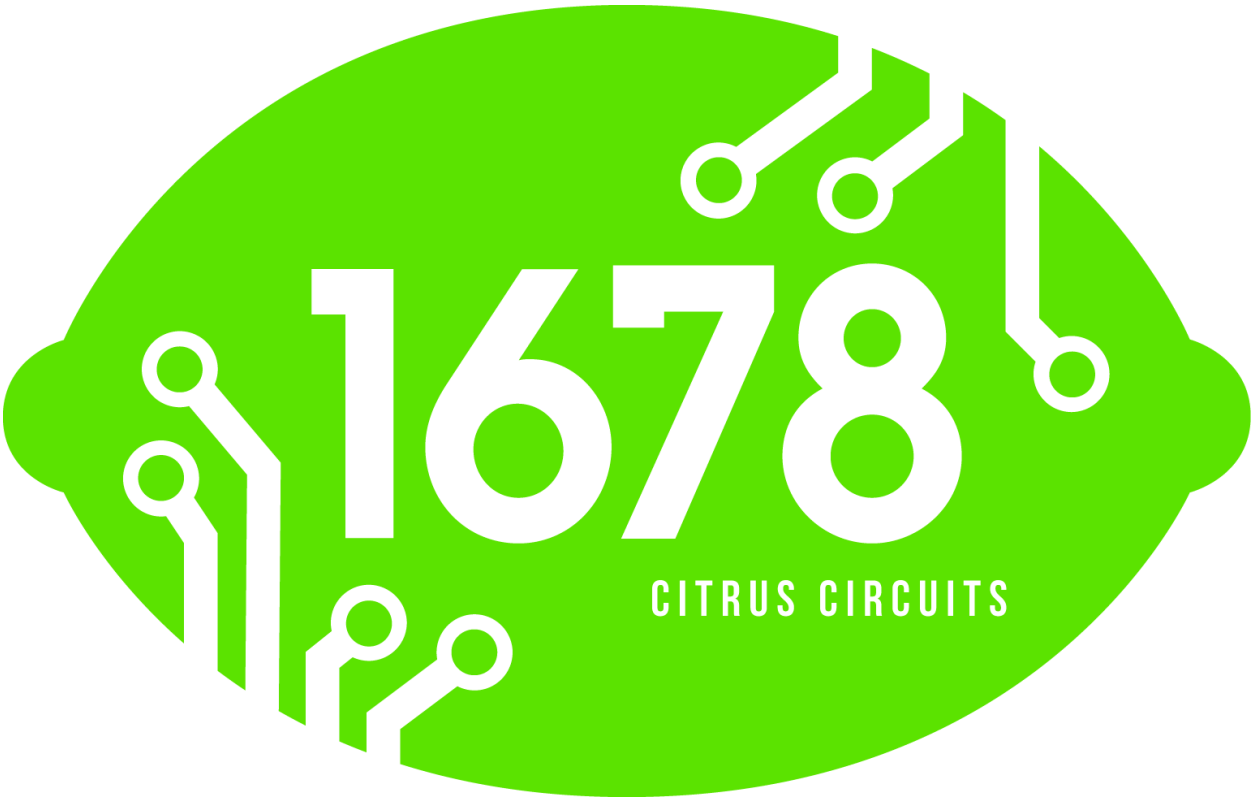


2016 Citrus Circuits
Electronic Scouting System

1678 App Programming Team:

Colin Unger, Bryton Moeller, Evan Long, Sam Chung
Abhijith Vemulapati, Samuel Resendez

August 3, 2016

Contents

Chapter 1

Introducation

1.1 History

Citrus Circuits first used a custom build electronic scouting system in 2013, and has used such a system every year since. Every year, we have released a whitepaper with the details of that year's scouting system. For past years, please reference the following whitepapers:

2013 White Paper: http://www.chiefdelphi.com/media/papers/download/3747
2014 White Paper: http://www.chiefdelphi.com/media/papers/download/4122
2015 White Paper: http://www.chiefdelphi.com/media/papers/download/4485

1.2 Background

1.2.1 Goal

The goal of Citrus Circuits' scouting system is to allow for seamless data input, processing, and visualization that can be used for both match strategy and alliance selection.

1.2.2 Why a Scouting System?

While a well performing robot is critical to success in an FRC competition, there are other elements that can help teams to reach a high competitive level. One of the most crucial of these is effective scouting and strategy. Ever since the 1999 FRC game Double Trouble, robots have been played on alliances of more than one robot. Since most FRC games are designed to prevent a single team from carrying the entire alliance[1], alliance strategy and alliance partners can determine the outcomes of matches and even whole competitions. Scouting systems allow teams to gather data which can help them to make better educated strategic decisions.

Our team's recent success is an indicator of the value of a well designed scouting system. Every year we have had a custom built electronic scouting system, we have reached Einstein field. While correlation does not imply causation, we believe that our scouting system has played a key role in helping us to achieve this level of competitive success.

1.2.3 Why Electronic?

Citrus Circuits used a paper and pencil system in 2012. However, we found that while data collection was effecient, processing the data was highly impractical. It is possible to use a spreadsheet to perform the calculations, but we have found that it can be far more efficient if consolidated into a single automated system that integrates data collection, processing, and visualization.

1.3 System Overview

1.3.1 Equipment and Costs

Our Equipment

Nexus 7 + T-Mobile SIM ($350) × 2 = $700
https://play.google.com/store/devices/details/Nexus_7_32GB_Black_Wi_Fi_Mobile_data_Unlocked_T_Mo?id=nexus_7_32gb_2013_lte_tmo

[1] For example, capturing in Stronghold (2016) and assists in Aerial Assist (2014).

Nexus 7 ($165) × 6 = $990
http://www.amazon.com/Nexus-Google-7-Inch-Black-Tablet/dp/B00DVFLJDS

MacBook Pro ($1099, but can be bought from other sources for cheaper) × 1 = $1099
http://store.apple.com/us/buy-mac/macbook-pro

Approximate Total: $3000

Budget System

Since the scouting system has a well-established place on our team, we have invested significant resources in purchasing high quality equipment. However, it is possible to run our system with a much cheaper setup that many teams should be able to afford. In order to show this, we have created a hypothetical set of low-cost equipment that would be capable of running a lightly modified version of our system. Note that we have not used any of this equipment and make no guarantees. It is, however, an example that an electronic system is not necessarily extravagantly expensive.

LG G Pad V410 ($120) × 2 = $240
https://play.google.com/store/devices/details/Nexus_7_32GB_Black_Wi_Fi_Mobile_data_Unlocked_T_Mo?id=nexus_7_32gb_2013_lte_tmo

Amazon Fire Tablet ($50) × 6 = $300
http://www.amazon.com/Nexus-Google-7-Inch-Black-Tablet/dp/B00DVFLJDS

Approximate Total: $540

Note: For both setups, these materials do not include the phones used for processing and viewing the data. Due to the widespread nature of mobile devices, it is highly likely that multiple team members would be willing to allow their phones to be used as part of the scouting system while at competition.

They also do not contain the data plans necessary for the system to upload data. However, the small amount of data uploaded means that low-cost plans can be used.

The MacBook Pro is only necessary for iOS programming. If the system was implemented solely in Android, the cost of the system could be significantly reduced, as in the budget system. However, a computer is still necessary to implement the system.

1.3.2 Necessary Capabilities

Experience in iOS, Android, and Python.

iOS Resources

In order to train new programmers in iOS, we use this excellent course provided by Stanford on programming iOS 8 apps: https://itunes.apple.com/bj/course/developing-ios-8-apps-swift/id961180099

Android Resources

We have found that Android often has a much steeper learning curve due to the more scattered and disorganized learning resources. Because of this, we have developed our own curriculum specifically geared toward giving new programmers the skills that they need to contribute to the Android components of the system. Note that this is not a full set of lessons on Android programming, since it only focuses on the skills that are very important for working with our system. This year, we have chosen to release all of these materials to the community on Github. If you are interested in using them or helping us to improve them, they can be found at https://github.com/frc1678/android-homework

For a more well-rounded introduction, we would recommend Android Developer website's official tutorial: `https://developer.android.com/training/index.html`

1.3.3 Foundations

The system relies on two main pieces of technology infrastructure: Firebase and Dropbox.

Firebase

Firebase is a data storage and syncing software that is intended to be reliable and easy to implement. It is responsible for the transfer and storage of all data other than photos in our system, allowing us to quickly and easily sync data across the various devices. Unlike our custom built syncing system of Realm and Dropbox last year, Firebase has been incredibly consistent, failing only once.

Dropbox

Dropbox is responsible for the transfer and storage of all of the robot photos. It is not as simple to implement as Firebase, but with some good code it can be very effective in its job.

1.3.4 System Components

The entire scouting system consists of seven different components. The basic flow of the scouting system can be broken up into three different sections: Collection, Processing, and Visualization.

Collection

The largest of the three sections of the scouting system, Collection is devoted to entering and uploading all of the data that is used by the other systems. This is the most fundamental of the sections, since neither Processing or Visualation can function without it. In fact, many scouting systems do not pass the Collection stage. For example, solely paper-based scouting systems have no capability to process or visualize the data in anything except its raw form. Because of Collection's fundamental role, it must be the most stable of all of the sections. In our system, Collection is made up of three applications:

Scout
Android: Collects easily determined data concerning robot performance on the field. Example data points from this year are the number of high goals scored and number of defense crossings. Collects data in every qualifying match.

Super Scout
Android: Collects more subjective data on robot performance on the field, such as robot speed, agility, and ball control. Generally used by an experienced technical team member. Collects data in every qualifying match.

Pit Scout
iOS: Collects images and robot design data in the pits. Example data points from this year include robot photos, Generally used by a person with good social and photography skills.

Processing

Immediately following Collection is Processing. Processing is the smallest of the three sections, consisting of only one application: the server. It relies on Collection for the raw data that it condenses down to meaningful calculated metrics. Processing enables us to actually use the data that we collect in an effective and efficient manner. The calculations used this year are covered later in the Calculations section.

Server
Python: Performs the calculations that extract metrics of robot performance from the raw data. Also allows the export of the calculated data points to an `.csv` file used in the strategy meeting (discussed later).

Visualization

Visualization is the final section of the scouting system, and the part most commonly encountered. The goal of the Visualization section is to allow the strategy and drive teams to quickly and flexibly visualize both raw and calculated data. The greatest challenge in constructing this section is in allowing a wide variety in usages. For example, while the drive coach may be most interested in very fast identifications of the most powerful opposing robot, what each looks like, and if it will be necessary to play defense, a strategist in the stands may not care about the amount of time it takes to examine the data, but will want to be able to view and graph the raw data. These different use cases cause the Visualization section to be the most feauture-rich.

Android Viewer

Android: Allows for the quick and flexible viewing of all processed and raw data. Features include the ability to graph calculated and raw data points over a robot's matches, viewing of a robot's photos, and notifications for matches to watch.

iOS Viewer

iOS: Allows for the quick and flexible viewing of all processed and raw data. Features include the ability to graph calculated and raw data points over a robot's matches, viewing of a robot's photos, and notifications for matches to watch.

