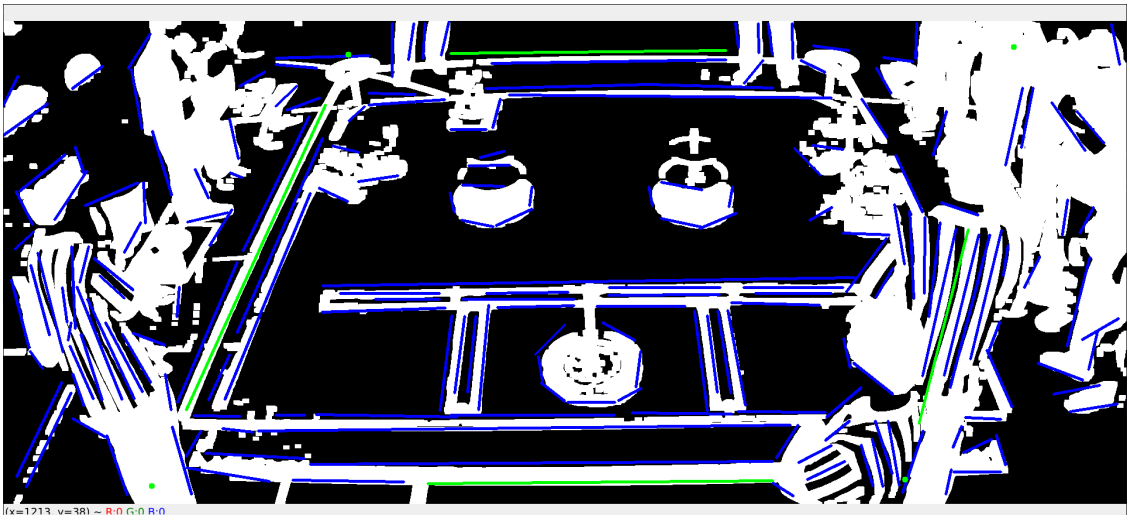


Figure 20: Field detection failure due to occluding robots (RevAmped Robotics, 2021)

Overall, however, the field detector is remarkably resilient to occlusion, lighting changes, and other artifacts. In its current state, it is more than usable, although possible improvements are discussed in the future work section.

## 5.2 Hub Detection

Once again, all 55 matches were evaluated for hub detection accuracy. Accuracy was evaluated based on the first frame after field detection completion. The algorithm *technically* failed on twelve out of fifty-five videos. However, many failures were due to extraneous circumstances. In three videos, detection failed due to failure of the field detector. In one other, a referee was almost completely obstructing the blue alliance hub, so detection was very difficult. In yet another, the match had already started by the time the field detector was able to get an obstruction free image, and a hub was almost completely occluded by robots. Removing these situations, which are arguably either due shortcomings of the perspective transform or are simply impossible situations, one can conclude that only seven out of fifty videos truly failed. Within those that "truly" failed, there were three categories of failure: under-selection, over-selection, and misidentification/mis-selection. Under-selection is when the bounding box does not cover the entire object. Over-selection is when the bounding box covers too much that is not the object. Misidentification is when the bounding box covers an object different than the intended one. I arbitrarily chose videos from each category of failure to investigate further.

### 5.2.1 Under-selection

Under-selection was defined to be any time the bounding box resulting from detection could have reasonably led to a game element not being scored. Under-selection was most common with the shared hub, as HoughCircles would sometimes exclude reflective areas, such as a commonly occurring one towards the bottom of the shared hub, from the detected circle. Under-selection with the blue hub did occur once because the hub was already tipped by the time the detection ran and lighting conditions on the bottom part of the hub became less favorable, leaving some light spots out of the bottom contour.

