1678

CITRUS CIRCUITS

2022

SCOUTING WHITEPAPER

TABLE OF CONTENTS

INTRODUCTION

History

Citrus Circuits' electronic scouting system was first developed and used during the 2013 FRC season of Ultimate Ascent. The team had previously used a paper scouting system but was overwhelmed by the management issues that arose when scouting 100 teams at the World Championship. A software development team of four students created the original system that has since grown into a full subteam of 20 students and multiple apps.

The system had two relatively unique attributes: (1) collection using Android tablets of both objective scoring—each focused on a single robot—and subjective ordinal rankings of robots' driving strengths and abilities within each match alliance; and (2) delivery of collected and processed data to a cell phone app in near real time to be used by the drive team to prepare for matches. The objective and subjective data were quantitatively combined using weighted scoring to provide two ranked priority draft lists for first and second picks for alliance selection prior to the playoffs.

The system proved successful in its initial use at the 2013 Central Valley Regional when 1678 was able to fully assemble an alliance based on collected scouting data to win the event from the 6th seed. The system again proved its worth at the Championship where 1678 won the Curie Division. The entire system ran at events with eight Scouts and two programmers.

The scouting system has evolved in several ways, including a couple of "hiccups." Initially, the system was cobbled together and then switched to Bluetooth in 2014. The first significant revision occurred in 2015 with the first multi-object game presented by FIRST. However, the scouting system failed to provide usable data at the first event of the year. In 2016, the Software Scouting team created a project management schedule that delivered substantial improvements to the system. Another app was added in 2016 to collect specific data from pit visits. In 2017, the crew of Scouts was expanded to assign three Scouts to each robot for collecting objective data to improve accuracy. In 2018, Bluetooth became less reliable with the prevalence of smartphones at events, and QR code communication was added for match data collection.

In 2020, the Match Collection, Pit Collection, and Viewer apps were rewritten in Kotlin for use on Android phones. The system has relied on several online database programs, adopting Firebase and then moving to MongoDB in 2020. At the first competition in 2020, the system did not deliver data, but the season was terminated by the COVID-19 pandemic before the system was updated further. In 2021, the subteam continued to meet remotely and work on making improvements to the structure of the 2020 system without the need to focus on a specific game. The 2022 system is a result of a substantial revision and improvement from the 2020 system.

Summary of Major Changes Since 2021

From 2021, the Viewer application now has a live picklist editor, team in match data, ranking by data point, team details graphs, a field map, a pit map, and robot pictures. Most of these features existed in the Viewer prior to Kotlin redevelopment in 2020, and their basic functionality was taken and improved for the current app. For detailed descriptions of these features, see the Viewer section.

At the end of the 2021 season, the Viewer application was still unable to retrieve updates from the MongoDB server, and also could not update without resetting the entire application. During the offseason, the primary goal was to have complete communication from collection to processing, and finally, to visualization. After having difficulties with MongoDB Realm and KMongo (a Kotlin-MongoDB plugin), the subteam decided to create a web API—nicknamed "Cardinal"—to access data from the cloud database and host it for the Viewer. For more information on Cardinal, see the 2021 Whitepaper.

During the 2022 season, the Server added a calculation for approximately measuring the accuracy of Scouts, known as SPR (Scout Precision Ranking). This calculation had been a part of previous years' Server calculations, but was removed because it was unreliable. This year, a new formula was designed to avoid the flaws of the previous method.

System Overview

The Citrus Circuits scouting system is broken up into three main stages: collection, processing, and visualization. Collection consists of the Match Collection app and the Pit Collection app. In the processing stage, the Server organizes and runs calculations on the data, which the Viewer app and the Picklist Editor can then visualize by pulling the data through Cardinal.

At competitions, Scouts use two collection apps: the Match Collection app and the Pit Collection app. Each app contains two "modes" for users to input either objective or subjective data.

The users of the two collection apps are as follows:

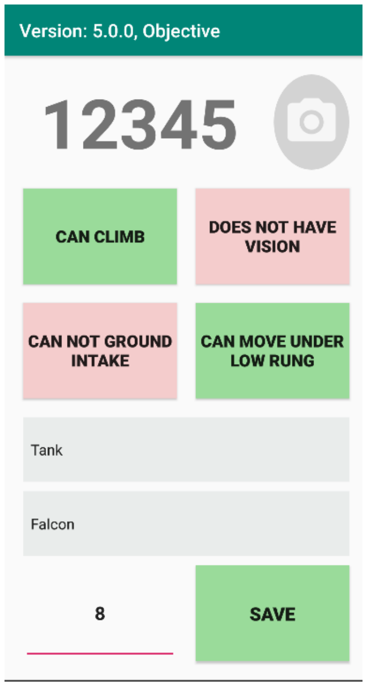
- Match Collection (Objective): Scouts
- Match Collection (Subjective): Super Scouts
- Pit Collection (Objective): Pit Scout (same as Match Strategist)
- Pit Collection (Subjective): Team members focused on 1678's alliance partners
    - This mode was not implemented this year for reasons mentioned in the next section.

PIT COLLECTION

The Pit Collection app is mainly used by the Match Strategist with the assistance of other strategists (depending on the competition). It runs on Android phones and was built using Kotlin and XML, and is used to record mechanical data and to take pictures of robots' mechanisms. In past years, there was also a subjective mode used by a pit group focused on 1678's alliance partners, in which they could record how feasible it would be to install a new mechanism on a team ("cheesecaking"). It was removed this year because strategists decided they did not need subjective pit information.

List of Data Points

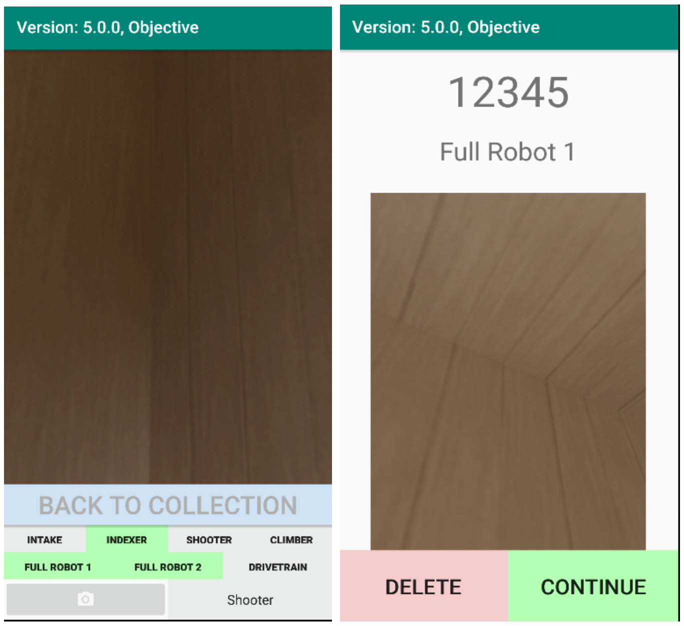
The Pit Collection app is used to collect a number of data points about the physical characteristics of a robot. These include whether or not the robot can climb, if it has ground intake, if it can move under the low bar, if it has a camera for vision on the far side of the field, what kind of drivetrain it has, what kind of drivetrain motors it has, and the number of drivetrain motors.



Pit Collection main team data collecting screen

Pictures

The Pit Scouts also take a picture for each robot's intake, indexer, shooter, climber, drivetrain, and two pictures of the full robot from different angles. These pictures are taken in the app using the phone camera and stored in the local Downloads folder to be pulled by the Server computer via a USB connection.



Taking pictures of the different parts of the robot

Naming Photos and JSON Files

The data that is collected is stored as JSON files in the downloads folder along with the pictures. For objective data, each file is named with its team number followed by `_obj_pit`. Subjective files are named the same way, except they use `_subj_pit` instead of `_obj_pit`. The pictures are named with the team number first and then what part of the robot it is a picture of. For example, `1678_indexer.jpg` or `0000_shooter.jpg`.

Starred Teams to Organize Multiple Scouts

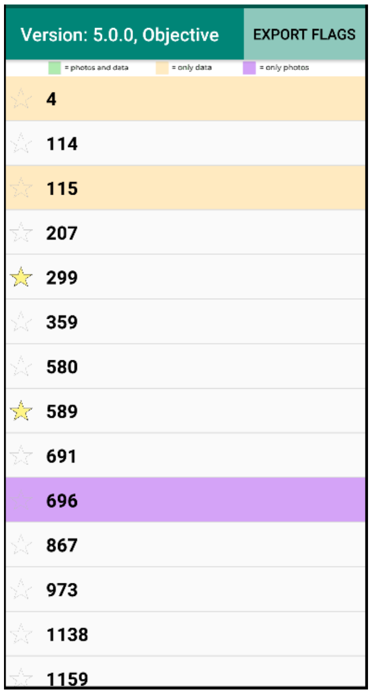
In order to organize scouting teams with multiple Pit Scouts, if a user long clicks on a team's cell in the list of teams, a yellow star will appear on the left side of that cell. This way, Pit Scouts can divide the teams between each other. Rather than starring teams in a range, Pit Scouts star teams one-by-one to allow for nonnumerical pit setup orders.

Flagging Multiple Teams

Flagging allows users to copy and paste a list of teams that meet set conditions. For instance, developers might set a certain drivetrain as a flag, then all the teams with that drivetrain will be saved to the clipboard to be pasted in a Slack channel. Conditions are set while developing the app, not in the app by users.

Highlighting to Show Scouting Progress

When looking at the list of teams, the color of each team's cell depends on what kind of data has already been collected on that team. If there is no data on a team the cell will be white, if there are only pictures the cell will be pink, if there is only data the cell will be yellow, and if there is both pictures and data the cell will be green.

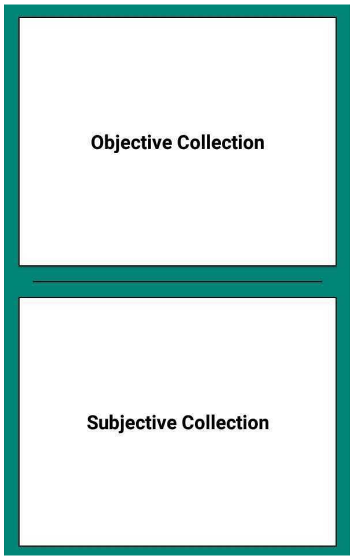


☆ 4
☆ 114
☆ 115
☆ 207
★ 299
☆ 359
☆ 580
★ 589
☆ 691
☆ 696
☆ 867
☆ 973
☆ 1138
☆ 1159

List of teams in Pit Collection with some teams starred or highlighted to show scouting progress

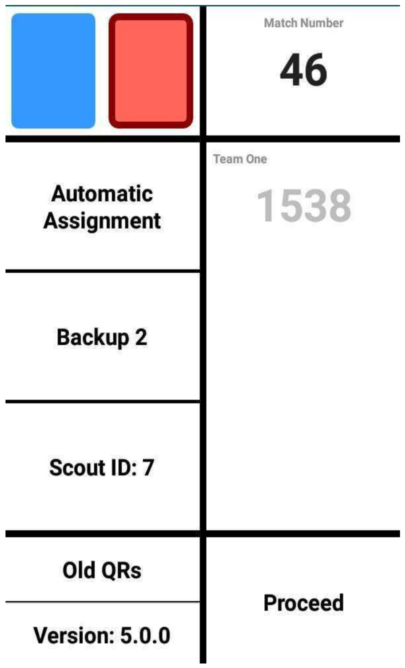
MATCH COLLECTION

The Match Collection app is used by Scouts in the stands during matches to collect robot performance data. It runs on Lenovo Android tablets and was built in Kotlin and XML. There are two modes of the Match Collection app: objective collection and subjective collection. In each match there are 18 Objective Scouts (three per robot) and two Super Scouts (one per alliance). Objective mode collects quantitative data (for example, how many balls a robot scores). Subjective mode is used to rank the performance of each robot on the alliance (for example, the field awareness ability between the robots on the Red alliance).