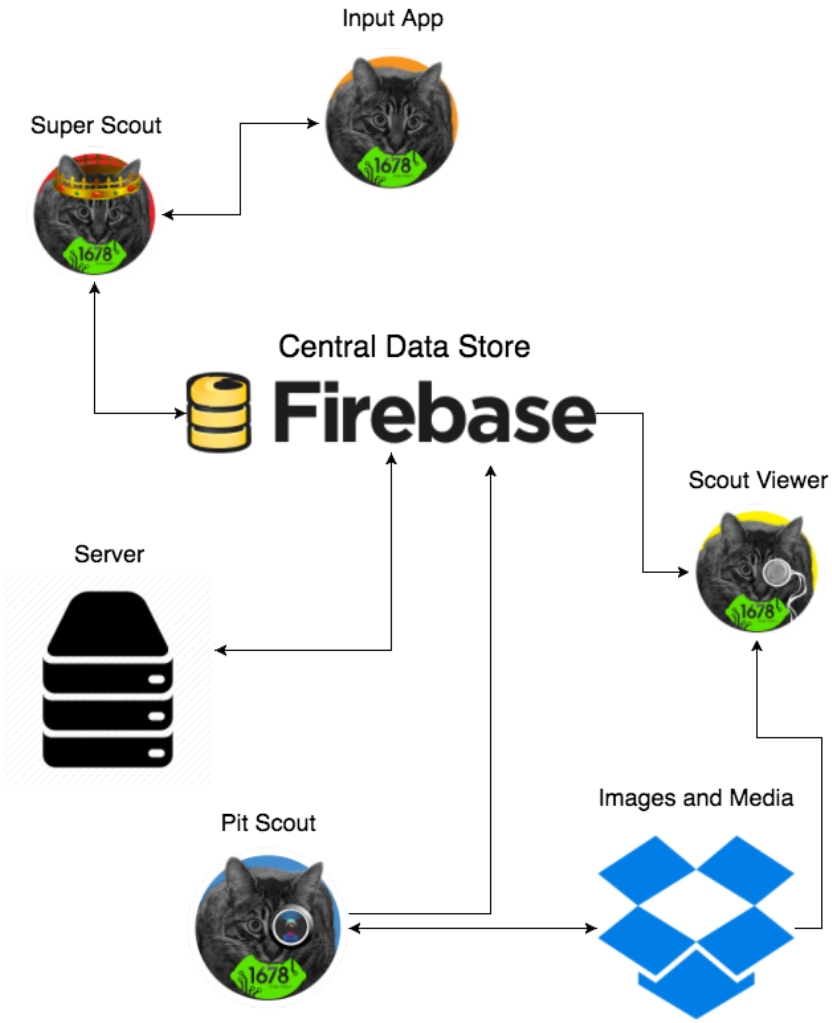


Figure 1.1: 2016 Scouting System Outline

Chapter 2

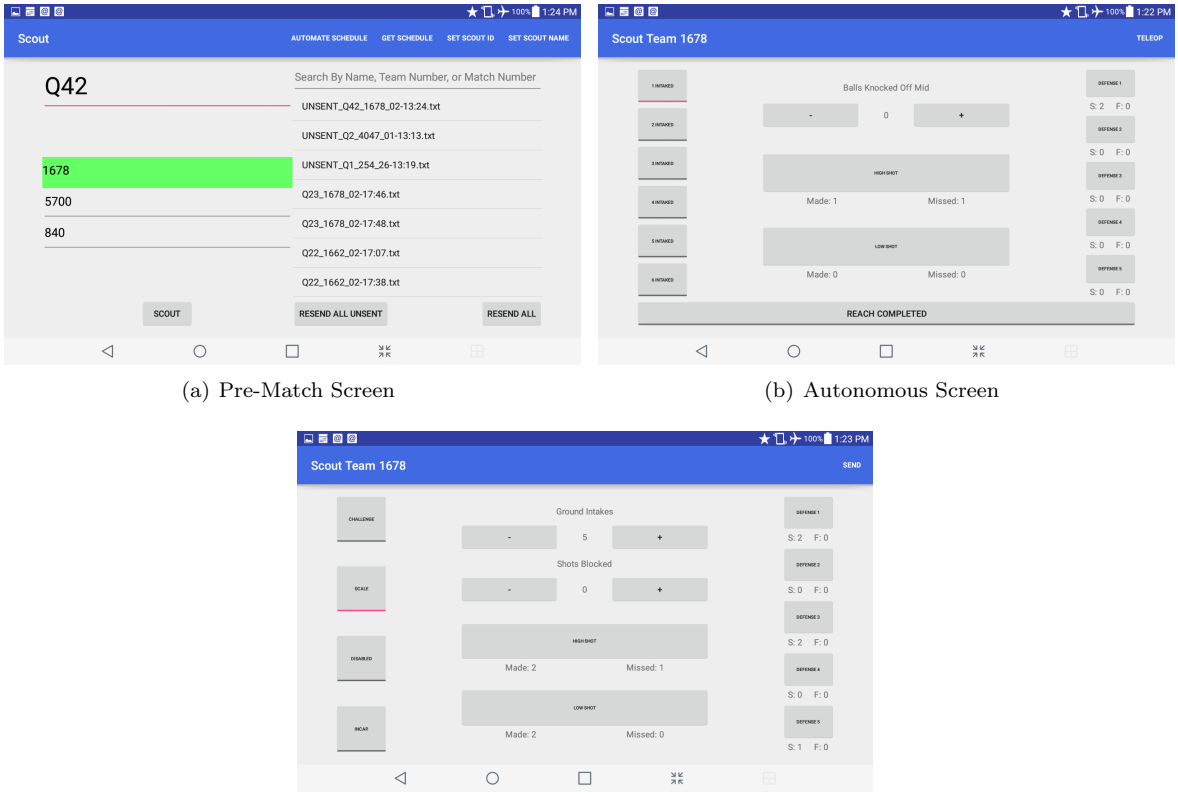
Technical Details

2.1 Collection

2.1.1 Scout

Feature List

- Simple and efficient interface for data entry
- Automated Bluetooth data transfer to Super Scout
- User initials entry to track scouts for performance reviews
- Rigorous backup system to prevent data loss



(a) Pre-Match Screen

(b) Autonomous Screen

(c) Teleoperated Screen

Figure 2.1: Scout User Interface

Details

The scout app is designed to be as consistent and simple as possible. Data is saved to disk before being sent via Bluetooth to the super scout, and a list of all of the matches and teams scouted on the right side of the prematch screen allows fast resubmission of data in the case of failed data transfer. This ability to quickly resubmit has been incredibly helpful due to the short time periods between matches. The scouting tablet has a set ID which allows it to

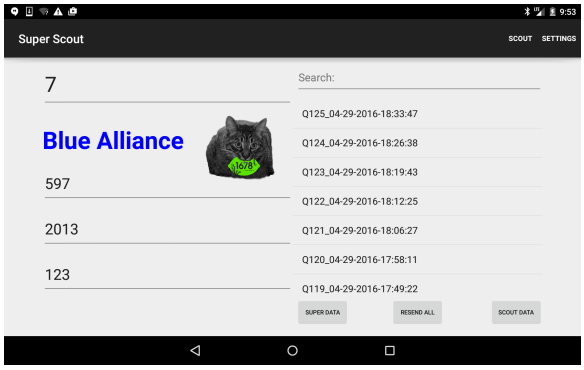
automatically select a team to scout from the schedule it retrieves from the super scout. In the past, manual entry of this information has been highly inconsistent due to typos. We have found that an automated system greatly increases data accuracy. In the case of a change in schedule, a manual override option is available, but it is very easy to return to the automated mode once the competition resumes the schedule.

Upon starting the match, a dialog appears which requests the scout's initials for tracking purposes. The user is required to reenter their initials at the beginning of every match to ensure that the initials are always correct despite scouts switching out. Data entry occurs through a simple interface consisting solely of buttons and counters. We have found that in the high pressure environment of scouting live matches, a simple and static interface of buttons provides greater data accuracy than a more complex and dynamic UI. In order to aid in gathering accurate timing data and to ensure that all actions are completed, many buttons such as defense crossings and high shots create a full-screen dialog with large color coded buttons of success and failure. Despite this effective interface, the collection of timing data is still minimized due to inaccuracy. Upon the final step of data collection, all data is automatically transferred to the paired super scout. All data is initially marked as unsent. When the scout detects a successful transfer, the data is marked as sent. A button on the prematch screen is provides the option to resend all unsent data.

2.1.2 Super Scout

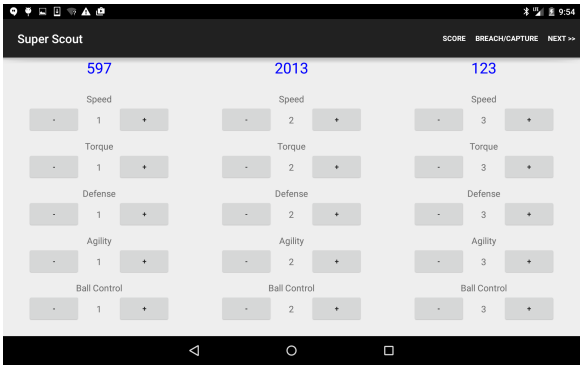
Feature List

- Simple and efficient interface for data entry
- Defense arrangement selection
- Note-taking capability for additional observations
- Automated Scout data collection through Bluetooth
- Background data processing and uploading
- Rigorous backup system to prevent data loss

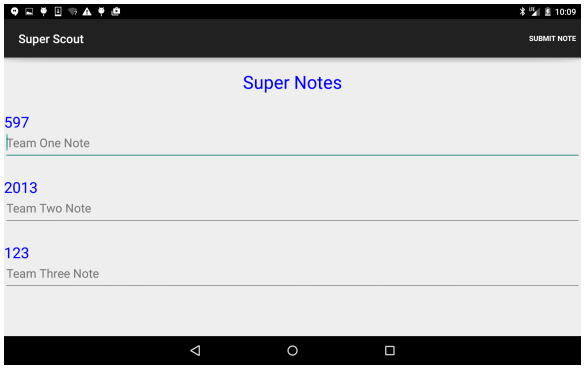


(a) Pre-Match Screen

(b) Defense Input Screen



(c) Data Input Screen



(d) Notes Input Screen

Figure 2.2: Super Scout User Interface

Details

The super scout app is responsible for transferring all of the data to Firebase. The scouts send all of their data via Bluetooth to the super scout, which modifies, saves, and uploads the data. Modification is needed to insert the correct defenses into the data. Because only the super scouts enter the defense positions on the field, the scouts only enter the position of the defense crossed when recording data. When the scout data arrives at the super scout, the super scout automatically converts all of the positions to the correct defenses. The scout data is saved on the super scout tablet for another level of redundancy to avoid data loss. Similar to the scout app, the super scout has a list of all of the saved files and can quickly reupload them.

Like the scout, match progression is done automatically using the schedule. The super scout can also override this automated progression in case a match needs to be replayed or any other deviation from the schedule occurs. Each super scout enters data on one alliance, not one robot like the scout.

Upon starting the match, the super scout enters the positions of the defenses that their alliance must face. To ensure that all of the defenses are correct, the user is only allowed to enter a possible arrangement of defenses. Similar

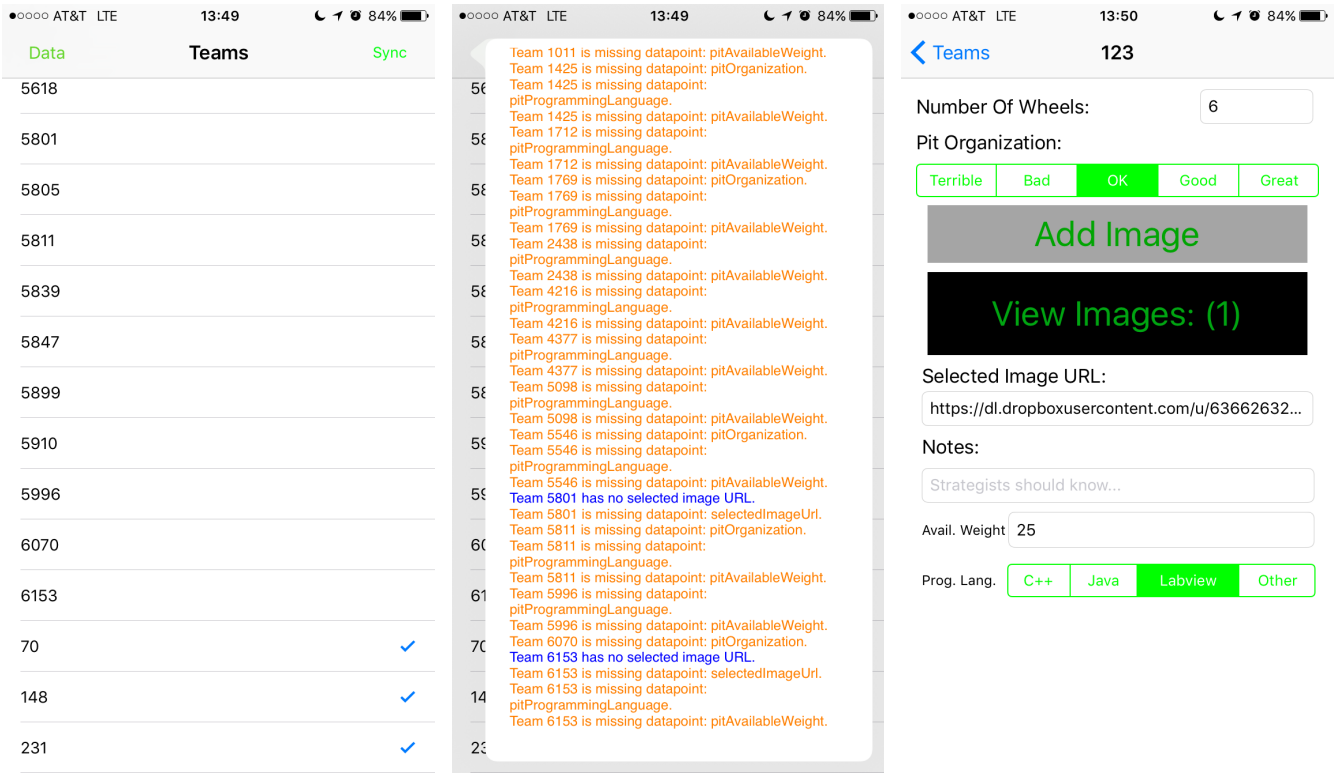
to the scout app, the interface is made up almost entirely of counters so that data entry is made as simple as possible. Evaluation is done on a scale of 1 (Worst) to 3 (Best) on their alliance. The best robot in that category on the field receives a 4 instead of a 3. If the super scout feels it is necessary to note something about a robot that does not fit into any of the categories they are scoring, they can tap on the team number to open a screen where they can enter notes on that team.