Figure 21: Shared shipping hub under-selection (RevAmped Robotics, 2021)

### 5.2.2 Over-selection

Over-selection had a less precise definition than under-selection. Over-selection will naturally occur as the shape of none of the hubs can be fully represented by a rectangular bounding box. Therefore, over-selection was defined as any time the bounding box stretched excessively and unnecessarily. Over-selection occurred mostly with the alliance hubs due to robot proximity (not occlusion). For example, one Oregon match failed because a blue robot was behind it at the time of detection, and a Wisconsin match failed because a blue robot was depositing some freight. These sorts of situations are hard to avoid with a color-based detector.

Figure 22: Blue alliance hub over-selection (RevAmped Robotics, 2021)

### 5.2.3 Misidentification

Misidentification/mis-selection was defined as when the bounding box was either significantly skewed away from the ground truth location or just somewhere completely random. It occurred only once and the situation was outside of the control of the hub detector, as a robot was almost completely occluding the blue alliance hub at the time of detection.

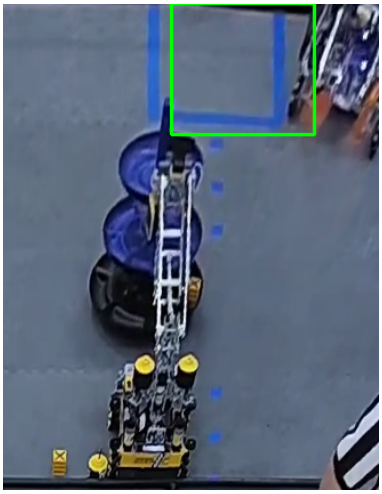


Figure 23: Blue alliance hub misdetection (RevAmped Robotics, 2021)

### 5.3 Hub Tracking

As tracking is handled by a standard algorithm, I will not evaluate it in-depth in this paper. However, it should be known that while CSRT (the chosen algorithm for this case) generally seems to perform well under partial occlusions, failures under full occlusions, such as when referees walk across the field, do arise.

### 5.4 Element Detection

With individual cubes, the element detector has no problem. Bounding boxes are generally correct and one per cube, though there are rare instances where it thinks one cube is actually many and tries to split it apart.

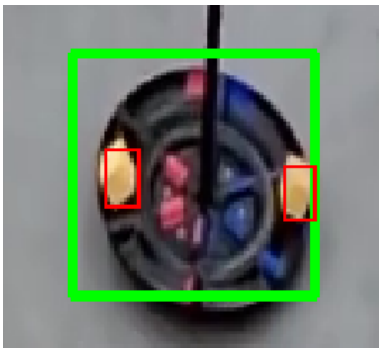


Figure 24: Successful detection of cubes (RevAmped Robotics, 2021)

Two to three connected cubes are also generally separated correctly provided the creases in the silhouette are more or less clear. Sometimes the generated bounding boxes can be wildly incorrect, but the *number* of bounding boxes is usually correct.

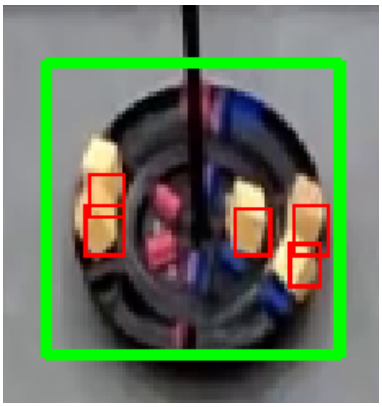


Figure 25: Successful detection of multiple conjoined cubes (RevAmped Robotics, 2021)

Where the algorithm suffers is with clusters composed of more than 3 cubes or smaller clusters where the separation between cubes is not clear at all. It separates the clusters sporadically and is also temporally inconsistent (from frame to frame, detections vary a lot).