Objective/Subjective selection screen

Objective Collection



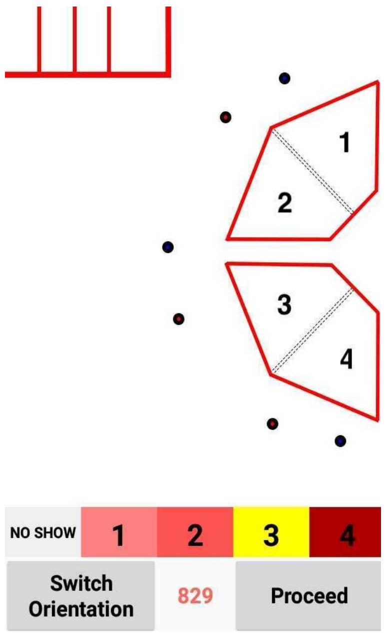
Objective Collection information input screen

Randomizing Scout IDs

In previous years, each Scout ID was always assigned to the same driver station position in each match. This year, to increase scouting accuracy, Scout assignments to robots were randomized. Groups of three students Scout each one of the six robots, then the groups of Scouts who are scouting the same robot are shuffled for the next match. This structure prevents Scouts from always scouting the same team as each other, so Scouts can be better compared to one another and against the data on The Blue Alliance to rank the accuracy of each Scout (see Scout Precision Ranking (SPR) in Server). The assignment order is determined via a file resource with 100+ randomized orders of Scout IDs, which is generated ahead of competition. Randomization is not done in real time.

Starting Positions

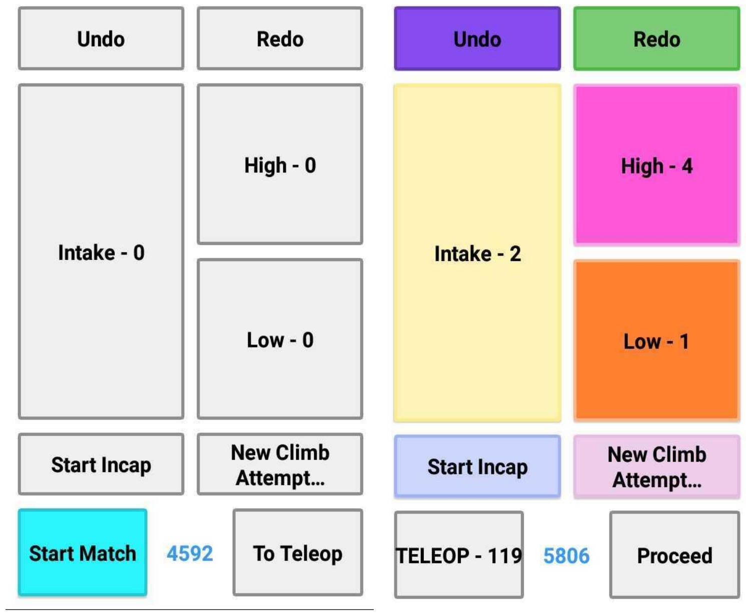
Before the main activity screen, Scouts must input their robot's starting position, which is based on subdivisions of the Tarmac. Starting position and the number of balls a team scored in auto gives strategists a hint at what autos a team may have, without Scouts drawing out a path or another complex solution.



Starting position screen red alliance

Simple UI

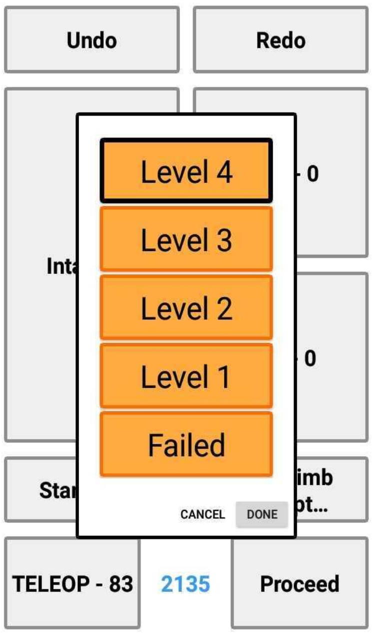
During week one, the different areas teams scored from, common penalties (e.g. scoring the wrong colored ball), and other pieces of extra data were also collected—but the large quantity of options caused inaccurate data and the extra data was unused. Most of these features were removed and the app was limited to nine buttons that tracked scored high, scored low, intake, incap, climbing, and other very simple actions (see the screenshots). At later competitions, after the extra buttons were removed, scouting accuracy increased and Scouts found the decrease in buttons much easier and were able to wait longer between breaks.



Objective Match Collection screen for Scouts before and after starting a match

Climb Popups

During Endgame, Scouts input the climbing level via a popup with four buttons: failed to climb, low climb, mid climb, high climb, and traversal climb. Originally, Scouts also collected climb time, but the metric was not always accurate due to the variety of climbs and Scouts having difficulty knowing when a robot stopped climbing. After the first competition, Scouts collected only the climb level and climb time (which is harder to scout) was noted down by the Stands Strategists (see Competition Roles for more information).



Climb popup for recording climb level during endgame

Incap Time

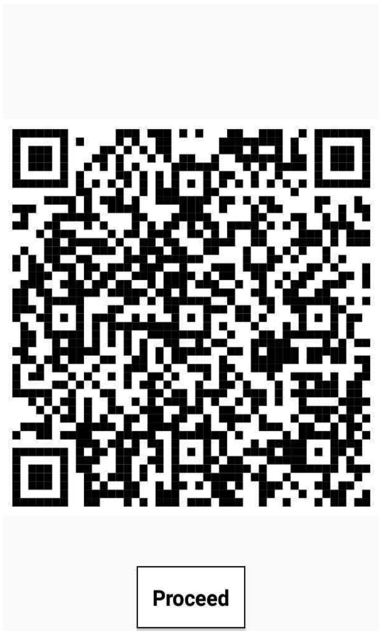
Incapacitation (incap) is defined as a period of time when a robot is dead on the field or having enough mechanical issues that it is unable to accomplish any actions. A toggle button notes down the start/stop timestamps for when a robot goes incap. When the data is processed, Server finds the difference in times between the timestamps to find the total time incap during a match, which is later combined to create the average time incap and the number of matches a team was incap. If a team is incap for less than 8 seconds, the incap time is considered inconsequential in the overall match turnout and Server calculations disregard it.

Intakes

Scouts track intakes by clicking counter buttons that display the current number. Strategists also used intakes to judge whether a defense robot has the ability to intake opponent balls in a timely manner.

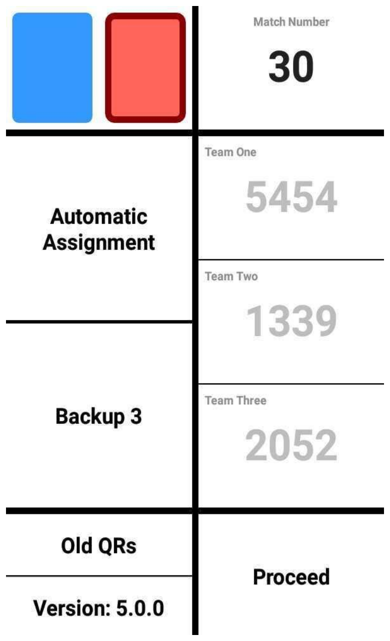
QR Schema/Complexity of the QR

QRs follow a specific formatting defined in Schema to lower the amount of data that needs to be displayed. Data point names are represented by letters of the alphabet and each section and data point is separated by special symbols such as '$'. This way, the QRs are easily generated and are small enough that all match data can be contained in one relatively small QR code.



QR code generator screen

Subjective Collection



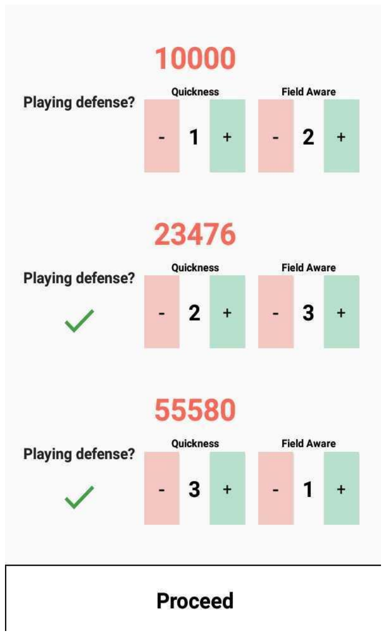
Subjective Collection information input screen

Quickness and Driver Awareness

Super Scouts rank each robot on an alliance—relative to the other robots on the alliance—by quickness and driver awareness. Ranks are from 1 to 3, with 3 indicating the best robot in that category on the alliance. Quickness is based on the maximum speed and maneuverability of the robot, while driver awareness is based on how much the driver is aware of the positions of their robot and other robots on the field.

Defense Checkbox

In addition to ranking data points, Super Scouts also fill out a simple checkbox about whether or not a robot played defense. This provides a simple estimate of a team's experience playing defense. It also helps identify matches to review for defense performance during the picklist meeting.

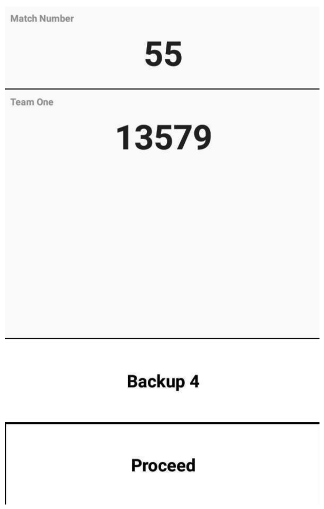


Subjective Collection team data input screen

Other

Retrieving QRs from Previous Matches

In the case that an objective or subjective Scout fails or misses scanning their QR, Match Collection has a method of storing and retrieving QRs. After the app generates each QR, its compressed string is also stored in a file in the Downloads folder of the tablet. If a Scout forgets to scan their QR, they can press the Old QRs button on the Team Assignment Screen to see a list of all the QRs from previous matches scouted on that tablet, and display each one by tapping on its cell. The QR is generated from the string pulled from the file in the tablet's Downloads folder.


Match Number

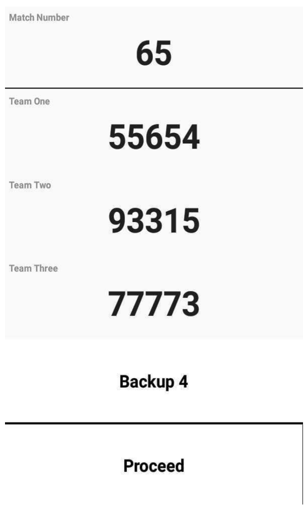
55

Team One

13579

Backup 4

Proceed


Match Number

65

Team One

55654

Team Two

93315

Team Three

77773

Backup 4

Proceed

Objective scouts *Subjective scouts*

Allows scouts to edit information before generating QR code

SERVER

During a competition, the Server works in tandem with the Scouts to provide accurate data to the strategists. As matches continue, the Scouts collect data which is calculated in different forms and then sent to the Viewer app. Raw and calculated data are stored in a local MongoDB database on a designated laptop, and are uploaded to a cloud database after every match. For more information on databases, see Section 4.1.6.1 Local and Cloud Databases in the 2020 Whitepaper.