When the super scout has finished scouting the match, the app transitions to a screen where the super scout must input their alliance's score and whether or not the alliance captured or breached. After this data is entered, all of the data is automatically written to Firebase and the app returns to the Pre Match Screen.

2.1.3 Pit Scout

Feature List

- Short list of data points for rapid collection
- Local data storage when offline
- Missing data display
- Ability to select default team image



(a) Team List Screen (b) Missing Data Display (c) Data Entry Screen

Figure 2.3: Pit Scout User Interface

Details

Unlike the scout and super scout apps, the pit scout app is designed to collect very few data points. Because of this, data entry is significantly simplified. There is a single data entry screen, where the user can enter the number of wheels, the pit organization, notes, the robot's available weight, and the programming language. The user can also add pictures, view all the entered pictures, and set a selected image to be viewed in the Visualization apps. All data other than pictures is automatically uploaded when entered, and is marked in red if invalid. The number of data points is minimal because all unnecessary data points are removed from the application. In order to minimize the inconvenience of answering the questions, we made it a high priority to only collect data that is absolutely necessary.

In order to increase how fast pit scouting is done, the app is able to function on multiple devices simultaneously so that more than one team member can pit scout at a time. This, combined with the low number of data points, has enabled us to complete pit scouting all the teams in a competition in approximately an hour.

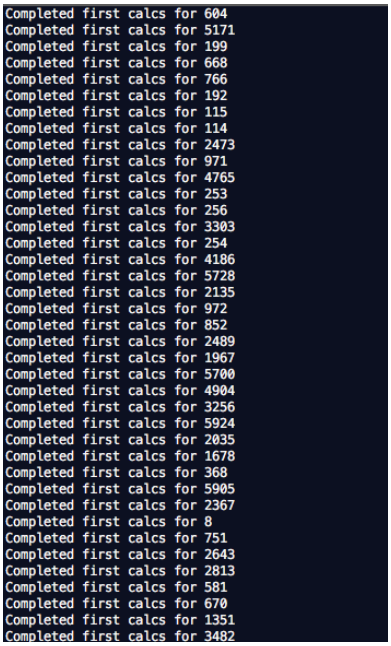
Navigation between teams is done through the Team List Screen. In order to make it easier to remember which teams still need to be scouted, a long press on a team adds a check to their name and moves them to the bottom of the list. Missing data can also be checked through the "Data" button, which displays a dialog of the data that is missing from teams.

2.2 Processing

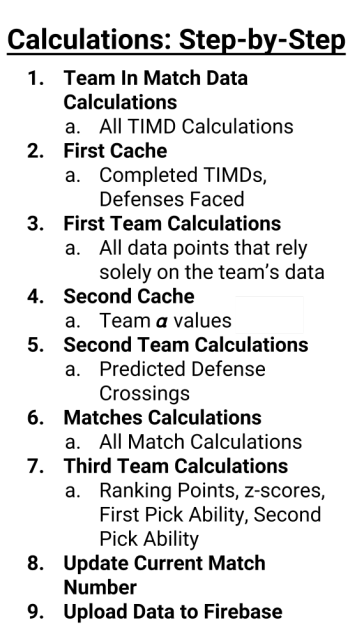
2.2.1 Server

Feature List

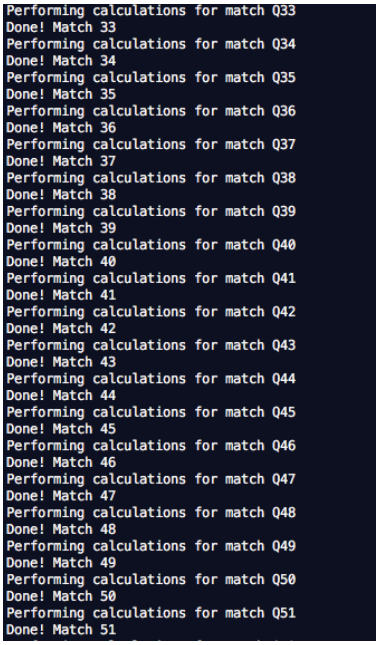
- Parallel processing of data for accelerated calculations
- Caching to improve processing speed
- Export of calculated data to `.csv` file for strategy meetings
- Automated email crash reporting to maximize uptime during competition
- Asynchronous data writing for accelerated uploading
- Scout performance review algorithm to maximize data accuracy



(a) Team Calculations



(b) Calculation Process



(c) Match Calculations

Figure 2.4: Server Calculations

Details

As the only application in Processing, the server processes all of the collected data. The application is run on a relatively low-powered server, and so it is necessary to take advantage of all of the processing power to increase the speed of the calculations. Near the end of competition, the system contains over ten thousand individual pieces of data. In order to ensure that the calculations are never more than a match or two behind, the server relies heavily on parallelism. The majority of the computation is done on the TIMDs due to their large number, so each of the TIMDs is assigned its own process. Processing speed is also increased through caching. Data points such as α values for teams, despite not being saved to Firebase, are preserved locally so they only need to be calculated once. Finally, data transfer to Firebase also parallelized to increase the speed of the calculation cycles.

While Firebase is not a relational database, the server attempts to simulate the functionality of one by linking all of the objects together upon downloading them. This allows the system to quickly transfer between Team, Match, and TIMD objects without relying on constantly filtering large lists of all of the objects.

Other capabilities of the server are the ability to run scout evaluations, export the calculated data to a `.csv` file for easy viewing, and automatically report crashes via email. In order to maximize the accuracy of our data, we evaluate the accuracy of the data generated by each scout so that scouts who are entering unreliable data can be identified and

either trained more or removed from their role as scout. The evaluations function by running a linear regression on the error between the score that would be generated by the collected data points and the actual score. Similar to OPR, this allows us to identify the calculated contribution of each scout to the total error. We have found this measurement to be fairly reliable and correlate strongly with the amount of training a scout has received.

The server is also able to export a `.csv` file of the calculated data for viewing in Excel during a strategy meeting. During the strategy meeting, this file is projected on a wall so that all of the participants can view the data. We have found this approach helps to increase the speed of resorting the teams and allow more flexibility in analysis than a custom-built method of viewing the data. In addition, the file is easily transferrable and viewable by many users due to the ubiquitous nature of Excel.

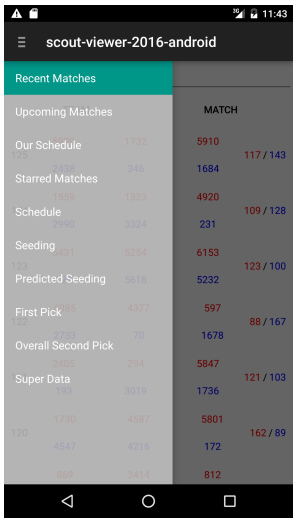
Finally, the server reports all crashes through email. Due to an unreliable network connection and various other issues, the server will occasionally crash and stop processing data. In order to maximize uptime, the server automatically sends an email to all of the app programming team when this occurs so that they can restart the server without any delay. In addition, this email includes the stack trace to aid in debugging the issue.

2.3 Visualization

2.3.1 Android Viewer

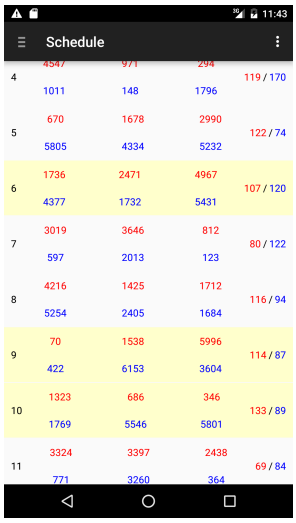
Feature List

- Background image downloading for immediate loading at competition
- Graphing data across matches for visualization of team performance trends
- Ranking of teams by any data point for flexible data analysis
- Access to individual TIMD raw data points
- Starring teams and matches to prevent missed matches-to-watch
- Match summary screen for quick evaluation of matches
- Quick access to team schedule

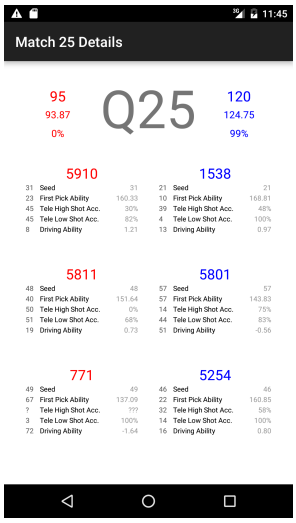


(a) Navigation Drawer

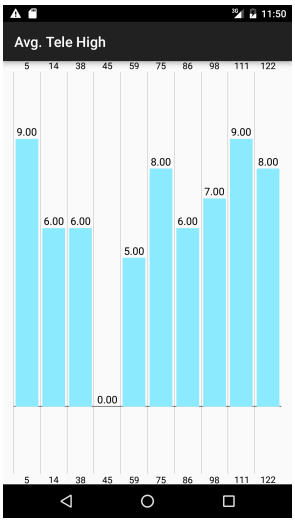
(b) Driver Ability Rankings



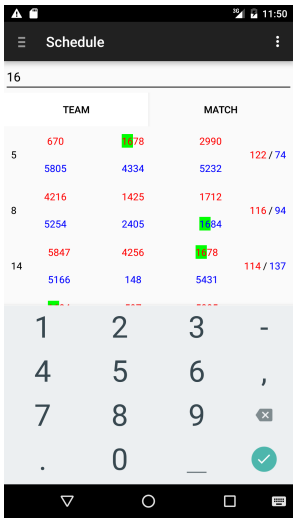
(c) Match Schedule



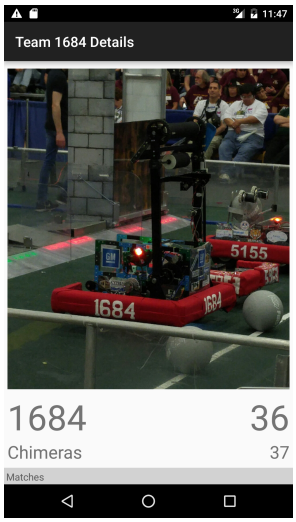
(d) Match Summary Screen



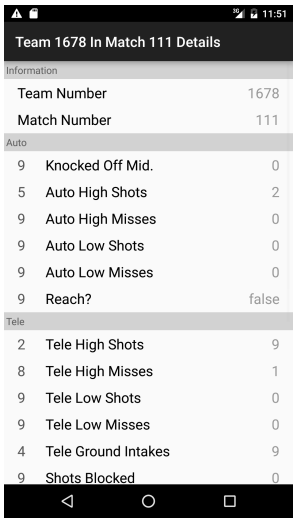
(e) Graphing Capabilities



(f) Searchable Lists



(g) Team Data Screen



(h) Raw TIMD Screen

Figure 2.5: Android Viewer User Interface

Details

As part of Visualization, the Android Viewer is one of the most feature-rich parts of the system. Built to meet the needs of all members of the drive and strategy teams, the Android Viewer is designed to be very flexible in its use. The core of the application is the navigation drawer, which allows the user to easily access many common data views, such as the team's schedule, the first and second pick lists, the current seeding, and all starred matches. Navigation through the app is intended to also be very flexible, allowing the user to follow their train of thought. For example, tapping a match on the schedule brings the user to that match's summary screen. Tapping a team on that screen brings the user to the data screen for that team. The user can then easily look at that team's matches, from which they can view the summary screen of a match, from which they can view another team. To help with the long chains of activities this creates, long pressing the back button returns the user to the application's root activity.