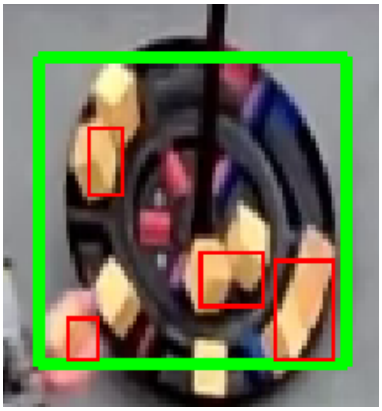


Figure 26: Failed detection of large clusters and clusters with unclear separation (RevAmped Robotics, 2021)

Finally, the thresholding is not perfect, either. It can sometimes encompass light skin tones, leading to incorrect detections under obstruction.

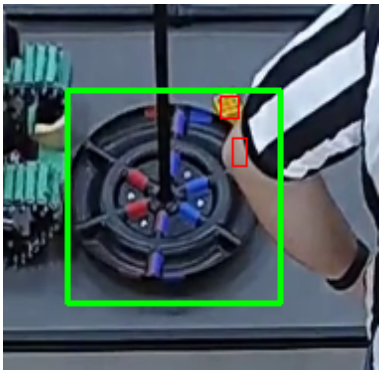


Figure 27: Incorrect detection of a referee's arm (RevAmped Robotics, 2021)

# 6 Future Work

As demonstrated, although many milestones to achieving automated scoring of FTC games are easily within reach, there are also many challenges awaiting future attempts. From frequent obstructions to having to identify small objects, there are several factors that make this task harder than it seems on the surface. In this section, I hope to suggest a few improvements and possible solutions for future attempts in the space.

## 6.1 Field Detection

### 6.1.1 Lens Distortion Correction

Though the field detector is extremely resilient, possible improvements do exist. One issue the field detector faced was dealing with the curvature due to lens distortion in the Wisconsin livestream. Even at a low (1/10th) resolution, the Line Segment Detector rightfully split up some curved lines. As described in the previous section, attempting to merge these lines often does not yield satisfactory results. Thus, the accuracy of the field detector would likely increase if lens distortion could be eliminated.

Of course, the challenge here is that traditional distortion correction methods such as Zhang's chessboard method (Zhang, 2000) are not applicable due to the lack of, well, a chessboard. One potential remedy is to glean the information from other known objects and their properties, such as the walls of the field or lines on the referees' shirts, both of which are known to be straight lines.

### 6.1.2 3-Line Perspective Estimation

A relatively easy improvement is to allow the use of less than four lines for estimating the perspective of the field. One common problem the field detector faced was dealing with obstructions. This can be overcome by estimating what the last line *would* have looked like from the position and slope of the other three lines. If the left line is occluded, for example, it could be assumed to have the negative slope of the right line. One could then assume its midpoint to be at the same $y$ as the right line. The $x$ could be reflected over the vertical bisector of the top or bottom line. This way the line could be fully reconstructed. This estimation would allow the field detector to be even more resilient to obstructions, though it can be argued that waiting for a better view with fewer obstructions is the better strategy. One can also argue that this feature is rather moot, because if the software were to be used in a real setting, the view could easily be cleared of obstructions for the one-time set up phase.

### 6.1.3 Viewing Side Resiliency

To simplify this work, livestreams with cameras viewing fields from the same side were selected. Ideally, the system should be able to adapt to at least some different viewing angles. For example, games can also be

viewed from the side closer to the alliance hubs. In this case, the field would have to be rotated or mirrored for consistency. The position of the alliance hubs in the overhead view can be used to recognize this situation.

## 6.2 Game Element Detection

The cube detector presented in this paper, though functional and accurate with individual cubes, leaves a lot to be desired with clumps of cubes. The separator can separate clumps of two or three cubes but tends to fall apart with larger clumps. The fact that this task can be hard, even for humans, suggests that machine learning may be the way forward. The ability for our eyes to easily harness information like orientation, clump size, and where shadows are makes the task doable for humans. Machine learning (ML) algorithms like Convolutional Neural Networks have the potential to build that understanding without explicit instructions on how to do so.