In the past, there were difficulties with sending data to the Viewer, so a new component was added to the system in the offseason. Now, data is sent through a web server (nicknamed Cardinal) hosted by a mentor. Cardinal reads data from the cloud database and provides an API for the Viewer to read it. For more information, see the Cardinal section of the 2021 Whitepaper.

Schema

Schema provides the database structure for the Scouting System. Schema files contain important information about all data fields calculated by the Server. Having all the data fields in Schema makes it extremely easy for developers to change them. Throughout the season, the data fields that are collected and calculated are constantly changing. The ability to change all the data field information in Schema (such as weighting) ensures that only minimal changes need to be made to the code itself. During competition, strategists may want to change the weighting of certain data fields, and storing data point information in the schema means that the changes are able to be made easily and quickly.

Testing

Most Server code has automated testing written for it using the pytest and unittest libraries. These tests are stored in a test folder alongside the src folder. Each calculation and script file has a corresponding test file. PRs that change code in the src folder also update the respective test files, which are reviewed during code reviews in place of user tests. A GitHub action automatically tests each PR when it is opened and displays whether it succeeds or passes.

Automated testing provides several advantages. Previously, it was difficult to standardize the environment in which code was run, leading to code that would function on one developer's device but fail on another. Tests also serve as documentation for the main code, acting as an example of how it should run. The automatic GitHub action ensures that each PR is functional before it is merged. Running tests with pytest is also convenient, since it allows parts of the Server to run in isolation. To test the entire Server, a field test is normally conducted (see Appendix B).

QR Blocklisting

If for any reason, a QR that has been inputted into the system needs to be removed, it is blocklisted rather than deleting the raw data. A QR may be blocklisted if a Scout missed a large portion of the match, leading to the data being inaccurate, or if a match gets replayed. Blocklisting a QR will not delete it from the database, but will flag it so its data is ignored by calculations. Calculations need to be rerun after blocklisting the QR to get accurate data. For more information, see Section 4.1.7.1 QR Handling in the 2020 Whitepaper.

Collecting Climb Data

In previous years, climb data was gathered from the TBA API. However, TBA reports robots who didn't climb but were awarded a free climb due to a foul as having climbed normally. Because of this, Scouts collected climb data this year in order to ensure optimal accuracy.

Using CSVs or JSONs

Previously, the Server stored information about an event's team list and match schedule in CSV files. This year, due to several reasons, that was changed to store team lists and match schedules in JSON files. Firstly, JSON files are easier to read than CSV files. JSON also has more versatility and is better for handling larger amounts of data.

CSVs are still used in some parts of the system. In order to send data to the Picklist Editor, the Server exports data from the local database as a CSV file. Robot images are also uploaded to Imgur, an online image hosting system, and the image URLs are exported into a CSV to be imported into the Picklist Editor.

Calculations

Data processing is split into multiple calculation files, each for a different category of data. Each calculation file has its respective schema file and collection in the database. All calculations are subclasses of the parent class BaseCalculations, which contains methods used by most or all calculations. Each calculation subclass has a run() method which checks for the newest changes to the database, calls the other methods within the class to create updated data, and then writes the changes to its collection. The server.py imports these calculations in the order listed in calculations.yml and simply calls the run function to execute the calculation. For more information on specific calculations, see Section 4.1.7.2 Consolidation and Objective TIM Calcs in the 2020 Whitepaper.

Subjective Data

In the past, subjective data was sent in QRs and stored in the database as Alliance In Match (AIM) data, with each database document storing a ranked list of three teams. This year, the structure was changed to split each team into its own document in the database, with values between 1 and 3 to represent its subjective rankings. This was done to allow Super Scouts to occasionally assign two teams to the same ranking if it was impossible to distinguish between their driver skills. Data for all three teams in an alliance is still sent together in a single AIM QR, which is split into three subjective TIM documents during decompression. This also allowed Super Scouts to collect information about individual teams which was not a ranking, such as whether or not they played defense.

Picklist Calculations

Pickability is a metric that allows teams to be pre-sorted before strategists discuss and refine the picklist order. Each team has a first and second pickability, which approximates their suitability as a first pick and a second pick respectively. Pickability is calculated using each team's values for certain data points, and depending on the individual weighting, a weighted sum is produced.

In the past, pickability weights were decided based on experience and trial and error. This year, a more mathematical approach was taken to determine which data points to use and how heavily to weigh their values. After an event, all the teams are sorted into their perceived ranks such as high, medium, low, etc. Next, a few different multivariate linear regressions are run to see what model best predicts each team according to the assigned ranks. We then use the given coefficients as our weights for each component data point. After this, we calculate the pickability values for each team and rank them from best to worst for every model estimated. Finally, we have a blind comparison to see what model leads to the rankings that make the most sense and choose to use that pickability model for the following competition. This would be run for both first and second pickability metrics.

Through this process, we found models that were able to predict the relative ability of teams better than our previous models. For example, in our second pickability metric at Houston Champs we were able to find that (intakes)^2 was a great estimator for driver ability and in turn, positional defense ability.

Scout Precision Ranking

In previous years, we attempted to measure the accuracy of Scouts for use in consolidation and robot assignments. Scout Precision Ranking (SPR) was calculated based on the number of decisions a Scout made that were considered to be "correct." A "correct" decision was considered to be the majority decision between the three Scouts on a robot. When there was conflicting data between all three Scouts, consolidation would use the value reported by the more accurate Scout. However, Scouts could still agree on the wrong data, meaning that incorrect decisions were counted as correct while calculating SPR. To avoid this, this year's SPR formula compares Scout data against data from The Blue Alliance.

TBA only reports official scores for alliances, not individual teams, so Scout precision calculations use the combined scores of Scouts on an alliance. Since there are nine Scouts per alliance, with three on each robot, there are 27 different possible combinations of three Scouts that will contain one Scout from each robot on the alliance. For each of these combinations, the cargo scores for each Scout are totaled to get the overall alliance score. The official alliance score is pulled from TBA (minus foul points, auto line points, and climb points) and then compared against the total scouted score to find the error. A match's average error is calculated by taking the average of all errors in all combinations.

Once the match average error has been calculated for each Scout in a match, the formula looks at each three-Scout combination that a specific Scout was in. The average errors of the other two Scouts in that combination are divided by 3 (since errors are the result of three-Scout combinations) and totaled to get the expected error of that combination. Then, the actual error of the combination, including the Scout in question, is subtracted from the expected error to find how much the focus Scout contributed to the error of that combination. The average of this value for all of a scout's combinations in a specific match is that scout's Scout-In-Match (SIM) Precision.

The average of a scout's SIM Precision values for all matches in a competition is that scout's overall Scout Precision value. The lower this value, the better, since it represents how much error on average a Scout contributed to their combinations.

For a more detailed example of the Scout Precision Ranking calculation, see Appendix C.

Device Data Pulling

Apart from QR scanning, data is also inputted to the Server by pulling it directly from devices plugged into the Server computer. The qr_input.py file automatically pulls qr data, which is downloaded onto tablets as a backup as well as being displayed, from any tablets which are plugged in when it is run. In addition, it also pulls JSON data and images from connected phones with data collected using the Pit Scout app. This data is stored in a directory within the Server, as well as inserted in the database. Once the Server has pulled all pit images, users of the Viewer app can plug their phones into the Server computer and have the images pushed to the Downloads folder of the phone using the send_viewer_images.py script. From there, Viewer can read and display those images.

VIEWER

Viewer is an Android app written in the Kotlin programming language. The app allows strategists to review, organize, and visualize processed scouting data live during competition in order to create educated match strategies.

Navigation

The Viewer uses a sidebar navigation view for selecting its primary pages. There are also multiple places in the pages where the user can navigate to other related pages, such as from the match schedule to the match details page. To optimize navigation, the sidebar can be viewed from any screen, including when the Viewer has opened multiple nested pages.

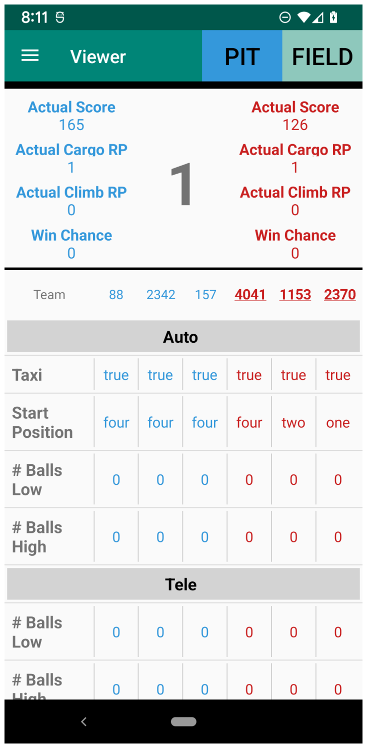
In addition to the navigation bar, a search bar is located directly at the top of the match schedule page, which filters the match schedule by that team. A user can submit the search button to go directly to the team's details page.

Robot Images

In order for strategists to quickly recall individual teams and distinguish among them, users can view the picture of a team's robot at the top of their team details page. These images are taken using the Pit Collection app and then locally downloaded onto every phone that has the Viewer (see Device Data Pulling in the Server section). By locating files using a predetermined naming system (see the Pit Collection section), the app can quickly display images without having to establish a connection to the Cardinal web server.

Match Details

After tapping on a match schedule cell, the match's details page is opened and consists of a header and data section. If the match has not been played, the data points displayed will be predictions and averages, and if it has been played, the data points will be the actual outcomes. In the header, alliance specific data is displayed (e.g. predicted score) and in the data section, team data is displayed in a table. By tapping a team number in the table, their team details page will open.



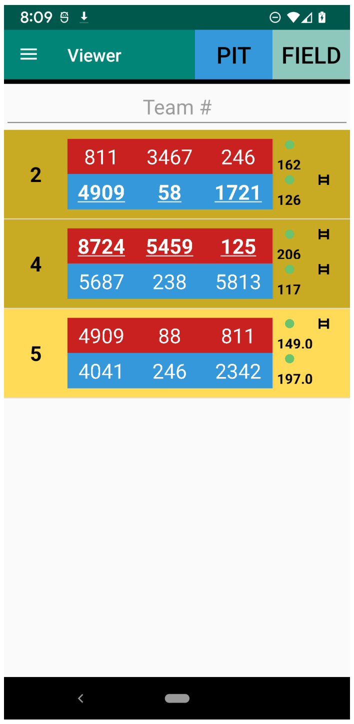
Actual Score 165 | Actual Score 126
Actual Cargo RP 1 | Actual Cargo RP 1
Actual Climb RP 0 | Actual Climb RP 0
Win Chance 0 | Win Chance 0

Team	88	2342	157	4041	1153	2370
Auto						
Taxi	true	true	true	true	true	true
Start Position	four	four	four	four	two	one
# Balls Low	0	0	0	0	0	0
# Balls High	0	0	0	0	0	0
Tele						
# Balls Low	0	0	0	0	0	0
# Balls High	0	0	0	0	0	0

Match Details screen

Starred Matches

Long pressing on a match in any match schedule screen will turn it yellow and make it appear in the starred matches list. The list of starred matches is stored in a local file in the device's phone's storage so when the app is reinstalled to update, starred matches will be saved.