Due to the poor cell data connection at competition, image loading can often take a significant amount of time. The user's time constraints in competition make waiting for the image to load impractical. To address this, a service runs in the background whenever the phone is turned on. This service automatically downloads the selected image for each team and saves it to disk. If the selected image changes, the service deletes the old image and downloads the new one. While this approach requires a significant amount of cell data and hard disk usage, it allows for nearly instantaneous image loading at competition.

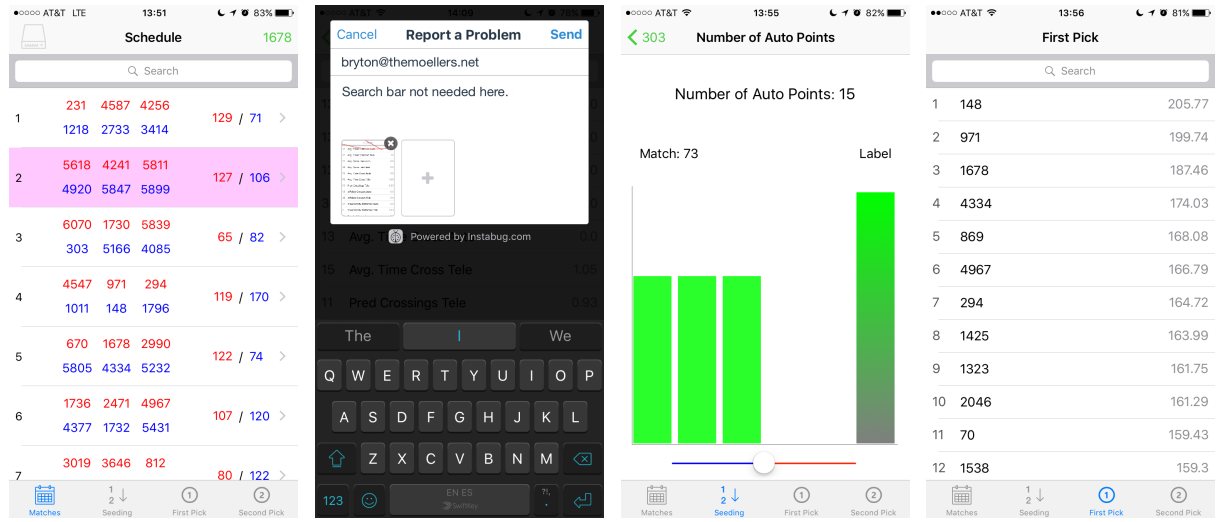
Another service runs constantly in the background in order to ensure that the user does not miss any matches they need to watch. The user can "star" matches and teams inside the app through a long press. The app will then notify them when the match is 3, 2, 1, and 0 matches away. This helps to overcome the chaos of competition and ensure that important matches with future alliance partners are not missed.

Three main features ensure that data analysis is as flexible as possible: graphing, raw data viewing, and rankings. The user is able to graph almost every data point over matches by tapping on the data point. For example, to identify any trends in the accuracy of a team's shooter, the user can tap on the "High Shot Accuracy" for a team. This will create a bar graph of the team's high shot accuracy in each of their matches. In order to examine what happened in each individual match, the user can tap on the bar for that match to bring up all of the raw collected data and notes from that match. In order to compare data points between teams, the user can long press a data point to rank all of the teams by that data point. For example, to identify the most accurate robots in the competition, the user can long press the "High Shot Accuracy" for a team, calling up a list of all of the teams ranked by their accuracy. The combination of these three features enable data to be easily analyzed on a variety of scales, from an individual TIMD to the whole competition.

2.3.2 iOS Viewer

Feature List

- Optional background image downloading for immediate loading at competition
- Caching of image data for lower network usage
- Graphing data across matches for visualization of team performance trends
- Ranking teams by any data point for flexible data analysis
- Starring teams and matches to prevent missed matches-to-watch
- Match summary screen for quick evaluation of matches
- Quick access to team schedule



(a) Match Schedule (b) Instabug Bug Reporting (c) Graphing Capabilities (d) First Pick Rankings

(e) Team Data Screen I (f) Team Data Screen II (g) Searchable Lists (h) Reversible Rankings

Figure 2.6: iOS Viewer User Interface

Details

Due to the similarities of the Android and iOS Viewers, the differences between the two will be noted. Unless stated, the iOS Viewer shares all of the features of the Android Viewer.

Instead of a navigation drawer, the iOS Viewer handles root view navigation through a navigation bar at the bottom of the screen. This is done due to the iOS application design's lesser reliance on navigation drawers.

The most significant difference between the iOS Viewer and the Android Viewer is in team images. The iOS Viewer has the option to turn off automatic image downloading and instead download images when the user requests them. While this conserves cell data and hard drive space, it does require the user to wait for the image to be downloaded. The iOS Viewer also allows the user to examine the non-selected images for a team, while the Android Viewer only allows the user to view the selected image.

The final difference between the two is in graphing and raw data viewing. The iOS Viewer does not show match number and data value on the graphs unless the bar is tapped, and does not allow viewing of raw TIMD data.

2.4 Logistics

2.4.1 Development Process and Schedule

Due to the lack of time during the six week build season for design and development of an effective scouting system, we attempt to build as much as possible during the offseason and continue to iterate throughout competition season. The development schedule can be divided into roughly six periods:

Offseason: June 1 January 8

Kickoff: January 9 January 11

Early Build Season: January 12 January 23

Late Build Season: January 24 February 23

Competition Season: February 24 April 9

Championships: April 10 April 30

2.4.2 Offseason

"Winners are made in the offseason"

During the offseason, we design and construct as much of the infrastructure as possible. Starting immediately after Championships, we begin meeting to evaluate the system from the previous year. Throughout the summer, we test out various different infrastructures until we have decided on the libraries and technologies we will use for the system. This decision is generally made near the time that the team resumes regular meetings in mid August.

From then until Kickoff, our focus is on training new members and constructing as much of the system as we can without knowing the game. In order to dedicate as much time to developing the new system as possible, we often do not run our scouting system at offseason competitions. If we do decide to use the system, it is run by newer members as part of their training.

By the end of the offseason, we have successfully implemented and tested the core elements of the system. For example, by Kickoff this year, we had designed and tested the Bluetooth data transfer protocol, Firebase uploading and downloading, and Dropbox photo uploading. During build season, all three of these technologies were very quickly implemented due to the work done during the offseason.

2.4.3 Kickoff

After the Game Reveal, we spend the rest of the day and the following two days brainstorming. After the Rules Test, the first question asked is known on our team as the "What" question. Essentially, this is where the strategy for the season is defined. Everyone is strictly banned from discussing designs until the entire team has settled upon this strategy. This period is important for scouting because scouting must integrate with strategy. The strategy that we pursue on the field dictates what we scout for. For example, our decision to often send our second pick to play defense made it necessary to track the driving ability of each team.

2.4.4 Early Build Season

Following Kickoff, the scouting team spends the majority of the following two weeks designing the scouting system. In these meetings, we solidify the design of each of the applications. The first step in this is to decide on the exact list of data points that we will track. After this list has been created, the scouting team moves through each of the apps and settles on a design for each app. For example, the need to track defense crossings led to the creation of the defense selection screen in the super scout. In order to maximize productivity, as each member's application is completed,

he or she drops out of the design process. The order in which applications are designed is usually chosen in order to ensure that more experienced members are involved until the end, while newer members learn the process but are also given maximum time to complete their programming.

The final step of this design process is the calculations. A small group of members of the scouting team (this year, one student and two mentors) work together to transform the collected data points into meaningful metrics of robot performance. While the bulk of these calculations are designed in the first two weeks, work goes on in a significant form throughout the rest of the build season as ways to circumvent previous problems in the calculations are discovered.

2.4.5 Late Build Season

Late build season is mainly dedicated to constructing the system. Almost all of meeting time is dedicated to developing the variety of apps necessary for the system to function.

One crucial feature of this period is the practice competitions. During a practice competition, we simulate a competition in order to test part or all of the system. These evaluations are incredibly useful in pinpointing issues of integration between different applications. At the end of build season, in order to ensure that all of the applications are completed, a single large practice competition is held in which the entire system functions as it will at competition.

The other significant event that takes place during this time is scout training. By this time, the leadership team has decided on the members of the travel team who will be involved in scouting. Once the scouting and super scouting apps are completed, the members of the scouting team begin to train the scouts in using them. Beginning training this early is advantageous in that it allows iteration of the app designs as we receive feedback from the scouts. This helps to ensure that the final design is the most usable for the scouts.

2.4.6 Competition Season

By the end of build season, all of the components of the scouting system have been completed and tested thoroughly in the practice competitions. The early part of competition season before our first competition is focused on scout training. Multiple full-day training sessions are held using the livestreams from early competitions. We select a livestream that accurately mirrors the view from the stands and seat the scouts in the formation that they will be in at competition. The scouts then proceed to all scout a single robot in each match and compare their data after the match. In order to complete their training for the day, the scouts must achieve a certain number of identical data sets in a row (typically this number is set at fifteen).

During this time, any apps that may have fallen behind schedule are also finished. With less work being done on the system, many of the more experienced members are moved over to assist any projects that have not met their deadline.

The time from after the first competition to our last is devoted to iteration. At each competition, we keep track of all of the user's opinions through both oral feedback and Instabug. This bug reporting software has been absolutely essential in the success of our scouting system this year. We cannot recommend this tool highly enough for gathering crash information and user opinions at competition where the users and developers are often very spread out and unable to communicate.

After each competition, we hold a debrief in which we decide on a set of tasks that must be accomplished by the next competition. An important part of this decision is not hesitating to change even a large part of the app if it is not acting as a useful part of the scouting system. For example, in 2014, much of the scouting system was centered around tracking the location of all the robots on the field. However, after the sacrifice in the accuracy of other data points was seen to be too great, the entire user interface was redesigned to no longer include location tracking before the following competition.

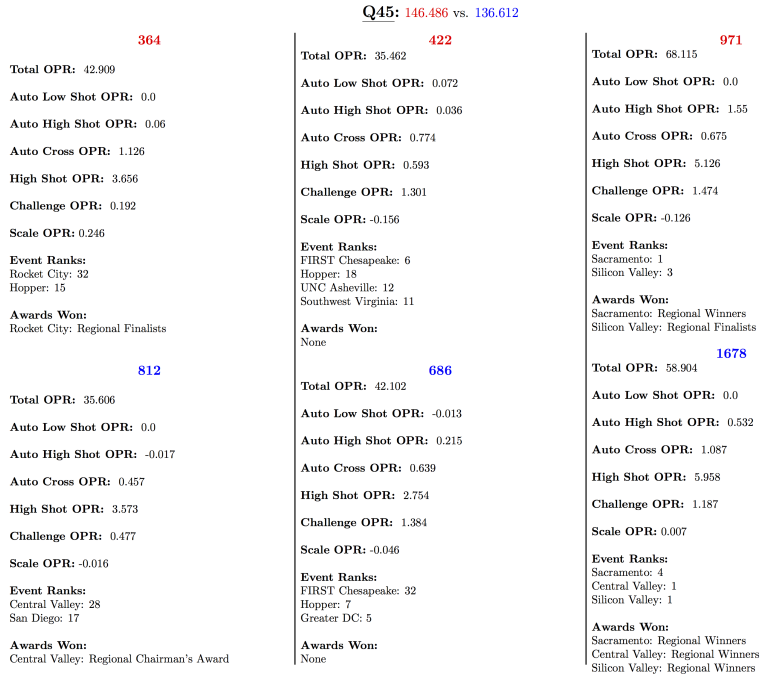
In addition, we ensure that we maintain access to all the data collected at previous competitions. Because we play against many teams at multiple competitions, having the data we collected at past competitions allows us to enter the competition with large amounts of data on multiple teams already. As the competition progresses, we slowly move from the use of the old data to the new.