The challenge with such an algorithm is the collection of data. To my knowledge, no kind of labeled dataset exists for FIRST games. Either such a dataset would have to be created, or a generic object detection algorithm would have to be applied. The good news is that Tensorflow based object detection already exists (FTC SDK Maintainers, 2021) and is widely used on FIRST robots, which indicates that it may be possible to build a similar system for a camera with an outside perspective.

## 6.3 Hub Detection

Generally, it may be possible to improve performance by tuning parameters like color thresholds and Canny parameters for `HoughLines`. Swapping out the geometry based shared hub detector for something color-based may also be helpful. As relying on the black parts of the shared hub produced problems as described in section 4.3.2, perhaps relying on the small color markers on the shared hub will be more fruitful. However, the biggest performance gain will likely come by allowing hub detection to run sooner, i.e. improving the field detector, as many of the problems faced were direct results of the field detector taking too long to detect the field and robots interfering because of it.

## 6.4 Robot Detection and Tracking

The reader may have been surprised to find no mention of tracking robots in this paper. The reason is that for most tasks in FTC, where the robots are simply does not matter; all that matters is the position of game elements. There are *some* tasks, however, that do rely on the position of the robot. To score such tasks, detecting and tracking robots is a necessity.

One approach to detection is to use background subtraction at the beginning of a match. There are two issues that would have to be overcome, however. First, robots would have to be differentiated from moving obstructions like referees. This could potentially be done by limiting the scope of the background tracker to areas where robots are expected. The second issue is that not all robots move at the start of the match during the autonomous phase. Continuing to run the background on those regions could be a solution.

As far as tracking goes, there are many solutions. A kernel tracker, as described in Yilmaz et al., 2006, is probably the most suitable. The chosen tracker will have to handle translational and rotational motion without losing the track, but multiple object tracking or full occlusion handling is not required, as robots do not tend to occlude each other and when it comes to occlusions due to the environment, partial, short-lived occlusions appear to be the most common. Based on Table IV in Yilmaz et al., 2006, the Appearance Tracking and Layering methods both look like good choices for future attempts.

**Table IV.** Qualitative Comparison of Geometric Model-Based Trackers (Init. denotes initialization. **#**: number of objects, **M**: multiple objects, **S**: single object respectively, **A**: affine or homography, **T**: translational motion, **S**: scaling, **R**: rotation, **P**: partial occlusion, **F**: full occlusion. Symbols $\surd$ and $\times$ respectively denote if the tracker requires or does not require training or initialization.)

| | # | Motion | Training | Occ. | Init. |
|---|---|---|---|---|---|
| Simple template matching | S | T | $\times$ | P | $\surd$ |
| Mean-shift [Comaniciu et al. 2003] | S | T + S | $\times$ | P | $\surd$ |
| KLT [Shi and Tomasi 1994] | S | A | $\times$ | P | $\surd$ |
| Appearance Tracking [Jepson et al. 2003] | S | T + S + R | $\times$ | P | $\surd$ |
| Layering [Tao et al. 2002] | M | T + S + R | $\times$ | F | $\times$ |
| Bramble [Isard and MacCormick 2001] | M | T + S + R | $\surd$ | F | $\times$ |
| EigenTracker [Black and Jepson 1998] | S | A | $\surd$ | P | $\surd$ |
| SVM [Avidan 2001] | S | T | $\surd$ | P | $\surd$ |

Figure 28: Table IV from Yilmaz et al., 2006

# 7 Conclusion

FTC is a very popular global robotics competition with over 7,000 teams worldwide. Teams in FTC compete at events to move up to higher level events, eventually getting to the "worlds" level if they are good enough. Generally, each match in FTC consists of four robots and two "alliances," which are determined randomly per match.

Judges and scorekeepers keep score and hand out penalties according to the rules for each match. At the end of a match, the teams on whichever alliance has the most points are considered the winners for ranking purposes. Scoring, however, can get pretty hectic, requiring three or four people to keep score per match. This can be seriously taxing and produces a real barrier to entry for running an event, as at least three volunteers are necessary just to do scoring. If the necessary number of volunteers can be reduced, FTC can be made more accessible.