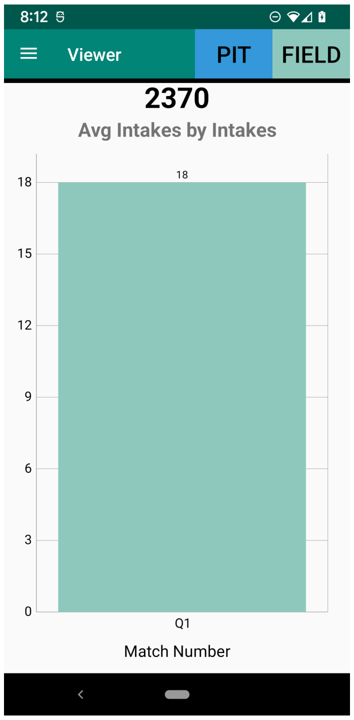


2	811	3467	246	162
	4909	58	1721	126
4	8724	5459	125	206
	5687	238	5813	117
5	4909	88	811	149.0
	4041	246	2342	197.0

Starred matches list

Team In Match Data Graphs

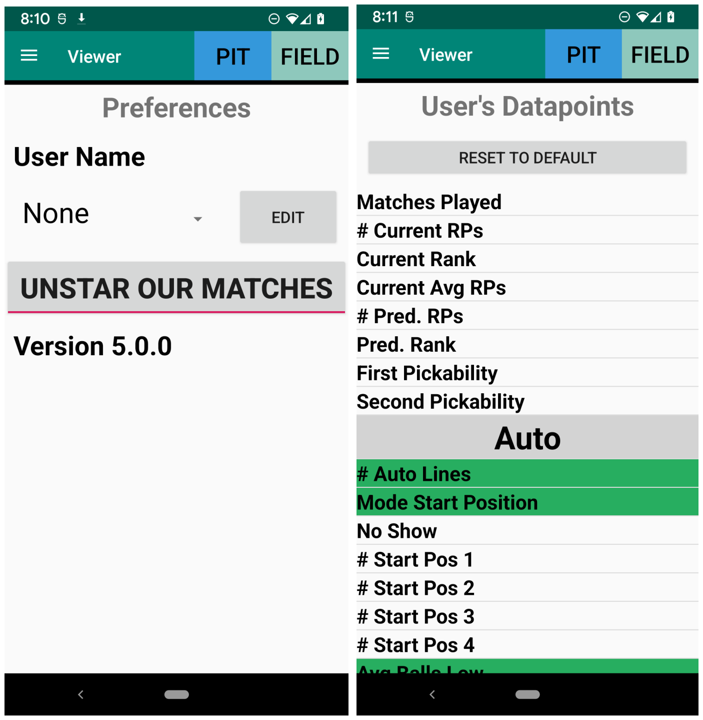
Tapping on a data point in team details opens up its TIM Data Graph—a bar graph of match number vs the data point's value. For example, by tapping on avgBallsScored, a graph will appear with the first bar showing the number of balls scored in their first match. Tapping on a bar opens its match details page.



TIM Data Graph

User Preferences

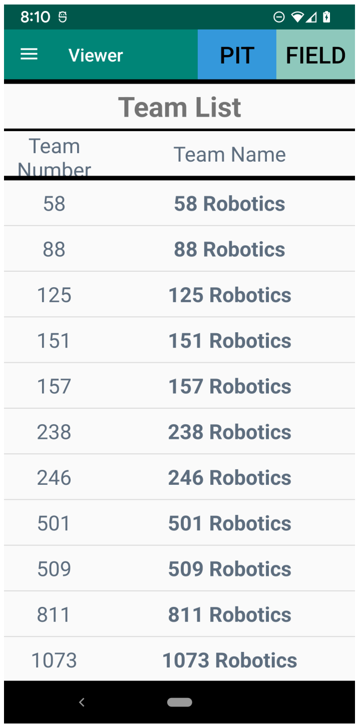
Users can select which data points to be displayed in match details, which is essential when users vary from a Match Strategist to a student assisting alliance partners in the pit. The Match Strategist may want to quickly view more data on a team's average intakes while mechanical members would rather view a team's motor type. Data points are saved in a file stored in the Downloads folder so they aren't reset after closing or updating the application. Regardless of which data points are selected in preferences, users can view all of a team's data in their team details page—Preferences only affect the match details page.



Preferences screen and User Preferences editable data points

Last Four Matches

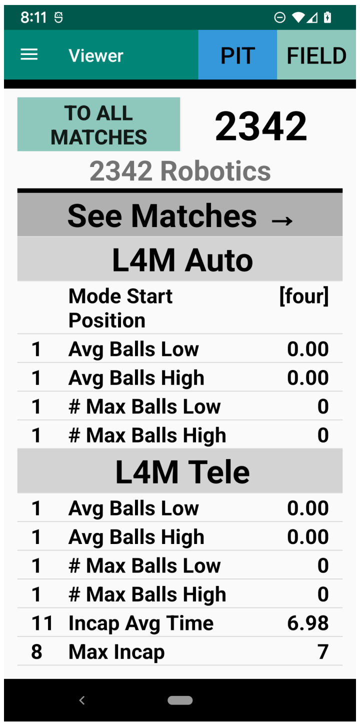
In team details, users can toggle between all-matches data points to the same data points but only calculated from a team's last four matches. This is used to estimate a team's performance during eliminations, since their performance at the beginning of the competition is likely lower than at the end of quals.


8:10 Viewer PIT FIELD

Team List

Team Number	Team Name
58	58 Robotics
88	88 Robotics
125	125 Robotics
151	151 Robotics
157	157 Robotics
238	238 Robotics
246	246 Robotics
501	501 Robotics
509	509 Robotics
811	811 Robotics
1073	1073 Robotics

Team List


8:11 Viewer PIT FIELD

TO ALL MATCHES 2342

2342 Robotics

See Matches →

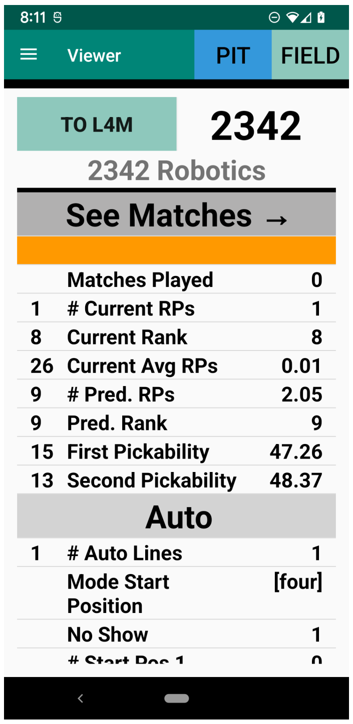
L4M Auto

	Mode Start Position	[four]
1	Avg Balls Low	0.00
1	Avg Balls High	0.00
1	# Max Balls Low	0
1	# Max Balls High	0

L4M Tele

1	Avg Balls Low	0.00
1	Avg Balls High	0.00
1	# Max Balls Low	0
1	# Max Balls High	0
11	Incap Avg Time	6.98
8	Max Incap	7

Last four matches screen


8:11 Viewer PIT FIELD

TO L4M 2342

2342 Robotics

See Matches →

	Matches Played	0
1	# Current RPs	1
8	Current Rank	8
26	Current Avg RPs	0.01
9	# Pred. RPs	2.05
9	Pred. Rank	9
15	First Pickability	47.26
13	Second Pickability	48.37

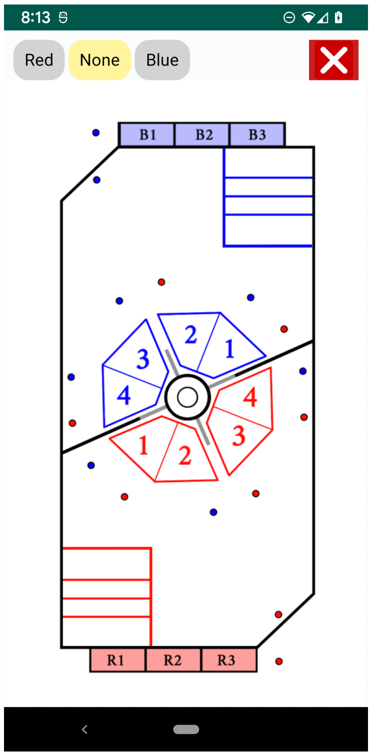
Auto

1	# Auto Lines	1
	Mode Start Position	[four]
	No Show	1
	# Start Pos 1	0

Team details screen

Field Map and Pit Map

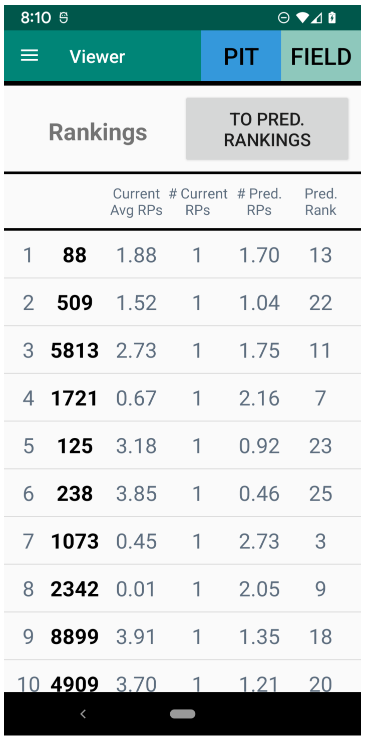
A pop-up of the game field map or the pit map can be accessed on any page via buttons in the header bar. This is largely for the Match Strategist to quickly refer to in the pits, without having to navigate away from the page they are currently on. Field maps are downloaded into the app's resource directory, and can be toggled between alliances to view perspective-specific maps. The pit map is downloaded into Downloads and then named pit_map (no extension) in order to be displayed.



Field Map

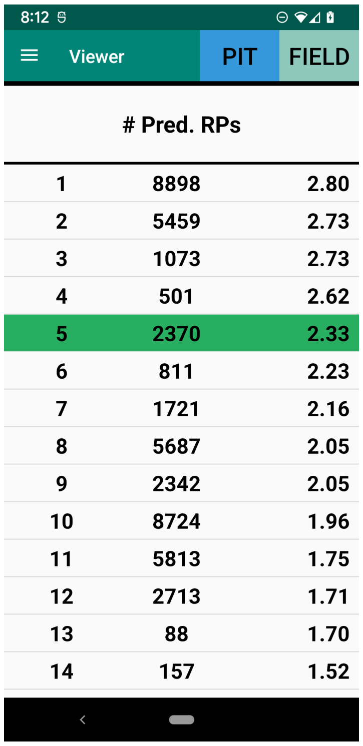
Rankings for Specific Data Points

By long pressing on a data point from the team details page, a user can view a ranked list of all the teams by that data point. These ranks are also displayed on the left side of team detail cells, but aren't refreshed as often to avoid lag. By viewing a ranked list, users can see which teams are below and above a certain team. By clicking on the cells in the ranked list, users can open a team's team details page.