2.4.7 Championships

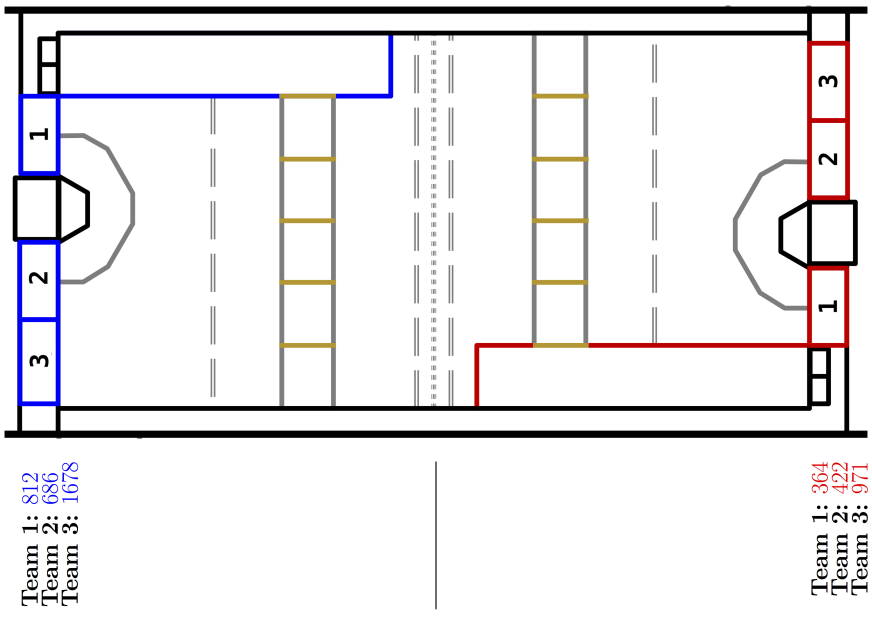
The time from after the final competition until we leave for the Championships is the final round of scouting system development. During these couple weeks, we work on implementing features that will be important at Championships. For example, the scouting team decides on the necessity of a separate ranking for a third pick robot and implements it if it is needed.

Championships also poses a significant challenge because of the lack of exposure that we have had to most of the teams in our division. In order to overcome this, we use a combination of data collected from The Blue Alliance

and recorded match videos from other competitions. A script put together before Championships generates strategy sheets for each match and displays OPR data on all of the teams in each of our matches. Another script automatically downloads the video recordings entered into The Blue Alliance and saves them to an external drive so that we are able to watch match video on the plane flight to St. Louis. This year, we downloaded over 100 GB of video. On the flight, an experienced member of the scouting team, the strategist, and the drive coach all sit together and work on designing the strategies for each of the matches. By doing this, we do not waste any time and most of the matches are planned out by the time we land.



(a) Front



(b) Back

Figure 2.7: Strategy Sheet Example

Chapter 3

Calculations

3.1 Calculations

3.1.1 Low Level

Average (Standard)

Data Points: twoBallAutoAttemptedPercentage, avgBallsKnockedOffMidlineAuto, avgMidlineBallsIntakedAuto, teleopShotAbility, avgHighShotsMadeTele, avgHighShotsAttemptedTele, avgLowShotsMadeTele, avgLowShotsAttemptedTele, highShotAccuracyAuto, lowShotAccuracyAuto, autoAbility, autoAbilityExcludeD, autoAbilityExcludeB, avgHighShotsMadeAuto, avgLowShotsMadeAuto, avgHighShotsMadeTele, avgHighShotsAttemptedTele, avgLowShotsMadeTele, avgLowShotsAttemptedTele, highShotsAccuracyTele, lowShotAccuracyTele, reachPercentage, avgGroundIntakesTele, avgShotsBlocked, siegeAbility, siegeConsistency, challengePercentage, scalePercentage, avgTorque, avgSpeed, avgAgility, avgDefense, avgBallControl, avgDrivingAbility, disfunctionalPercentage, disabledPercentage, incapacitatedPercentage, breachPercentage

The average of a data point across across a team's played TIMDs. Boolean values of True and False are treated as 1 and 0 respectively.

Standard Deviation (Standard)

Data Points: sdHighShotsAuto, sdHighShotsTele, sdLowShotsAuto, sdLowShotsTele, sdSiegeAbility, sdGroundIntakes, sdTeleopShotAbility, sdAutoAbility, sdShotsBlocked, sdMidlineBallsIntakedAuto, sdBallsKnockedOffMidlineAuto

The standard deviations of a data point across a team's played TIMDs. Boolean values of True and False are treated as 1 and 0 respectively.

Average (Defense)

Data Points: beachedByDefensePercentage, slowedByDefensePercentage, unaffectedByDefensePercentage, avgNumTimesBeachedByDefense, avgNumTimesSlowedByDefense, avgNumTimesUnaffectedByDefense, avgSuccessfulTimesDefensesCrossedAuto, avgSuccessfulTimesDefensesCrossedTele, avgFailedTimesDefensesCrossedAuto, avgFailedTimesDefensesCrossedTele, avgTimeForDefenseCrossAuto, avgTimeForDefenseCrossTele, crossingsSuccessRateForDefenseAuto, crossingsSuccessRateForDefenseTele

The average of a value across all the played TIMDs in which the team faced a defense d. Boolean values of True and False are treated as 1 and 0 respectively.

Standard Deviation (Defense)

Data Points: sdSuccessfulDefenseCrossesAuto, sdSuccessfulDefenseCrossesTele, sdFailedDefenseCrossesAuto, sdFailedDefenseCrossesTele

The standard deviations of a data point across all the played TIMDs in which the team faced a defense d. Boolean values of True and False are treated as 1 and 0 respectively.

3.1.2 High Level

timd.autoAbility

timd.autoAbility is the contribution of points that team T makes in TIMD τ during the autonomous period. This value is defined as

$$t.aA(\tau) = 10 * t.hSA(\tau) + 5 * t.lSA(\tau) + 2 * t.dRA(\tau) + t.CDA(\tau)$$

where $t.hSA(\tau)$ is the `timd.numHighShotsMadeAuto`, $t.lSA(\tau)$ is the `timd.numLowShotsMadeAuto`, $t.dRA(\tau)$ is the `timd.didReachAuto`, and $t.CDA(\tau)$ is 5 if the sum of `timd.timesSuccessfulCrossedDefensesAuto` is greater than 0 and 0 if it is not.

timd.teleopShotAbility

`timd.teleopShotAbility` is the contribution of points that team T makes in TIMD τ during the teleop period through high and low shots. This value is defined as

$$t.tSA(\tau) = 5 * t.hST(\tau) + 2 * t.lST(\tau)$$

where $t.hST(\tau)$ is the `timd.numHighShotsMadeTele` and $t.lST(\tau)$ is the `timd.numLowShotsMadeTele`.

timd.siegeAbility

`timd.teleopShotAbility` is the contribution of points that team T makes in TIMD τ during the teleop period through challenging and scaling. This value is defined as

$$t.sA(\tau) = 5 * t.dCT(\tau) + 15 * t.dST(\tau)$$

where $t.dCT(\tau)$ is the `timd.didChallengeTele` and $t.dST(\tau)$ is the `timd.didScaleTele`.

Z-Score

Data Points: `zScoreTorque`, `zScoreSpeed`, `zScoreAgility`, `zScoreDefense`, `zScoreBallControl`, `zScoreDrivingAbility`

As can be seen from the graph, the ranking system used by the scouts creates a distribution of scores with many teams ranked near the middle.

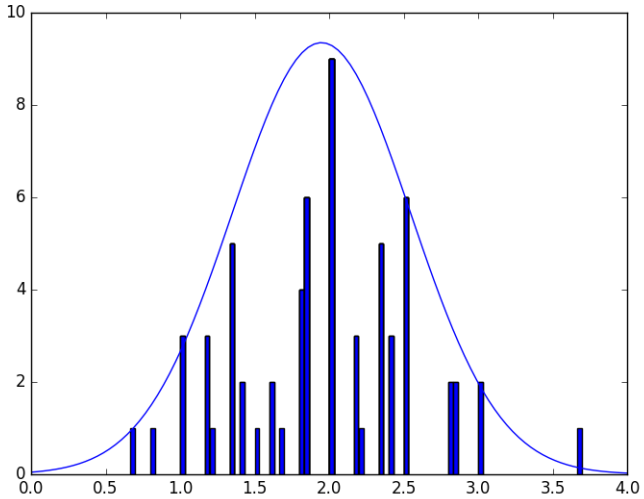


Figure 3.1: Super Ranking Average Distribution

In order to properly spread out the teams, a z-score is used. The z-score of a super ranking average is

$$zS(T) = \frac{\mu(T) - \mu_{comp}}{\sigma_{comp}}$$

where $\mu(T)$ is the super ranking average for team T, $\mu_{comp} = \frac{\sum_{T \in C} \mu(T)}{|C|}$ where C is the set of all teams in the competition, and σ_{comp} is the standard deviation of the $\mu(T)$'s across C.

firstPickAbility

Strategically, the goal of our first pick is to act as a solely offensive robot and generate the maximum number of points in combination with our team. Therefore, the strength of a robot as a first pick can be determined solely by predicting the score they would generate on an alliance with our team in elimination matches. This score is the sum of the predicted score and the points generated by the captures and breach chances. Summing these three contributions for an alliance A of our team and team X yields the equation

$$fpa(X) = S_p(A) + 25 * cC(A) + 20 * bC(A)$$

where $S_p(A)$ is the predicted score of the alliance A, $cC(A)$ is the chance that alliance A will earn a capture, and $bC(A)$ is the chance that alliance A will earn a breach.

predictedScore

$$S_p(A) = aDPT(A) + \sum_{T \in A} aA(T) + \sum_{T \in A} tSA(T) + \sum_{T \in A} sA(T)$$

where A is a set of all teams on the alliance, $aA(T)$ is the `autoAbility` of team T, $tSA(T)$ is the `teleopShotAbility` of team T, and $sA(T)$ is the `siegeAbility` of team T. $aDPT(A)$ is the predicted number of points alliance A receives from defense crossings. This function is defined as

$$aDPT(A) = \sum_{dc \in \{a,b,c,d,lb\}} 5 * \min\{pca(A, dc), 2\}$$

where dc is a defense category and $pca(A, dc)$, the predicted number of times alliance A crosses the defense of category dc, is

$$pca(dc) = \sum_{T \in A} pdc(T, dc)$$

where $pdc(T, dc)$ is the set of the predicted number of times that team T will cross the defense in defense category dc. $pdc(T, dc)$ is simply the average of the predicted number of crossings of the defenses in the category dc.

$$pdc(T, dc) = \sum_{d \in dc} pc(T, d)$$

The predicted number of crossings for defense d is more difficult to define. Intuitively, this would simply be the average of the number of times that team T has crossed d in previous matches. However, at low numbers of observations, such as when a team has only faced a defense once or twice, there will be a significant error in this assumption. For example, if T plays with teams that are great at crossing the portcullis in the only matches where T faces the portcullis, T may average zero crosses of the portcullis even if T is capable of crossing the portcullis.