In this paper, I focused on laying the foundations for developing scoring assistance tools for FTC. I specifically work on archived livestreams from last year's competition, Freight Frenzy. The primary goal in Freight Frenzy is to move freight, which comes in the form of yellow cubes, white spheres, and rubber ducks, from the warehouse to a number of shipping hubs. I tackled field detection, shipping hub detection, and freight detection limited to yellow cubes for simplicity.

Overall, my conclusion is that scoring FTC is hard. Conditions at events are often less than ideal for computer vision, with lighting varying a lot, artifacts like lens distortion and color tints being common, and obstructions occurring frequently. Even so, some surprisingly resilient algorithms can be created; the field and shipping hub detectors work quite well even under these conditions. The two stage pipeline for the field detector makes it especially powerful. The freight detector needs significant work, but is somewhat functional. Potential avenues for improvement include the application of CNNs. There is work to be done, but the foundations have been laid, and the gates to future research in scoring FTC or other similar competitions are open.

# 8 References


FFMPEG Contributors. (n.d.). *Ffmpeg*. https://ffmpeg.org/

Finlayson, G. D., Schiele, B., & Crowley, J. L. (1998). Comprehensive colour image normalization. In H. Burkhardt & B. Neumann (Eds.), *Computer vision — eccv'98* (pp. 475–490). Springer Berlin Heidelberg.

FTC Organizers. (2021). *Freight frenzy game manual part 2*. https://firstinspiresst01.blob.core.windows.net/first-forward-ftc/game-manual-part-2-traditional.pdf

FTC SDK Maintainers. (2021). *Using tensorflow in freight frenzy*. https://github.com/FIRST-Tech-Challenge/FtcRobotController/wiki/Using-TensorFlow-in-Freight-Frenzy

FTC Wisconsin. (2021). *Ftc wisconsin live stream*. https://www.youtube.com/watch?v=6UbA9wDfa5k

Google and Tesseract Contributors. (n.d.). *Tesseract*. https://github.com/tesseract-ocr/tesseract

Hamid, N., & Khan, N. (2016). LSM: perceptually accurate line segment merging. *Journal of Electronic Imaging*, *25*(6), 061620. https://doi.org/10.1117/1.JEI.25.6.061620

Jing, C., Potgieter, J., & Noble, F. (2015). A colour detection and connected-objects separation methodology for vex robotics. *345*, 391–399. https://doi.org/10.1007/978-3-319-16841-8_36

Lukežič, A., Voj'iř, T., Čehovin Zajc, L., Matas, J., & Kristan, M. (2018). Discriminative correlation filter tracker with channel and spatial reliability. *International Journal of Computer Vision*.

OpenCV Contributors. (n.d.). *Opencv*. https://opencv.org/

PyTesseract Contributors. (n.d.). *Pytessearct*. https://github.com/madmaze/pytesseract

Rad, M., Roth, P. M., & Lepetit, V. (2020). ALCN: adaptive local contrast normalization. *CoRR*, *abs/2004.07945*. https://arxiv.org/abs/2004.07945

RevAmped Robotics. (2021). *Ftc 2021-22 - oregon state championship*. https://www.youtube.com/watch?v=JvAtzLdOtbM

Rosebrock, A. (n.d.). *4 point opencv getperspective transform example*. https://pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/

Tsering, N. (2017). *Answer on stackoverflow post about finding line intersections with numpy*. https://stackoverflow.com/a/42727584

Yilmaz, A., Javed, O., & Shah, M. (2006). Object tracking: A survey. *ACM Comput. Surv.*, *38*(4), 13–es. https://doi.org/10.1145/1177352.1177355

youtube-dl Contributors. (n.d.). *Youtube-dl*. https://github.com/ytdl-org/youtube-dl

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, *22*(11), 1330–1334.