		Current Avg RPs	# Current RPs	# Pred. RPs	Pred. Rank
1	88	1.88	1	1.70	13
2	509	1.52	1	1.04	22
3	5813	2.73	1	1.75	11
4	1721	0.67	1	2.16	7
5	125	3.18	1	0.92	23
6	238	3.85	1	0.46	25
7	1073	0.45	1	2.73	3
8	2342	0.01	1	2.05	9
9	8899	3.91	1	1.35	18
10	4909	3.70	1	1.21	20

Ranking teams by RP



# Pred. RPs

1	8898	2.80
2	5459	2.73
3	1073	2.73
4	501	2.62
5	2370	2.33
6	811	2.23
7	1721	2.16
8	5687	2.05
9	2342	2.05
10	8724	1.96
11	5813	1.75
12	2713	1.71
13	88	1.70
14	157	1.52

Ranking a team by a specific data point

Data Refreshing

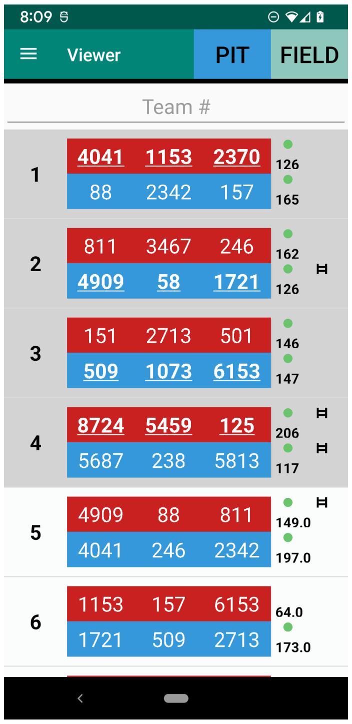
Data in the Viewer was originally fetched only on startup. Now, data refreshing automatically fetches the data and updates caches in a set time interval. When the data refreshes, callbacks are run in all active fragments in the backstacks which provide their own implementations for updating the UI.

Reading from Web Server and Authentication

The data in the Viewer is retrieved using a web server called Cardinal (see the Server section). Viewer can request different pieces of data from the database by their collection name. Each request is authenticated with an API key.

Predicted vs. Actual Scores and Ranking Points in Match Schedule

Cells of matches in the match schedule show up light gray after the match has finished and data on the match has been received by Viewer. The score shown in each cell and in the match details is the predicted score if the match has not been played yet and the actual score if the match has been played already. This is the same for predicted and actual ranking points.



1	4041	1153	2370	126
	88	2342	157	165
2	811	3467	246	162
	4909	58	1721	126
3	151	2713	501	146
	509	1073	6153	147
4	8724	5459	125	206
	5687	238	5813	117
5	4909	88	811	149.0
	4041	246	2342	197.0
6	1153	157	6153	64.0
	1721	509	2713	173.0

Match schedule with played matches colored gray

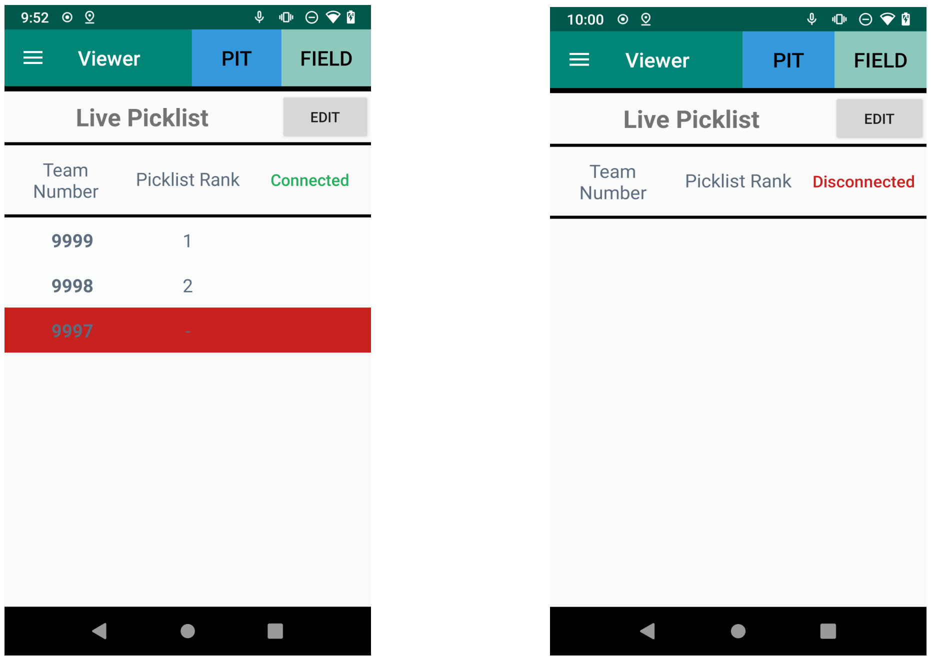
Strategist Notes

Our strategists previously used a spreadsheet to organize their notes on teams. Rather than sending information back and forth via a messaging app, Stands Strategists can note down information on a team in their team details page. The data will be stored on the Cardinal web server and pulled by all the other devices. This feature was added late to the season and not used very much—it will likely be redesigned to be more efficient for Stands Strategists next year.

Live Picklist

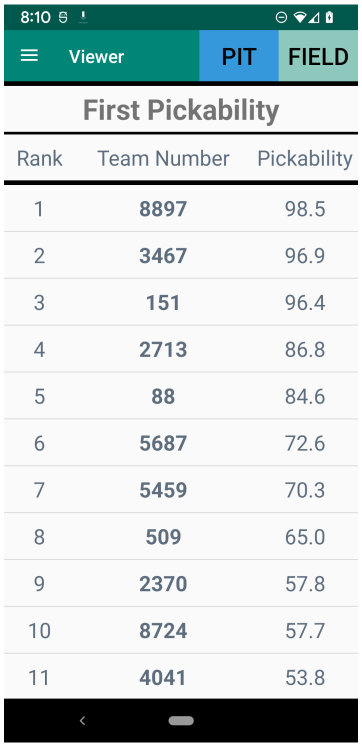
Live picklist allows users to view and edit the picklist from the Viewer. After the picklist meeting, picklist rankings are exported from the Picklist Editor spreadsheet, processed by the Server, and then uploaded to the picklist collection in MongoDB. A new Python web server using FastAPI called Grosbeak was created to allow CRUD to access the picklist collection using a websocket connection. Websockets are authenticated using an API key different from Cardinal. Once a connection is initialized, the server sends the picklist information to the Viewer. Whenever the picklist is edited, the information gets sent to all clients currently connected and the UI gets updated. To start editing the picklist, a message is sent containing a password that is verified on the server side.

Live picklist was developed for the last regional in the season and still has large amounts of testing and improvement to undergo. It was functional at that competition, but had trouble with a stable connection. It will likely be improved during the offseason or next season.



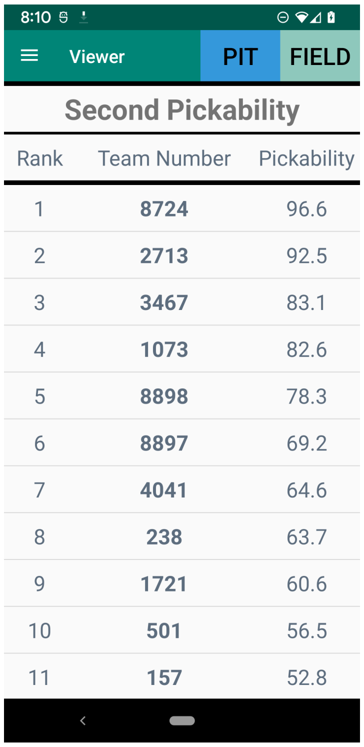
Team Number	Picklist Rank
9999	1
9998	2
9997	-

Live Picklist connected and disconnected



First Pickability

Rank	Team Number	Pickability
1	**8897**	98.5
2	**3467**	96.9
3	**151**	96.4
4	**2713**	86.8
5	**88**	84.6
6	**5687**	72.6
7	**5459**	70.3
8	**509**	65.0
9	**2370**	57.8
10	**8724**	57.7
11	**4041**	53.8



Second Pickability

Rank	Team Number	Pickability
1	**8724**	96.6
2	**2713**	92.5
3	**3467**	83.1
4	**1073**	82.6
5	**8898**	78.3
6	**8897**	69.2
7	**4041**	64.6
8	**238**	63.7
9	**1721**	60.6
10	**501**	56.5
11	**157**	52.8

First and second pickability screens

PICKLIST EDITOR

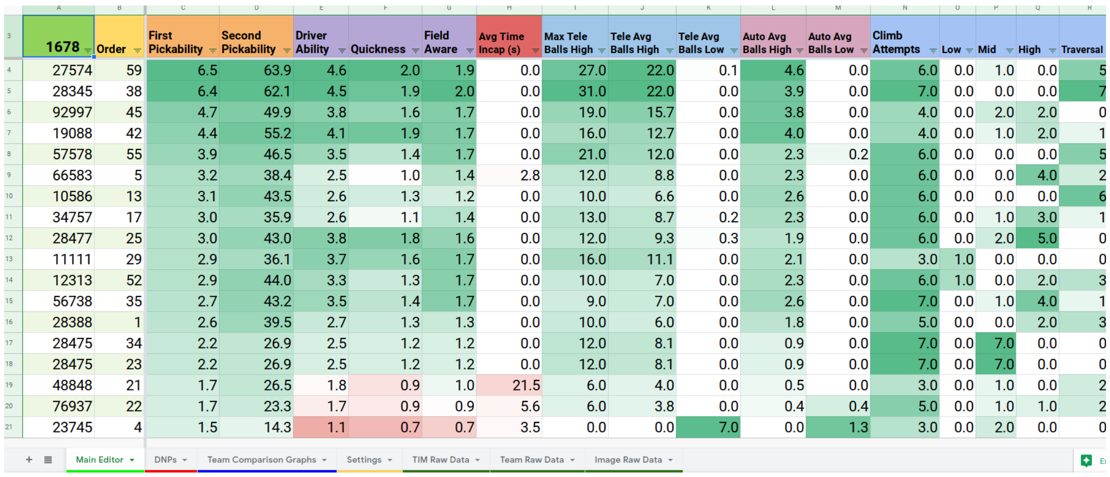
The Picklist Editor is used by 1678 during its picklist meetings to construct a team picklist. The app is built on Google Sheets and uses scripts written with Google Apps Script.

Team Rank Ordering

In the main editor, a list of the teams in the competition is shown in the first column, initially ordered by their ‘pickability’ ratings, based on either first or second pick rank scores (for more information on how pickability is calculated, see Pickability in the Server section). Ranks are displayed in the first two columns, but are typically hidden throughout the meeting, since they are only a starting place for the picklist order.

Each row displays the data for its respective team. The data is taken directly from a separate raw data sheet and the data displayed is selected in another page of the spreadsheet.

To reorder the teams, the picklist operator edits the ranking number (e.g. to move a team in between first and second, change their rank to 1.5). Then, the scripts behind the Picklist Editor will automatically reorder the teams and update their ranks to whole numbers. The spreadsheet is designed for bubble sorting, or starting from the top and working downwards.