In order to account for this, we assume a correlation between a team's ability to cross d and their ability to cross the other defenses. If T crosses the other eight defenses more times than any other robot in the competition, it is likely that it will be good at crossing the ninth defense, even if it has not shown the ability. The prediction $N(T, d)$ resulting from this assumption is therefore

$$N(T, d) = \bar{n}c(d) * \theta(T)$$

where $\bar{n}C(d)$ is the average number of d crossings for all of the robots in the competition and $\theta(T)$ is a measure of how much better at crossing T is better than the other robots in the competition. To find $\theta(T)$, we simply compare T's ability to cross defenses with the other robots in the competition. $\theta(T)$ becomes the average of this difference in ability $\alpha(T, d)$ for each of the defenses. However, due to the low number of observations at this point in the competition, we also include a measure $\beta(T, d)$ of how much to weight the $\alpha(T, d)$ value for each defense. The more observations we have for the team and the competition, the more we trust the $\alpha(T, d)$ and the more we weight it:

$$\theta(T) = \frac{\sum_{d \in ds} \alpha(T, d) * \beta(T, d)}{5}$$

where ds is the set of the nine defenses. The division by 5 is necessary to normalize the value, since there are a total of five defenses on the field in every TIMD. In order to find $\alpha(T, d)$, we simply compare how good T is at crossing d in relation to the rest of the competition:

$$\alpha(T,d) = \frac{nc(T,d)}{\bar{n}c(d)}$$

$\beta(T,d)$ needs to increase with the more observations that we have of T in a match with d, and the more times d has been seen in the competition, since $\alpha(T,d)$ relies on data points collected from those two types of observations. Therefore, we define $\beta(T,d)$ as:

$$\beta(T,d) = \frac{C_{prop}(d) + T_{prop}(T,d)}{2}$$

where $C_{prop}(d)$ is the proportion of TIMDs in the competition where defense d was faced and $T_{prop}(T,d)$ is the proportion of team T's TIMDs where defense d was faced.

secondPickAbility

The second pick ability of a team T is defined as

$$spa(T) = [1 - dfp(T)] * [aA(T) + sA(T) + \gamma * zSS(T) + \delta * zSA(T) - \epsilon * mdct(T)]$$

where $dfp(T)$ is the disfunctionalPercentage of team T, $aA(T)$ is the autoAbility of team T, $sA(T)$ is the siegeAbility of team T, $zSS(T)$ is the zScoreSpeed of team T, $zSA(T)$ is the zScoreAgility of team T, and $mdct(T)$ is the meanDefenseCrossTime of team T. γ, δ, and ϵ were determined by a linear regression using the second pick list from our first competition. For our season, we have found the values $\gamma = 2.4$, $\delta = 1.2$, and $\epsilon = 2.4$ to be very accurate.
The reasoning behind this calculation is more complicated than that of firstPickAbility. Again, however, it is grounded in the strategy we intended to execute on the field. As can be seen in all three of our regionals, the role of our second pick was to score points in the autonomous period, to ensure the capture at the end of the match, and to play effective defense on the opposing alliance for the majority of the game. In order to address all three of these actions, the calculation is constructed of three parts.

1. $(1 - dfp(T)) * aA(T)$: Prediction of the number of points the team would generate in the autonomous period.
2. $(1 - dfp(T)) * sA(T)$: Prediction of the number of points the team would generate in sieging (either challenging or scaling) the tower. Not including the capture points balanced out the fact that many teams highly skilled at challenging did not do so in the qualification matches due to the importance of achieving a breach.
3. $(1 - dfp(T)) * (\alpha * zSS(T) + \beta * zSA(T) - \gamma * mdct(T))$: Accurate determination of the skill of team's drivers in accomplishing the tasks necessary for a second pick.

The main challenge in assembling this calculation was the integration of both subjective data collected by the super scout and the objective data collected by the scout. In order to find this balance which could not be easily derived, linearly regressed constants were used.

sdPredictedScore

$$S_{p_{sd}}(A) = \sqrt{aDPT_{sd}(A)^2 + \sum_{T \in A} aA_{sd}(T)^2 + \sum_{T \in A} tSA_{sd}(T)^2 + \sum_{T \in A} cA_{sd}(T)^2}$$

The calculation merely sums the standard deviations of the various components of $S_p(A)$. Due to the difficulty of determining the standard deviation throughout the calculation of $aDPT(A)$, $aDPT_{sd}(A)$ is determined using a Monte Carlo method[1].

winChance

In order to determine the win chance of alliance A facing opposing alliance O, Welch's t-test[2] is used. This test is expressed in the formula

$$t = \frac{\bar{X}_1 + \bar{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}$$

[1] https://en.wikipedia.org/wiki/Monte_Carlo_method
[2] https://en.wikipedia.org/wiki/Welch%27s_t-test

where $\bar{X}_1$ is the mean of the first sample, s_1 is the standard deviation of the first sample, N_1 is the size of the first sample, $\bar{X}_2$ is the mean of the second sample, s_2 is the standard deviation of the second sample, and N_2 is the size of the second sample. This t is then converted to a win probability $wC(A)$ using the cumulative distribution function[3] for a t-distribution $T(t|\nu)$.

In this case, $\bar{X}_1$ is $S_p(A)$ for alliance A, s_1 is $S_{p\_\sigma}(A)$ for alliance A, and N_1 is the average number of completed TIMDs $\bar{\nu}CT(A)$ of the teams on alliance A. Similarly, $\bar{X}_2 = S_p(O)$, $s_2 = S_{p\_\sigma}(A)$, and $N_2 = \bar{\nu}CT(O)$. The function $wC(A)$ comes out to

$$wC(A, O) = T(t|\nu)$$

t is the t-value generated by the Welch's test and ν is the degrees of freedom approximated by the Welch-Satterthwaite equation[4].

$$\nu \approx \frac{(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2})^2}{\frac{s_1^4}{N_1^2 * \nu_1} + \frac{s_2^4}{N_2^2 * \nu_2}}$$

where $\nu_1 = N_1 - 1$ (the degrees of freedom for the first variance) and $\nu_2 = N_2 - 1$ (the degrees of freedom for the second variance).

captureChance

In order to determine the capture chance for alliance A, the cumulative distribution function for a normal distribution is used. This cumulative distribution form is defined as

$$p = F(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-(t-\mu)^2}{\sigma^2}} dt$$

where x is threshold value, μ is the mean of the sample, and σ is the standard deviation of the sample. In this case, x is the number of shots necessary to weaken the tower, μ is

$$\mu = \sum_{T \in A} hSA(T) + \sum_{T \in A} lSA(T) + \sum_{T \in A} hST(T) + \sum_{T \in A} lST(T)$$

where $hSA(T)$ is team T's `avgHighShotsMadeAuto`, $lSA(T)$ is team T's `avgLowShotsMadeAuto`, $hST(T)$ is team T's `avgHighShotsMadeTele`, and $lST(T)$ is team T's `avgLowShotsMadeTele`. The standard deviation σ is defined as

$$\sigma = \sqrt{\sum_{T \in A} hSA_\sigma(T)^2 + \sum_{T \in A} lSA_\sigma(T)^2 + \sum_{T \in A} hST_\sigma(T)^2 + \sum_{T \in A} lST_\sigma(T)^2}$$

With these two definitions, $F(x|\mu, \sigma)$ is the probability that alliance A will not weaken the opposing alliance's tower. In order to find the probability that the tower will be weakened, we subtract it from 1 to get $1 - F(x|\mu, \sigma)$. However, this does not include the probability of all three robots challenging or scaling the tower. While the product of these three probabilities would be intuitive, this ignores the fact that many teams will not challenge or scale the tower due to the focus on breaching. When these teams are placed on the same alliance as a team capable of weakening the tower, this team will do its best to ensure that all three robots end up on the batter at the end of the match. Therefore, $cC(A)$ can be more accurately estimated as

$$cC(A) = (1 - F(x|\mu, \sigma)) * \max(sC(T) \mid T \in A)$$

where $sC(T)$ is the `siegeConsistency` of team T.

breachChance

Due to the ease of a breach, most alliances have the capability to successfully perform a breach. The difficulty in achieving a breach is more a result of prioritizing and coordinating it. Therefore, the presence of a team practiced in achieving a breach will most like cause the alliance to perform a breach. Therefore, the breach chance for alliance A can be defined as the highest `breachPercentage` on the alliance:

$$bC(A) = \max(\{bP(T) \mid T \in A\})$$

where $bP(T)$ is the `breachPercentage` of team T.

[3] https://en.wikipedia.org/wiki/Cumulative_distribution_function
[4] https://en.wikipedia.org/wiki/WelchSatterthwaite_equation

predictedRPs

The `predictedRPs` $pRP(A)$ for alliance A is defined as

$$pRP(A) = 2 * wC(A) + cC(A) + bC(A)$$

predictedSeed

`predictedSeed` is calculated by sorting teams by their predicted total number of RPs $pRS_{total}(T)$. In order to maximize the accuracy of the prediction, we use the actual number of ranking points $nRP(M)$ generated by played matches and reserve the predicted number of ranking points $pRP(M)$ to unplayed matches.

$$pRS_{total}(T) = \sum_{M \in P} nRP(M) + \sum_{M \in U} pRP(M)$$

where P is the set of team T's played matches and U is the set of team T's unplayed matches. In the case of ties, the second and third tiebreakers, auto points and siege points respectively, are used. Similar logic is used in generating this predicted number of auto points and predicted number of siege points, with the actual number of points generated being used for played matches and the predicted number of points being used for unplayed matches. Using this method, the predicted number of auto points pAP_{total} is calculated using the formula

$$pAP_{total}(T) = \sum_{M \in P} nAP(M) + \sum_{M \in U} pAP(M)$$

where $nAP(M)$ is the `numAutoPoints` for match M and $pAP(M) = \sum_{T \in A} aA(T)$ is the predicted number of auto points for alliance A in match M. The third tiebreaker, total number of siege points $pSP_{total}(T)$, is defined as

$$pSP_{total}(T) = \sum_{M \in P} nSP(M) + \sum_{M \in U} pSP(M)$$

$nSP(M)$ is the `numSiegePoints` for match M and $pSP(M) = \sum_{T \in A} sA(T)$ is the predicted number of siege points for alliance A in match M.

Chapter 4

Conclusion

4.1 Review

4.1.1 Overall

The Citrus Circuits 2016 Scouting System was the most effective yet. Stable, fast, and easy to use, it allowed for flexible and powerful strategic analysis that was crucial in helping us to reach the Einstein field for the fourth year in a row.

4.1.2 Infrastructure

Firebase

Firebase was a core part of the scouting system and was a significant improvement over the combined Realm/Dropbox mechanism used in the 2015 system. The simplicity of integrating Firebase into applications helped us to accomplish more development and testing throughout the six-week build season. The automated syncing was highly reliable even in conditions where network connectivity was very poor. In fact, it only failed once, when the device was taken offline for approximately 12 hours and the data on Firebase was wiped and rewritten multiple times. We *highly recommend* the use of Firebase in FRC scouting systems.

Note: Firebase has been updated since the development of the Citrus Circuits 2016 Scouting System. Firebase now includes many new features, such as integrated photo and media storage, which we will be migrating the system to in the coming year.

Dropbox

Due to our poor experience with Dropbox last year, we scaled down our usage to photo storage. Fortunately, it excelled in this role and was perfectly consistent throughout the season once the uploading and downloading mechanism was refined. While sometimes complicated to use, Dropbox was an effective method of storing photo data too large to sync through Firebase.

4.1.3 Contact Us

We are highly interested in helping to develop the FRC scouting community and opening the power of an electronic scouting system up to other teams. If you have any questions, please contact us at `frc1678@gmail.com`. In addition, all of the code from the 2015 and 2016 scouting systems can be found on Github at `https://github.com/frc1678`.