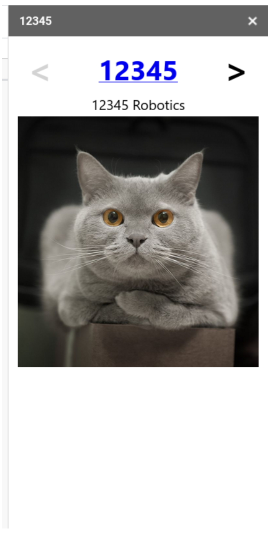


1678	Order	First Pickability	Second Pickability	Driver Ability	Quickness	Field Aware	Avg Time Incap (s)	Max Tele Balls High	Tele Avg Balls High	Tele Avg Balls Low	Auto Avg Balls High	Auto Avg Balls Low	Climb Attempts	Low	Mid	High	Traversal
27574	59	6.5	63.9	4.6	2.0	1.9	0.0	27.0	22.0	0.1	4.6	0.0	6.0	0.0	1.0	0.0	5
28345	38	6.4	62.1	4.5	1.9	2.0	0.0	31.0	22.0	0.0	3.9	0.0	7.0	0.0	0.0	0.0	7
92997	45	4.7	49.9	3.8	1.6	1.7	0.0	19.0	15.7	0.0	3.8	0.0	4.0	0.0	2.0	2.0	0
19088	42	4.4	55.2	4.1	1.9	1.7	0.0	16.0	12.7	0.0	4.0	0.0	4.0	0.0	1.0	2.0	1
57578	55	3.9	46.5	3.5	1.4	1.7	0.0	21.0	12.0	0.0	2.3	0.2	6.0	0.0	0.0	0.0	5
66583	5	3.2	38.4	2.5	1.0	1.4	2.8	12.0	8.8	0.0	2.3	0.0	6.0	0.0	0.0	4.0	2
10586	13	3.1	43.5	2.6	1.3	1.2	0.0	10.0	6.6	0.0	2.6	0.0	6.0	0.0	0.0	0.0	6
34757	17	3.0	35.9	2.6	1.1	1.4	0.0	13.0	8.7	0.2	2.3	0.0	6.0	0.0	1.0	3.0	1
28477	25	3.0	43.0	3.8	1.8	1.6	0.0	12.0	9.3	0.3	1.9	0.0	6.0	0.0	2.0	5.0	0
11111	29	2.9	36.1	3.7	1.6	1.7	0.0	16.0	11.1	0.0	2.1	0.0	3.0	1.0	0.0	0.0	0
12313	52	2.9	44.0	3.3	1.3	1.7	0.0	10.0	7.0	0.0	2.3	0.0	6.0	1.0	0.0	2.0	3
56738	35	2.7	43.2	3.5	1.4	1.7	0.0	9.0	7.0	0.0	2.6	0.0	7.0	0.0	1.0	4.0	1
28388	1	2.6	39.5	2.7	1.3	1.3	0.0	10.0	6.0	0.0	1.8	0.0	5.0	0.0	0.0	2.0	3
28475	34	2.2	26.9	2.5	1.2	1.2	0.0	12.0	8.1	0.0	0.9	0.0	7.0	0.0	7.0	0.0	0
28475	23	2.2	26.9	2.5	1.2	1.2	0.0	12.0	8.1	0.0	0.9	0.0	7.0	0.0	7.0	0.0	0
48848	21	1.7	26.5	1.8	0.9	1.0	21.5	6.0	4.0	0.0	0.5	0.0	3.0	0.0	1.0	0.0	2
76937	22	1.7	23.3	1.7	0.9	0.9	5.6	6.0	3.8	0.0	0.4	0.4	5.0	0.0	1.0	1.0	2
23745	4	1.5	14.3	1.1	0.7	0.7	3.5	0.0	0.0	7.0	0.0	1.3	3.0	0.0	2.0	0.0	0

Picklist Editor

Sidebar

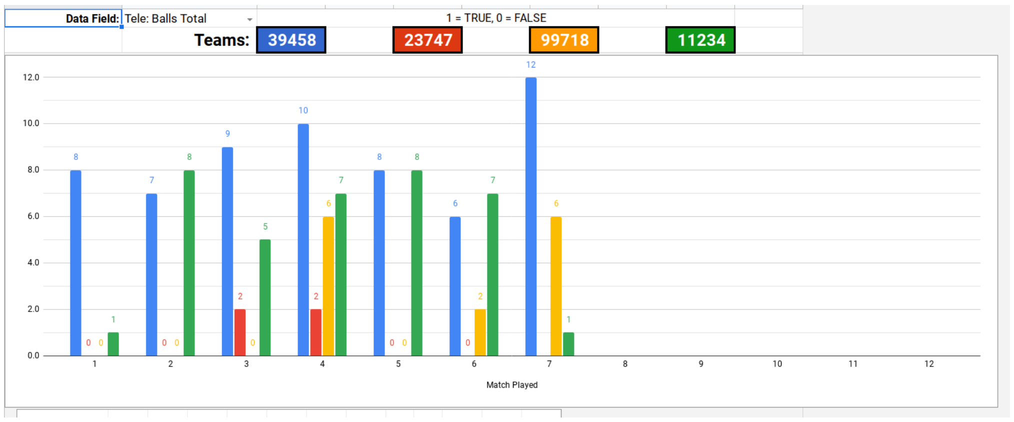
The Picklist Editor provides a sidebar feature to display pictures of robots. Whenever a change is made in the order, the next team's robot pictures are shown in the sidebar, allowing the members of the picklist meeting to more easily remember which robot is which. The picklist operator can also edit the team number in the very top left corner cell to manually choose an image.



Picklist Editor sidebar

Team Performance Comparison Graphs

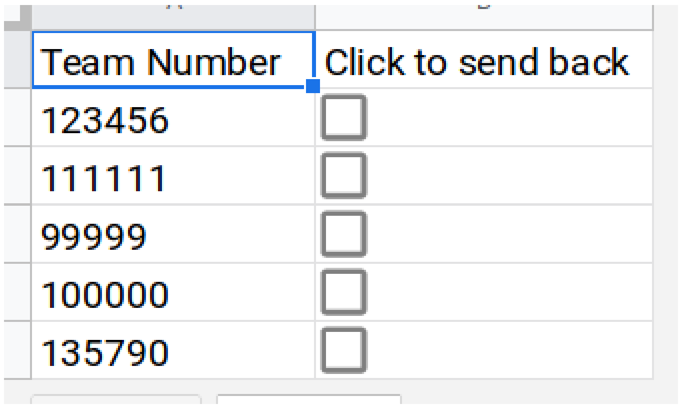
When a match-by-match comparison between teams is necessary, the Picklist Editor has a graphing feature to compare the data of up to four teams in a single data point. The graphs can show changes in data over multiple matches.



Team performance comparison bar graph

Removing teams from the Picklist

In order to limit the teams to rank—to save time, Strategists determine teams that aren't performing at the level they'd like for 1678's alliance and remove them at the beginning of the meeting from the picklist. These teams are discussed before the picklist meeting, between the Match Strategist, Stands Strategists, and strategy mentors. By typing "DNP" in their order, a team is removed from the list on the main page. In case the team is later reconsidered, the operator can send the team back by checking its box.



Team Number	Click to send back
123456	☐
111111	☐
99999	☐
100000	☐
135790	☐

DNP Editing

Updating Data Points Used in the Ranking Process

The data points shown in the editor as well as their order shown changed drastically throughout the season based on feedback from students and mentors in picklist meetings. Due to the way that the Picklist Editor is set up, it was not difficult to add or remove data points in the middle of picklist meetings, although strategists did attempt to determine all data points beforehand to save time.

Operational Steps

Before 1678's picklist meetings, the picklist operator is sent raw data in CSV format from the Server. During picklist meetings, the picklist operator displays the Picklist Editor through a projector.

Then, starting with the first pick order from the ranking equations, the teams are reviewed starting at the top and compared on a pairwise basis progressing down the list, deciding for each team whether it should be moved up the picklist and by how many places. Once the potential first picks have been ranked (usually down to 10 or 12 teams), the rows containing those teams are hidden from the editor and the potential second picks are then ranked in the same manner. Once the re-ranking process is complete, a CSV export of the final picklist is sent back to the Server. At a regional, where the event continues in the morning of the second day, strategists make further changes to the picklist based on further information and closer review of specific teams.

Google Apps Script

The Picklist Editor is built on Google Apps Script, which is very similar to the JavaScript language. In fact, using the official command line interface Clasp, it is possible to clone the scripts into a local development environment as JavaScript files. This season, the scripts were restructured to be more organized, and they were rewritten using the TypeScript programming language. TypeScript is a powerful superset of JavaScript which allows for static type checking and increased safety.

For more information on the Picklist Editor's scripts, see the Picklist Editor repository on GitHub.

Future Updates

In future seasons, we are planning to make improvements to the Picklist Editor such as better sidebar performance in pulling and a central logging system.

VIDEO SYSTEM

The Video System is a vital part of data collection and strategization by allowing strategists to review and rewatch matches—regardless of The Blue Alliance's speed in uploading videos and the video's angle. Those in charge of the Video System take videos of all qualification and elimination matches for further analysis. These match videos allow strategists to create match strategies for upcoming matches, provide feedback to drivers, and rewatch a team's performance in a specific match during the picklist meeting.

Video System operators start by setting up the tripod and camera in a high-up and centered position overlooking the match field. It uses a script written in Python to name (based on the inputted match number and team numbers based on the match schedule) and copy the video from the SD card to a USB drive.

During elimination matches, videos are sent via Slack to match strategists on or near the field. To gain access to the internet, a Video System operator tethers their laptop to a mobile phone with cell service. This allows strategists to rewatch matches and develop a better strategy for the next match in almost no time.

APPENDICES

Appendix A: Production Schedule

Prior to Kickoff — All Citrus Circuits students work on either training or preparing for an offseason competition. We take inventory of all materials before each season and order any needed supplies.

1/8/2022 — All Citrus Circuits students watch the Kickoff broadcast and participate in a full-team discussion about what 1678 will attempt to do in the new season.

1/9/2022 — The Scouting subteam meets with strategy members to determine data points to display, and then what data points are necessary to collect in order to process them. For each data point on the final list, the following information is noted down:

- The data type
- A description of what the data point represents
- Some example values

- Which database collection it would be stored in
- Whether it would be collected raw, or if it needed to be calculated from other data points
    - If it was a raw data point, how it would be collected (by Scouts, Super Scouts, Pit Scouts, Stands Strategists, or the The Blue Alliance API)
- If it would be displayed in the Viewer, and if so, how it would be visualized

1/9/2022 to 1/15/2022 — Back-End students update the schema files to contain the new data points for the 2022 season. Front-End students update the Match Collection and Pit Collection app with new data and UI designs.

1/15/2022 to 1/30/2022 — Back-End students updated the calculation files to match the new schema. Front-End students finish the first version of the collection apps and begin to update the Viewer and add new features.

1/30/2022 to 2/27/2022 — Software Scouting begins to conduct end-to-end field tests to ensure the system runs together. Front-End collects user feedback on the UI of apps. The list of data points to calculate/collect is updated based on strategy discussions. Students watch xRC Simulator matches to test out the scouting system before real match videos are available. Coming closer to competition season, Scouts and Super Scouts are trained on how to use the Match Collection app.

2/27/2022 to 4/23/2022 — During competition season, the Wednesday or Sunday before each competition is a feature freeze, a deadline that completely stops development on all new features until after the competition. The subteam runs a full-system test before every competition in order to catch and fix last-minute bugs. On the first meeting after returning from each competition, the full subteam participates in a debrief and communicates with users and mentors in order to prioritize which changes to make before the next competition.

Appendix B: How to Run a 1678 System Test

1. Before the test, collect match videos. Ideally, these are high-scoring matches from a previous regional/district competition. If the competition season hasn't started yet, use Week 0 videos or screen recordings of the xRC Simulator.

2. Make a new Server branch to merge in any hotfixes that may need to be made during the field test. Name it with the date of the field test. Pull any unmerged PRs that need to be tested onto this branch.

3. Decide on a TBA event key to use (eg. 2022cada for the 2022 Sacramento Regional). This will ideally match the event used for the videos, but it doesn't have to — it can be an event from a previous year as long as it has a match schedule and team list. Use the event key to create a test database and team list and match schedule files.

4. Set up the system as it would be at competition and send the newest .apk files to the tablets and phones.

5. Recruit volunteers to use the Scout and Super Scout apps to collect data from match videos. If there are fewer than 20 students available, have people use multiple tablets at once. The main concern is not to get accurate data, but to thoroughly use every feature of the collection apps. The match videos are only there as a guide, so it's okay if the team number assigned to a Scout is different from the robot they are scouting. Have a user enter test data using the Pit Collection app.

6. After each match, scan the data into the Server. Make sure data is being entered into the local and cloud databases, and that the web server is able to send it to the Viewer. Monitor the Viewer to make sure data is updating and being displayed correctly.

7. Write down every bug or suggestion for improvement as it comes up, no matter how small. Afterwards, go through the list and prioritize which ones to address first.

Appendix C: SPR Calculation Walkthrough

Consider a Scout named X. To calculate X's SPR, the formula starts with a single match that X scouted in. It finds two other Scouts in that match, one scouting each of the other teams in the alliance. Now, it adds the scout-reported scores from the three Scouts together to calculate a theoretical alliance score and compares it against The Blue Alliance's official alliance score to get that combination's error. For example, in a combination where X said Robot 1 scored 12 points, Y said Robot 2 scored 31 points, and Z said Robot 3 scored 10 points, and TBA reports that the entire alliance scored 50 points, that combination's error is 3 points.

Then, the formula takes the error of another combination with X and two different Scouts on the other two teams. This process repeats until it has gone through all the possible combinations containing X and the other two teams that X didn't scout. The average error of all these combinations is X's average combination error in that match. The entire process is repeated for all Scouts in that match to find their average combination errors.

Then, the formula returns to look at each individual combination that X was in. For each combination, it finds the average combination errors of the other two Scouts and divides each by 3. It sums the two errors to get the expected error of that combination. For example, in a combination with X, Y, and Z, where Y has an average error of 15 and Z has an average error of 4, it would find the expected error of that combination to be 6.33.

Then, it subtracts the error from the specific combination from the expected error. If X had been completely accurate, the error from that single combination should be similar to the expected error, so the result should be close to 0. If X had been off, they would have contributed further error to the combination error, so the result would be less than or greater than 0 depending on how far above or under they were. If the error of the XYZ combination was 6.33, then the formula would determine that X did not contribute any extra error on top of the 6.33 that was already contributed by Y and Z. If the error of the XYZ combination was 7, then the formula would determine that X contributed approximately 0.66 extra points in error.

This process is repeated for every Scout in that match, over all matches in a tournament. Each scout's average score across all their matches is averaged to calculate their overall Scout precision.

Appendix D: Subteam Structure

Students on Software Scouting are split into either Back-End or Front-End. Each end has its own student lead, who is chosen by the previous year's team captains and head mentors along with all other subteam leads. Back-End students code mostly in Python and develop the Server and Schema, while Front-End students mainly use Kotlin and create all the apps that users interact with. However, Software Scouting is still considered to be a single subteam, and students on both ends regularly review each other's code and participate in full subteam discussions and system tests. Some students also occasionally write code for the opposite end. Veteran members serve as guides for newer members, and students are often paired to work together on tasks. The Software Scouting subteam works closely with the Strategy subteam, and the majority of scouting developers are also members of the Strategy subteam.

Appendix E: Competition Roles

Roles at competition are broken up into developers, scouts, operators, and strategists. These are all student roles, with the exception of mentors assisting Stands Strategists, but even then the students are the primary voices.

Developers — One Front-End developer and one Back-End developer. Two Objective Scouts also serve as back-up developers who primarily Scout but can assist in case there are many issues.

Scouts — Since 18 Objective Scouts (a.k.a. Scouts) are needed per match, about 21 Scouts travel to each competition in order to give three Scouts a break at a time. In addition, there are three Super Scouts (a.k.a. Super Scouts) so that one can be on break while the other two scout. Since Super Scouts are required to have extensive experience on the Strategy subteam, the student on break often chooses to help other strategists. The lead Scout manages all of the logistics for the Scouts: meals, shifts, handing out tablets, and anything else they might need.

Operators — One student Scout who operates the spreadsheet during picklist meetings, as well as two Video System operators (who double as back-up Scouts), who record and label each match (see Video System section).

Strategists — Two Stands Strategists write specific notes on each team at competition. The subject of their notes changes year-to-year, but this year they largely focused on teams' defensive ability and any matches they played defense in. At regionals, Stands Strategists tweak the picklist depending on teams' performance before alliance selection, on the second day of competition. The Match/Pit Strategist uses the Pit Collection app during practice matches and then plans and communicates match strategies during the rest of the days of competition.

Appendix F: Resources

- Old Whitepapers
  - https://www.citruscircuits.org/scouting.html
- Software Scouting Software
  - Server code: https://github.com/frc1678/server-2022-public/tree/main
  - Cardinal code: https://github.com/frc1678/cardinal-2022-public/tree/main
  - Grosbeak (Live Picklist) code: https://github.com/frc1678/grosbeak-2022-public/tree/main
  - Viewer app: https://github.com/frc1678/viewer-2022-public/tree/release
  - Picklist Editor code: https://github.com/frc1678/picklist-editor-2022-public/tree/main
  - Schema files: https://github.com/frc1678/schema-2022-public/tree/main
  - Pit Collection app: https://github.com/frc1678/pit-collection-2022-public/tree/main
  - Match Collection app: https://github.com/frc1678/match-collection-2022-public/tree/main
- Fall Workshops
  - Link to 2020 Fall Workshops: https://www.youtube.com/watch?v=oioWQnJdvQo&list=PL6j32uphg3L9imPGEGz-dHhFFkKUAjFrm

- Link to 2015 Fall Workshop on Scouting Philosophy: https://youtu.be/eytf7ngfm0A
- Link to 2015 Fall Workshop on Scouting Philosophy presentation: https://www.citruscircuits.org/uploads/6/9/3/4/6934550/1678_fall_scouting_workshop.pptx_2.pdf
- You can view previous years' Fall Workshops on the Citrus Circuits

  website: https://www.citruscircuits.org/fallworkshops.html

  and in the playlists tab on the 1678 YouTube channel:

  https://www.youtube.com/c/CitrusCircuits/playlists
- Contact us if you have any questions!

  softwarescouting@citruscircuits.org

Appendix G: Starting a Scouting System

We recommend teams to start with a small system structure: you can use a web app or paper and pencils—whichever is easiest for you. Citrus Circuits has successfully used a Google Forms scouting system at off-season events to train new members in scouting principles and methods. Then, prioritize training your Scouts. We highly recommend finding at least one hour a week where your Scouts can watch match videos and then discuss with one another: What would you have done differently? What team did really well? What were some small mistakes? Who had the best speed?

If you are able to spare only about two members of your team for scouting, then we recommend either having each one take notes about the ability of one alliance (training is very important when you are recording qualitative notes) or you can reach out to fellow teams and gauge interest in forming a scouting alliance. Each team can get a copy of the data to review for their matches and picklist meeting, and none of them have to give up a bunch of members. If you are a part of a scouting alliance, however, try to create a uniform training method (e.g. schedule a two-hour Zoom training where they practice taking notes or using your scouting app).

If you have any questions at all about starting your own scouting system, want to get our team's advice given your resource level, or have any other questions, please contact us at softwarescouting@citruscircuits.org.

Appendix H: Hardware

In order to use apps at competitions we were required to purchase different pieces of hardware. These are 40 tablets for Match Collection, four Android phones for Pit Collection and

Viewer, a laptop to run the Server, three scanners to scan QR codes, four cases for transportation (two for tablets and one for each of the Video System and Server), and multiple SD cards with phone plans. If you want to know more about the hardware please look at section 4.2 of the 2020 Whitepaper.

Appendix I: Codebook

TEAM IN MATCH DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
auto_high_balls	Integer	The number of balls scored by a team in the high goal in auto
auto_low_balls	Integer	The number of balls scored by a team in the low goal in auto
auto_total_balls	Integer	The number of balls scored by a team in either goal in auto
climb_attempts	Integer	The number of times a team attempted to climb (0 if they did not attempt to climb, 1 if they attempted to climb at any point)
climb_level	String	The rung level a team climbed to in endgame (None, Low, Mid, High, or Traversal)
confidence_rating	Integer	The number of individual scouts datasets that contributed to the consolidated data (3 if all 3 scouts were included, 2 or 1 if there was missing scout data)
incap	Integer	The amount of time a team spent incap, disregarding incap periods of fewer than 8 seconds
intakes	Integer	The number of balls a team intook
match_number	Integer	The number of the qualification match
start_position	String	The position a team started from at the start of the match (Zero for no-show, One, Two, Three or Four)
team_number	Integer	The team's identifying number
tele_high_balls	Integer	The number of balls scored by a team in the high goal in tele
tele_low_balls	Integer	The number of balls scored by a team in the low goal in tele
tele_total_balls	Integer	The number of balls scored by a team in either goal in tele

OBJECTIVE TEAM DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
auto_avg_high_balls	Float	The average number of balls scored in the high goal by a team in auto
auto_avg_low_balls	Float	The average number of balls scored in the low goal by a team in auto
auto_avg_total_balls	Float	The average number of balls scored in either goal by a team in auto
auto_max_high_balls	Float	The maximum number of balls scored in a single match in the low goal by a team in auto
auto_max_low_balls	Float	The maximum number of balls scored in a single match the high goal by a team in auto
auto_sd_high_balls	Float	Calculated by finding the number of autonomous high goals a team scored in each of their matches and taking the standard deviation of that dataset
auto_sd_low_balls	Float	Calculated by finding the number of autonomous low goals a team scored in each of their matches and taking the standard deviation of that dataset
avg_climb_points	Float	The average number of points a team earned from climbing in each match
avg_incap_time	Float	The average amount of time a team spent incapacitated in each match, disregarding incap periods of fewer than 8 seconds
avg_intakes	Float	The average number of balls a team intook in each match
climb_all_attempts	Integer	The total number of times a a team attempted to climb, regardless of success
climb_percent_success	Float	The percent of climb attempts that resulted in a successful climb at any level
high_rung_successes	Float	The total number of successful climbs to the high rung
lfm_auto_avg_high_balls	Float	The maximum number of balls scored in a single match in the high goal in auto during a team's most recent 4 matches
lfm_auto_avg_low_balls	Float	The maximum number of balls scored in a single match in the low goal in auto during a team's most recent 4 matches

lfm_auto_max_high_balls	Integer	The maximum number of balls scored in a single match the high goal by a team in auto in their most recent 4 matches
lfm_auto_max_low_balls	Integer	The maximum number of balls scored in a single match in the low goal by a team in auto in their most recent 4 matches
lfm_avg_incap_time	Float	The average amount of time a team spent incapacitated, disregarding incap periods of fewer than 8 seconds, in their most recent 4 matches
lfm_climb_all_attempts	Integer	The total number of times a a team attempted to climb, regardless of success, in their most recent 4 matches
lfm_climb_percent_success	Float	The percent of climb attempts that resulted in a successful climb at any level in a team's most recent 4 matches
lfm_high_rung_successes	Integer	The total number of successful climbs to the high rung in a team's most recent 4 matches
lfm_low_rung_successes	Integer	The total number of successful climbs to the low rung in a team's most recent 4 matches
lfm_matches_incap	Integer	The total number of matches where a team was incapacitated, disregarding incap periods of fewer than 8 seconds, in their most recent 4 matches
lfm_max_climb_level	String	The highest rung a team successfully climbed to in their most recent 4 matches
lfm_max_incap	Integer	The longest period of time a team spent incap in a single match, disregarding incap periods of fewer than 8 seconds, in their most recent 4 matches
lfm_mid_rung_successes	Integer	The total number of successful climbs to the mid rung in a team's most recent 4 matches
lfm_mode_start_position	String	The most common position(s) a team started from in their most recent 4 matches
lfm_tele_avg_high_balls	Float	The average number of balls scored in the high goal by a team in tele in their most recent 4 matches
lfm_tele_avg_low_balls	Float	The average number of balls scored in the low goal by a team in tele in their most recent 4 matches