Chapter 5

Appendix

5.1 Additional Resources

Simbots Seminar Series: Scouting and Match Strategy
Seminar by the renowned mentor of Team 1114 Karthik Kanagasabapathy. Covers various methods of scouting, development of match strategy, and the foundations of mathematical analysis techniques in FRC.
`https://www.youtube.com/watch?v=l8syuYnXfJg`

Overview and Analysis of FIRST Stats
Comprehensize overview of mathematical analysis of FRC and FTC teams. Covers concepts ranging from simple analyses such as OPR, DPR, and CCWM to more complex methods such as WMPR, EPR, and MMSE techniques. Some knowledge of linear algebra required.
`https://www.chiefdelphi.com/forums/attachment.php?attachmentid=19081&d=1433595051`

Citrus Circuits Fall Workshops
A variety of workshops/seminars put on by students and mentors on Team 1678. For those interested in scouting and strategy, I would recommend the "Strategy and Game Analysis" and "Scouting Development" workshops.
`http://www.citruscircuits.org/fall-workshops.html`

Team 2834 Scouting Database Explanation
Simple introduction the concept of OPR. Great for beginners and those without much knowledge of linear algebra.
`https://www.chiefdelphi.com/media/papers/download/2321`

The Blue Alliance
Excellent website for viewing data on FRC. Provides an API for easily pulling data from the site to use in custom calculations.
`http://www.thebluealliance.com`

5.2 Collected Data Point Reference

5.2.1 Scout

- `timd.scoutName:` The initials or name of the scout who scouted the match. Used for scout evaluations and tracking down the source of incorrect data.
- `timd.ballsIntakedAuto:` A list of the balls the robot intaked during the autonomous period.
- `timd.numBallsKnockedOffMidlineAuto:` The number of balls the robot removed from the midline during the autonomous zone without intaking them.
- `timd.timesSuccessfulCrossedDefensesAuto:` A dictionary of the time in milliseconds the robot took to cross the defense during the autonomous period.
- `timd.timesFailedCrossedDefensesAuto:` A dictionary of the time in milliseconds the robot took to attempt but fail to cross the defense during the autonomous period. An attempted cross is defined as one in which the robot's frame contacts the defense or enters the infinite vertical zone extending upward and downard from the defense. Entering the ramps of the Outerworks does not count as an attempt, as the robot has not yet entered the space above or below the defense.
- `timd.numHighShotsMadeAuto:` The number of high shots the robot successfully made during the autonomous period.

- `timd.numLowShotsMadeAuto`: The number of low shots the robot successfully made during the autonomous period.
- `timd.numHighShotsMissedAuto`: The number of high shots the robot missed during the autonomous period.
- `timd.numLowShotsMissedAuto`: The number of low shots the robot missed during the autonomous period.
- `timd.didReachAuto`: Whether or not the robot successfully achieved a reach during the autonomous period.
- `timd.numHighShotsMadeTele`: The number of high shots the robot successfully made during the teleoperated period.
- `timd.numLowShotsMadeTele`: The number of low shots the robot successfully made during the teleoperated period.
- `timd.numHighShotsMissedTele`: The number of high shots the robot missed during the teleoperated period.
- `timd.numLowShotsMissedTele`: The number of low shots the robot missed during the teleoperated period.
- `timd.numGroundIntakesTele`: The number of balls the robot intakes from the ground during the teleoperated period.
- `timd.numShotsBlockedTele`: The number of shots an opposing robot attempted that the robot successfully blocked from entering the goal by causing the ball to be deflected off of its structure.
- `timd.didChallengeTele`: Whether or not the robot successfully achieved a challenge.
- `timd.didScaleTele`: Whether or not the robot successfully achieved a scale.
- `timd.timesSuccessfulCrossedDefensesTele`: A dictionary of the time in milliseconds the robot took to cross the defense during the teleoperated period.
- `timd.timesFailedCrossedDefensesTele`: A dictionary of the time in milliseconds the robot took to attempt but fail to cross the defense during the teleoperated period. An attempted cross is defined as one in which the robot's frame contacts the defense or enters the infinite vertical zone extending upward and downward from the defense. Entering the ramps of the Outerworks does not count as an attempt, as the robot has not yet entered the space above or below the defense.
- `timd.didGetIncapacitated`: Whether or not the robot died during the match.
- `timd.didGetDisabled`: Whether or not the robot started the match dead or was absent from the match.

5.2.2 Super Scout

- `timd.rankTorque`: A qualitative measure of the robot's ability to push. Robots are ranked from 1 (Worst) to 3 (Best) of the robots on their alliance. The best robot on the field is given a 4 instead of a 3. If a robot does not take any action that allows the measurement of this characteristic, a 2 is entered. This system will from now on be called the *ordinal ranking system*.
- `timd.rankSpeed`: A qualitative measure of the robot's speed, measured by the ordinal ranking system.
- `timd.rankDefense`: A qualitative measure of the robot's ability to play defense, measured by the ordinal ranking system.
- `timd.rankBallControl`: A qualitative measure of the robot's ability to control the ball through actions such as intaking, moving, and scoring. Measured by the ordinal ranking system.
- `timd.rankAgility`: A qualitative measure of the robot's ability to navigate the field quickly and effectively, measured by the ordinal ranking system.
- `timd.superNotes`: Any notes that the super scouts believes need to be inputted about the robot.
- `redScore`: The score of the red alliance in that match. Only inputted by the red super scout.
- `blueScore`: The score of the blue alliance in that match. Only inputted by the blue super scout.

- `redDefensePositions`: The defenses and their positions that the red alliance must cross. Only inputted by the red super scout.
- `blueDefensePositions`: The defenses and their positions that the blue alliance must cross. Only inputted by the blue super scout.
- `redAllianceDidCapture`: Whether or not the red alliance successfully captured the opposing tower. Only inputted by the red super scout.
- `blueAllianceDidCapture`: Whether or not the blue alliance successfully captured the opposing tower. Only inputted by the blue super scout.
- `redAllianceDidBreach`: Whether or not the red alliance successfully breached the Outerworks. Only inputted by the red super scout.
- `blueAllianceDidBreach`: Whether or not the blue alliance successfully breached the Outerworks. Only inputted by the blue super scout.

5.2.3 Pit Scout

- `selectedImageUrl`: The Dropbox public url of the robot's main image that is displayed by default in the viewing application.
- `otherImageUrls`: A list of the Dropbox public urls of the other images for the robot.
- `pitOrganization`: A measure of how organized the team's pit is. Measured on a scale from 1 (Worst) to 5 (Best).
- `pitNotes`: Any notes the pit scout believes should be entered on the robot.
- `pitNumberOfWheels`: The number of non-tread wheels the robot has. For example, if the robot only has treads, a 0 would be entered. If the robot has two wheels and two treads, a 2 would be entered.
- `pitAvailableWeight`: The available weight the robot has, not counting bumpers and battery.
- `pitProgrammingLanguage`: The programming language used to program the robot.

5.3 Calculated Data Point Reference

5.3.1 Competition

- `firstPickAbility`: Score prediction calculation to determine the strength of the robot as a first pick.
- `secondPickAbility`: Weighted sum to determine the strength of the robot as a second pick.
- `actualSeed`: Seed calculated from collected data. Would ideally be pulled from TBA by default, but overridden if TBA of FIRST's reporting is down.
- `actualNumRPs`: Actual number of ranking points in the robot's matches calculated from collected data. Would ideally be pulled from TBA by default, but overridden if TBA or FIRST's reporting is down.
- `numAutoPoints`: Actual number of auto points in the robot's matches calculated from collected data. Would ideally be pulled from TBA by default, but overridden if TBA or FIRST's reporting is down.
- `predictedSeed`: Predicted seed at the end of qualification matches. Calculated using `predictedNumRPs`.
- `predictedNumRPs`: Predicted number of ranking points at the end of qualification matches. Calculated using `breachChance`, `captureChance`, and `winChance`.
- `highShotAccuracyAuto`: Average high shot accuracy for the robot in autonomous. Average of `timd.highShotAccuracyAuto` over matches.
- `lowShotAccuracyAuto`: Average low shot accuracy for the robot in autonomous. Average of `timd.highShotAccuracyAuto` over matches.

- `autoAbility`: Average contribution of points during the autonomous period. Average of `timd.autoAbility` over matches.
- `autoAbilityExcludeD`: Average contribution of points during the autonomous period not including points earned for crossing Class D defenses. Average of `timd.autoAbilityExcludeD` over matches.
- `autoAbilityExcludeB`: Average contribution of points during the autonomous period not including points earned for crossing Class B defenses. Average of `timd.autoAbilityExcludeB` over matches.
- `avgHighShotsMadeAuto`: Average number of high shots made during the autonomous period. Average of `timd.highShotsMadeAuto`.
- `avgLowShotsMadeAuto`: Average number of low shots made during the autonomous period. Average of `timd.lowShotsMadeAuto`.
- `twoBallAutoAccuracy`: Accuracy of high goal shooting in autonomous periods where the robot attempts a two-ball auto.
- `twoBallAutoAttemptedPercentage`: Percentage of matches where the robot attempts a two-ball auto.
- `avgBallsKnockedOffMidlineAuto`: The average number of balls the robot knocks off the midline during the autonomous period. Average of `timd.numBallsKnockeedOffMidlineAuto` over matches.
- `avgMidlineBallsIntakedAuto`: The average number of balls the robot intakes off of the midline during the autonomous period. Average of `timd.numBallsIntakedOffMidlineAuto` over matches
- `reachPercentage`: Percentage of matches that the robot reaches during the autonomous period.
- `teleopShotAbility`: The average number of points the robot contributes through shots made during the teleoperated period. Average of `timd.teleopShotAbility` across matches.
- `avgHighShotsMadeTele`: Average number of high shots the robot made during the teleoperated period. Average of `timd.highShotsMadeTele` across matches.
- `avgHighShotsAttemptedTele`: Average number of high shots the robot attempted during the autonomous period. Average of `timd.highShotsAttemptedTele` across matches.
- `avgLowShotsMadeTele`: Average number of low shots the robot made during the teleoperated period. Average of `timd.lowShotsMadeTele` across matches.
- `avgLowShotsAttemptedTele`: Average number of low shots the robot attempted during the teleoperated period. Average of `timd.lowShotsAttemptedTele` across matches.
- `highShotAccuracyTele`: Average high shot accuracy during the teleoperated period. Average of `timd.highShotAccuracyTele` across matches.
- `lowShotAccuracyTele`: Average low shot accuracy during the teleoperated period. Average of `timd.lowShotAccuracyTele` across matches.
- `avgGroundIntakesTele`: Average number of times the robot intakes a boulder from the ground. Average of `timd.numGroundIntakesTele` across matches.
- `avgShotsBlocked`: Average number of shots launched at the goal from an opposing robot that the robot blocked by placing the robot in a position such that the ball bounces off the robot and does not enter the goal. Average of `timd.numShotsBlocked` across matches.
- `blockingAbility`: The number of points on average the robot denies the other alliance by blocking its shots. Equal to `avgShotsBlocked` multiplied by five times the average `avgNumHighShots` across all robots in the competition.
- `siegeAbility`: The average number of points per match that the robot contributes through challenging and scaling. Average of `timd.siegeAbility` across matches.
- `siegeConsistency`: Percentage of matches in which the robot either scales or challenges.