lfm_tele_max_high_balls	Integer	The maximum number of balls scored in a single match in the high goal in tele during a team's most recent 4 matches
lfm_tele_max_low_balls	Integer	The maximum number of balls scored in a single match in the low goal in tele during a team's most recent 4 matches
lfm_traversal_rung_successes	Integer	The total number of successful climbs to the traversal rung in a team's most recent 4 matches
low_rung_successes	Integer	The total number of successful climbs to the low rung
matches_incap	Integer	The total number of matches where a team was incapacitated, disregarding incap periods of fewer than 8 seconds
matches_played	Integer	The total number of matches a team played in
matches_played_defense	Integer	The number of matches where a team played defense
max_climb_level	String	The highest rung a team successfully climbed to
max_incap	Integer	The longest period of time a team spent incap in a single match, disregarding incap periods of fewer than 8 seconds
mid_rung_successes	Integer	The total number of successful climbs to the mid rung
mode_climb_level	String	The most common rung(s) a team successfully climbed to
mode_start_position	String	The most common position(s) a team started from
position_zero_starts	Integer	The number of matches where a team was a no-show (did not start on the field)
position_one_starts	Integer	The number of matches where a team started from Position #1
position_two_starts	Integer	The number of matches where a team started from Position #2
position_three_starts	Integer	The number of matches where a team started from Position #3
position_four_starts	Integer	The number of matches where a team started from Position #4
team_number	Integer	The team's identifying number
tele_avg_high_balls	Float	The average number of balls scored in the high goal by a team in tele
tele_avg_low_balls	Float	The average number of balls scored in the low goal by a team in tele

tele_avg_total_balls	Float	The average number of balls scored in either goal by a team in tele
tele_max_high_balls	Integer	The maximum number of balls scored in a single match in the low goal by a team in tele
tele_max_low_balls	Integer	The maximum number of balls scored in a single match the high goal by a team in tele
tele_sd_high_balls	Float	Calculated by finding the number of teleop high goals a team scored in each of their matches and taking the standard deviation of that dataset
tele_sd_low_balls	Float	Calculated by finding the number of teleop low goals a team scored in each of their matches and taking the standard deviation of that dataset
traversal_rung_successes	Integer	The total number of successful climbs to the traversal rung

PREDICTED TEAM DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
current_avg_rps	Float	The team's current official Ranking Score, as reported by TBA
current_rank	Integer	The team's current official seed, as reported by TBA
current_rps	Integer	The team's current official total RPs, as reported by TBA
predicted_rank	Integer	A team's predicted seed by the end of qualifications
predicted_rps	Float	The total number of RPs a team is expected to earn by the end of qualifications
team_number	Integer	The team's identifying number

PICKABILITY DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
first_pickability	Float	Calculated by taking a weighted average of a team's values for certain datapoints, measures approximately how suited a team would be as a first pick alliance partner
second_pickability	Float	Calculated by taking a weighted average of a team's values for certain datapoints, measures approximately how suited a team would be as a second pick alliance partner
team_number	Integer	The team's identifying number
test_first_pickability	Float	Allows strategists to experiment by changing datapoints and weights during competition and compare with the official first_pickability results without affecting the official first_pickability values
test_second_pickability	Float	Allows strategists to experiment by changing datapoints and weights during competition and compare with the official second_pickability results without affecting the official second_pickability values

PREDICTED ALLIANCE IN MATCH DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
actual_rp1	Float	TBA's reported official cargo RP for the alliance in match
actual_rp2	Float	TBA's reported official hangar RP for the alliance in match
actual_score	Integer	TBA's reported official score for the alliance in match
alliance_color_is_red	Boolean	Whether or not the alliance being predicted for is the red alliance in the match
final_predicted_rp1	Float	The predicted cargo RP at the time of the match being played, does not continue updating with new team data after the match was already played.
final_predicted_rp2	Float	The predicted hangar RP at the time of the match being played, does not continue updating with new team data after the match was already played.

final_predicted_score	Float	The predicted score at the time of the match being played, does not continue updating with new team data after the match was already played.
has_actual_data	Boolean	Whether or not TBA has provided official match results for the match
has_final_scores	Float	The predicted score at the time of the match being played, does not continue updating with new team data after the match was already played.
match_number	Integer	The number of the qualification match
predicted_rp1	Float	An alliance's predicted cargo RP. 1.0 if they were predicted to achieve it, 0.0 if not. Is a float to allow for possible fractional values in the future
predicted_rp2	Float	An alliance's predicted hangar RP. 1.0 if they were predicted to achieve it, 0.0 if not. Is a float to allow for possible fractional values in the future
predicted_score	Float	An alliance's predicted match score
win_chance	Float	The estimated probability of the alliance winning the match
won_match	Boolean	Whether or not the alliance actually won the match, as reported by TBA

SCOUT IN MATCH PRECISION DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
alliance_color_is_red	Boolean	Whether or not the team the scout reported data for was on the red alliance
match_number	Integer	The number of the qualification match
scout_name	String	The name of the scout
sim_precision	Float	The scout's precision score in a single match (SIM = Scout-in-Match)
team_number	Integer	The team's identifying number

SCOUT PRECISION DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
scout_name	String	The name of the scout
scout_precision	Float	The scout's overall scout precision score across the entire competition

TBA TEAM IN MATCH DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
auto_line	Boolean	Whether or not a team successfully crossed the auto line in auto, as reported by TBA
match_number	Integer	The number of the qualification match
team_number	Integer	The team's identifying number

TBA TEAM DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
team_number	Integer	The team's identifying number
team_name	String	The team's name
auto_line_successes	Integer	The total number of matches where a team successfully crossed the auto line in auto, as reported by TBA

OBJECTIVE PIT DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
can_climb	Boolean	Whether or not a team has a climbing mechanism
can_eject_terminal	Boolean	Whether or not a team is able to push balls to the human player through the terminal
can_intake_terminal	Boolean	Whether or not a team can intake balls from the human player through the terminal
can_under_low_rung	Boolean	Whether or not a team can drive under the low rung
drivetrain	Integer/Enum	The type of drivetrain a team has (0: tank, 1: mechanum, 2: swerve, or 3: other)
drivetrain_motor_type	Integer/Enum	The type of motors a team uses (0: minicim, 1: cim, 2: neo, 3: falcon)
drivetrain_motors	Integer	The number of motors a team has on their drivetrain
has_ground_intake	Boolean	Whether or not a team can intake balls off the floor
has_vision	Boolean	Whether or not a team has a vision system
team_number	Integer	The team's identifying number

SUBJECTIVE TEAM DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
driver_ability	Float	Calculated by taking the weighted average of a team's field awareness and quickness, then taking the z-score of that value compared to the values of all other teams at the competition
driver_field_awareness	Float	A team's field awareness score, adjusted based on the scores of the teams they played with (if they were compared against teams that had lower scores, their score was lowered to reflect that they had "easier" comparisons)
driver_quickness	Float	A team's quickness score, adjusted based on the scores of the teams they played with (if they were compared against teams that had lower scores, their score was lowered to reflect that they had "easier" comparisons)
match_number	Integer	The number of the qualification match
team_number	Integer	The team's identifying number
test_driver_ability	Float	Allows strategists to experiment by changing datapoints and weights during competition and compare with the official driver_ability results without affecting the offical driver_ability values
unadjusted_field_awareness	Float	A team's field awareness score, before being adjusted based on the scores of the teams they played with
unadjusted_quickness	Float	A team's quickness score, before being adjusted based on the scores of the teams they played with

SUBJECTIVE TEAM IN MATCH DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
alliance_color_is_red	Boolean	Whether or not the team the scout reported data for was on the red alliance
field_awareness_score	Integer	The ranking of a team's field awareness relative to their alliance partners (3 for most aware, 1 for least aware)
match_collection_version_number	String	The version of the Match Collection app that was used by the scout to collect the data
match_number	Integer	The number of the qualification match
played_defense	Boolean	Whether or not the team played defense against their opponents in the match
quickness_score	Integer	The ranking of a team's quickness relative to their alliance partners (3 for most quick, 1 for least quick)
schema_version	Integer	The version of the QR schema used to compress/decompress the QR
scout_name	String	The name of the scout
serial_number	String	The serial number of the tablet used by the scout to collect the data
team_number	Integer	The team's identifying number
timestamp	Integer	The time that the QR was generated, in Unix time

UNCONSOLIDATED OBJECTIVE TEAM IN MATCH DATASET

DATAPOINT	DATA TYPE	DESCRIPTION
schema_version	Integer	The version of the QR schema used to compress/decompress the QR
serial_number	String	The serial number of the tablet used by the scout to collect the data
match_number	Integer	The number of the qualification match
timestamp	Integer	The time that the QR was generated, in Unix time
match_collection_version_number	String	The version of the Match Collection app that was used by the scout to collect the data
scout_name	String	The name of the scout
alliance_color_is_red	Boolean	Whether or not the team the scout reported data for was on the red alliance

team_number	Integer	The team's identifying number
scout_id	Integer	The scout ID (1-18) of the tablet used to collect the data
start_position	String	The position a team started from at the start of the match (Zero for no-show, One, Two, Three or Four)
timeline	Array	A list of actions performed by a team in a match. Each action is a dictionary with the following keys- time, an integer between 150 and 0 representing how many seconds have passed since the start of the match, and action_type, a string representing the action performed by a robot (score_ball_high, score_ball_low, intake, start_incap, end_incap, or climb_attempt)
climb_level	String	The rung level a team climbed to in endgame (None, Low, Mid, High, or Traversal)

WHAT WE LEARNED

Our team has learned a lot of lessons we hope to apply to future years.

(1) Over the course of the season, we drastically reduced the number of data points collected, particularly by Objective Scouts. Based on Scout feedback, this made it much easier for Scouts to keep track of everything (the barrage of scoring in Rapid React was somewhat unexpected even after the Week 0 events) and know where all the buttons were in the app. Our data accuracy improved greatly as a result of the simplification of our system, so, in future years, we plan to prioritize even more the reduction of data points to only collect data that is truly necessary.

(2) Partway through this season, we began keeping a Schema Tracker spreadsheet that lists all the data points we are collecting, what app collects that data, the type, and when it was last updated by the different parts of our system. This greatly helped us keep track of the names of data points to make sure they were consistent across all parts of our system, helping to prevent data point naming errors. For the first half of the competition season, we frequently had these

issues during competitions, and had to hotfix data point names, particularly in the Viewer to ensure all the data points were displayed.

(3) We also learned more about the importance of documenting all of our subteam processes. Particularly when our team members graduate, it is useful to have information clearly laid out so nothing gets lost. This is also helpful for training new members and helping them learn more about our team.

(4) This year, we created a pre-competition checklist that includes every task we need to remember before a competition. Going through the checklist before every competition helps ensure nothing gets forgotten, such as clearing old data off of our tablets and packing the necessary elements of our system before we leave.

In the future, we hope to remember these lessons we've learned and use them to make our system better than ever. We have many new features we plan to implement in our apps, and we want to continue cleaning up our code and improving our systems every year.

Please note that all competition data shown in this document is fake and has been created for the sole purpose of this document. If you have any questions, want assistance, want help understanding our code, or have any other input, please reach out to us via email!