- `challengePercentage`: Percentage of matches in which the robot successfully challenges.
- `scalePercentage`: Percentage of matches in which the robot successfully scales.
- `avgTorque`: The average super scout ranking of the robot's torque. Average of `timd.rankTorque` across matches.
- `zScoreTorque`: The number of standard deviations that the robot's `avgTorque` is away from the competition mean.
- `avgSpeed`: The average super scout ranking of the robot's speed. Average of `timd.rankSpeed` across matches.
- `zScoreSpeed`: The number of standard deviations that the robot's `avgSpeed` is away from the competition mean.
- `avgAgility`: The average super scout ranking of the robot's agility. Average of `timd.rankAgility` across matches.
- `zScoreAgility`: The number of standard deviations that the robot's `avgAgility` is away from the competition mean.
- `avgDefense`: The average super scout ranking of the robot's defense. Average of `timd.rankDefense` across matches.
- `zScoreDefense`: The number of standard deviations that the robot's `avgDefense` is away from the competition mean.
- `avgBallControl`: The average super scout ranking of the robot's ball control. Average of `timd.rankBallControl` across matches.
- `zScoreBallControl`: The number of standard deviations that the robot's `avgBallControl` is away from the competition mean.
- `avgDrivingAbility`: The average calculated ranking of the robot's driving ability. Average of `timd.drivingAbility` across matches.
- `zScoreDrivingAbility`: The number of standard deviations that the robot's `avgDrivingAbility` is away from the competition mean.
- `disfunctionalPercentage`: The percentage of the robot's matches in which the robot is dead or absent. The sum of the robot's `disabledPercentage` and its `incapacitatedPercentage`. Average of `timd.isDisfunctional` across matches.
- `disabledPercentage`: The percentage of the robot's matches in which the robot begins the match dead or is absent. Average of `timd.isDisabled` across matches.
- `incapacitatedPercentage`: The percentage of the robot's matches in which the robot dies during the match. Average of `timd.isIncapacitated` across matches.
- `breachPercentage`: The percentage of the robot's matches in which the robot's alliance successfully breaches the Outerworks.
- `defensesCrossableAuto`: A list of the defenses the robot can cross in the autonomous period in string form to be used in the strategy meeting spreadsheet. Example: "pc, mt, rp, rt, rw, lb". Calculated as the list of defenses the robot has crossed one or more times in the autonomous period.
- `defensesCrossableTele`: A list of the defenses the robot can cross in the teleoperated period in string form to be used in the strategy meeting spreadsheet. Example: "pc, mt, rp, rt, rw, lb". Calculated as the list of defenses the robot has crossed one or more times in the teleoperated period.
- `avgNumTimesDefensesCrossedAuto`: A dictionary of the average number of times the robot crossed each defense during the autonomous period. An average of `timd.numTimesDefenseCrossedAuto` over the matches where the robot faced the defense.
- `beachedByDefensePercentage`: The percentage of successful or beached crosses where the robot was beached by the defense. Only used for Category A defenses.

- `slowedByDefensePercentage`: The percentage of successful or beached crosses where the robot was slowed by the defense. Only used for Category A defenses.
- `unaffectedByDefensePercentage`: The percentage of successful or beached crosses where the robot was relatively unaffected in speed by the defense. Only used for Category A defenses.
- `avgNumTimesBeachedByDefense`: The average number of times per match the robot was beached by the defense. Only used for Category A defenses.
- `avgNumTimesSlowedByDefense`: The average number of times per match the robot was slowed by the defense. Only used for Category A defenses.
- `avgNumTimesUnaffectedByDefense`: The average number of times per match the robot was relatively unaffected in speed by the defense. Only used for Category A defenses.
- `avgSuccessfulTimesDefensesCrossedAuto`: A dictionary of the average number of times the robot successfully crosses the defense during the autonomous period when it faced the defense. Average of the number of entries in `timd.numTimesSuccessfulDefenseCrossesAuto` across the matches where the robot faced the defense.
- `avgSuccessfulTimesDefensesCrossedTele`: A dictionary of the average number of times the robot successfully crosses the defense during the teleoperated period when it faced the defense. Average of the number of entries in `timd.numTimesSuccessfulDefenseCrossesTele` across the matches where the robot faced the defense.
- `avgFailedTimesDefensesCrossedAuto`: A dictionary of the average number of times the robot failed an attempt to cross the defense during the autonomous period when it faced the defense. Average of the number of entries in `timd.numTimesFailedDefenseCrossesAuto` across the matches where the robot faced the defense.
- `avgFailedTimesDefensesCrossedTele`: A dictionary of the average number of times the robot failed an attempt to cross the defense during the teleoperated period when it faced the defense. Average of the number of entries in `timd.numTimesFailedDefenseCrossesTele` across the matches where the robot faced the defense.
- `avgTimeForDefenseCrossAuto`: A dictionary of the average time in milliseconds that the robot took to cross the defense during the autonomous period. Average of `timd.crossingTimeForDefenseAuto` across the matches where the robot faced the defense.
- `avgTimeForDefenseCrossTele`: A dictionary of the average time in milliseconds that the robot took to cross the defense during the autonomous period. Average of `timd.crossingTimeForDefenseTele` across the matches where the robot faced the defense.
- `predictedSuccessfulDefenseCrossesAuto`: A dictionary of the number of times the robot is predicted to cross the defense in the autonomous period. Based on the robot's past performance and its performance relative to other robots in the competition.
- `predictedSuccessfulDefenseCrossesTele`: A dictionary of the number of times the robot is predicted to cross the defense in the teleoperated period. Based on the robot's past performance and its performance relative to other robots in the competition.
- `crossingsSuccessRateForDefenseAuto`: A dictionary of the percentage of attempted crossings of the defense during the autonomous period that the robot successfully crosses the defense during the autonomous period.
- `crossingsSuccessRateForDefenseTele`: A dictionary of the percentage of attempted crossings of the defense during the teleoperated period that the robot successfully crosses the defense during the teleoperated period.
- `sdHighShotsAuto`: The standard deviation of the robot's `timd.numHighShotsAuto` across the robot's played matches.
- `sdHighShotsTele`: The standard deviation of the robot's `timd.numHighShotsTele` across the robot's played matches.
- `sdLowShotsAuto`: The standard deviation of the robot's `timd.numLowShotsAuto` across the robot's played matches.
- `sdLowShotsTele`: The standard deviation of the robot's `timd.numLowShotsTele` across the robot's played matches.

- `sdSiegeAbility`: The standard deviation of the robot's `timd.siegeAbility` across the robot's played matches.
- `sdGroundIntakes`: The standard deviation of the robot's `timd.numGroundIntakesTele` across the robot's played matches.
- `sdTeleopShotAbility`: The standard deviation of the robot's `timd.teleopShotAbility` across the robot's played matches.
- `sdAutoAbility`: The standard deviation of the robot's `timd.autoAbility` across the robot's played matches.
- `sdShotsBlocked`: The standard deviation of the robot's `timd.numShotsBlockedTele` across the robot's played matches.
- `sdMidlineBallsIntakedAuto`: The standard deviation of the robot's `timd.numBallsIntakedOffMidlineAuto` across the robot's played matches.
- `sdBallsKnockedOffMidlineAuto`: The standard deviation of the robot's `timd.numBallsKnockedOffMidlineAuto` across the robot's played matches.
- `sdSuccessfulDefenseCrossesAuto`: The standard deviation of the robot's `timd.numTimesSuccessfulDefenseCrossesAuto` across the robot's played matches where it faced the defense.
- `sdSuccessfulDefenseCrossesTele`: The standard deviation of the robot's `timd.numTimesSuccessfulDefenseCrossesTele` across the robot's played matches where it faced the defense.
- `sdFailedDefenseCrossesAuto`: The standard deviation of the robot's `timd.numTimesFailedDefenseCrossesAuto` across the robot's played matches where it faced the defense.
- `sdFailedDefenseCrossesTele`: The standard deviation of the robot's `timd.numTimesFailedDefenseCrossesTele` across the robot's played matches where it faced the defense.

5.3.2 Match

- `predictedRedScore`: The predicted score of the red alliance.
- `predictedBlueScore`: The predicted score of the blue alliance.
- `sdPredictedRedScore`: The standard deviation of the `predictedRedScore` using the standard deviations of the component data points. Used for determination of the `redWinChance`.
- `sdPredictedBlueScore`: The standard deviation of the `predictedBlueScore` using the standard deviations of the component data points. Used for determination of the `blueWinChance`.
- `redWinChance`: The chance of the red alliance winning the match. Calculated using Welch's t-test.
- `redBreachChance`: The chance of the red alliance breaching the opposing Outerworks. Calculated using the maximum `breachPercentage` on the red alliance.
- `redCaptureChance`: The chance of the red alliance capturing the opposing tower.
- `blueWinChance`: The chance of the blue alliance winning the match. Calculated using Welch's t-test.
- `blueBreachChance`: The chance of the blue alliance breaching the opposing Outerworks. Calculated using the maximum `breachPercentage` on the blue alliance.
- `blueCaptureChance`: The chance of the blue alliance capturing the opposing tower.
- `predictedRedRPs`: The predicted number of ranking points awarded to the red alliance. Calculated using `redWinChance`, `redBreachChance`, and `redCaptureChance`.
- `actualRedRPs`: The actual number of ranking points awarded to the red alliance.
- `predictedBlueRPs`: The predicted number of ranking points awarded to the blue alliance. Calculated using `blueWinChance`, `blueBreachChance`, and `blueCaptureChance`.
- `actualBlueRPs`: The actual number of ranking points awarded to the blue alliance.

5.3.3 Team In Match Data (TIMD)

- `timd.highShotAccuracyAuto`: The accuracy of the robot's high shots in the match's autonomous period.
- `timd.lowShotAccuracyAuto`: The accuracy of the robot's low shots in the match's autonomous period.
- `timd.highShotAccuracyTele`: The accuracy of the robot's high shots in the match's teleoperated period.
- `timd.lowShotAccuracyTele`: The accuracy of the robot's low shots in the match's teleoperated period.
- `timd.highShotsAttemptedAuto`: The number of high shots the robot attempted during the match's autonomous period.
- `timd.highShotsAttemptedTele`: The number of low shots the robot attempted during the match's autonomous period.
- `timd.lowShotsAttemptedAuto`: The number of high shots the robot attempted during the match's teleoperated period.
- `timd.lowShotsAttemptedTele`: The number of low shots the robot attempted during the match's teleoperated period.
- `timd.teleopShotAbility`: The number of points the robot contributed to the alliance score through high and low shots made during the match's teleoperated period.
- `timd.siegeAbility`: The number of points the robot contributed to the alliance score through scales and challenges in the match.
- `timd.autoAbility`: The number of points the robot contributed to the alliance score due to actions performed during the match's autonomous period.
- `timd.siegeConsistency`: Whether or not the robot successfully challenged or scaled.
- `timd.numRPs`: The number of ranking points the robot received from the match.
- `timd.drivingAbility`: The driving ability of the robot in the match.
- `timd.numAutoPoints`: The number of points the robot's alliance scored during the autonomous period.
- `timd.numScaleAndChallengePoints`: The number of points the robot's alliance scored through challenges and scales.
- `timd.numBallsIntakedOffMidlineAuto`: The number of balls the robot intaked off the midline during the autonomous period.
- `timd.beachedByDefensePercentage`: The percentage of successful or beached crosses where the robot was beached by the defense. Only used for the Category A defense the robot faced in the match.
- `timd.slowedByDefensePercentage`: The percentage of successful or beached crosses where the robot was slowed by the defense. Only used for the Category A defense the robot faced in the match.
- `timd.unaffectedByDefensePercentage`: The percentage of successful or beached crosses where the robot was relatively unaffected in speed by the defense. Only used for the Category A defense the robot faced in the match.
- `timd.totalNumTimesCrossedDefensesAuto`: The total number of crosses the robot accomplished during the autonomous period. Counts unscored crosses as well as scored crosses.
- `timd.wasDisfunctional`: Whether or not the robot was dead or absent for a significant part of the match.
- `timd.numTimesSuccessfulDefenseCrossesAuto`: A dictionary of the number of times the robot successfully crossed the defense during the autonomous period.
- `timd.numTimesSuccessfulDefenseCrossesTele`: A dictionary of the number of times the robot successfully crossed the defense during the teleoperated period.
- `timd.numTimesFailedDefenseCrossesAuto`: A dictionary of the number of times the robot attempted and failed to cross the defense during the autonomous period.

- `timd.numTimesFailedDefenseCrossesTele`: A dictionary of the number of times the robot attempted and failed to cross the defense during the teleoperated period.
- `timd.crossingsForDefensePercentageAuto`: The percentage of attempted defense crossings during the autonomous period at which the robot succeeds in crossing the defense.
- `timd.crossingsForDefensePercentageTele`: The percentage of attempted defense crossings during the teleoperated period at which the robot succeeds in crossing the defense.
- `timd.crossingTimeForDefenseAuto`: A dictionary of the average time in milliseconds the robot took per successful crossing of the defense during the autonomous period.
- `timd.crossingTimeForDefenseTele`: A dictionary of the average time in milliseconds the robot took per successful crossing of the defense during the teleoperated period.
- `timd.secondPickAbility`: The second pick ability of the robot based solely on its performance in the match.
- `timd.attemptedTwoBallAuto`: Whether or not the robot attempted a two ball auto in the match.