

2020 Citrus Circuits

Electronic Scouting System Whitepaper

Written by Jackson A, Lucca B, David C, Carl C, Thomas D, Rayko F, Emily F, Logan H, Claire H, Teo H, Adam H, Harry J, Ellie K, Kathy L, Emily L, Zatarra N, Sophia P, Kevin R, Jake R, Nathan S, Ethan T, Livy T, Kate U, Ludi W, Adam W, and Justin Y

September 6, 2020

Contents

1	Introduction	4
1.1	Bolded Terms	4
1.2	Purpose of Whitepaper	4
1.3	Why Scout	4
1.4	History	4
1.5	System Overview	4
1.6	Subteam Structure	5
1.6.1	Crews	5
1.6.2	Scouting Leadership Team	6
1.7	The Game	6
2	Principles	7
2.1	Organizational Principles	7
2.1.1	Team Culture About Scouting	7
2.1.2	Subteam Culture	7
2.1.2.1	Health	7
2.1.2.2	Collaboration	7
2.1.2.3	Leadership Opportunities	8
2.2	Software Principles	8
2.2.1	Purpose	8
2.2.2	Prioritization	8
2.2.3	Prioritization of Tasks	8
2.2.4	Limiting Scope	8
2.2.5	User Interface Optimization	9
3	Processes	10
3.1	Scouting Leadership Public Meetings	10
3.2	Subteam Procedures	10
3.2.1	Importance of Documentation	10
3.2.2	Meeting Summaries	10
3.2.3	Meeting Leads	10
3.2.4	Development	10
3.2.4.1	Development of Prioritized Data Fields	10
3.2.4.2	Task Management	10
3.3	Testing Processes	11
3.3.1	Reviews	11
3.3.2	Tests	11
3.3.3	Schema Changes	11
3.3.4	Field Tests	11
3.3.5	Daily Product Tests	11
3.4	Competition Process	11
3.4.1	Before Competition	11
3.4.2	During Competition	12
3.4.3	After Competition	13
3.5	Offseason	13
3.5.1	Offseason Training	13

4	Product	14
4.1	Software	14
4.1.1	Purpose of Our Software	14
4.1.2	Visualization	14
4.1.2.1	Picklist Editor	14
4.1.2.2	Match Strategy Viewer	15
4.1.3	Collection	16
4.1.3.1	Match Collection	16
4.1.3.2	Pit Collection	17
4.1.4	Data Transfer	17
4.1.5	Schema	18
4.1.6	Server	19
4.1.6.1	Local and Cloud Databases	19
4.1.6.2	Communicaitaion with TBA	19
4.1.6.3	Logging	19
4.1.7	Computations	19
4.1.7.1	QR Handling	19
4.1.7.2	Consolidation and Objective TIM Calcs	19
4.1.7.3	Inner Goals Regression	20
4.1.7.4	Subjective Team Computations	20
4.1.7.5	Pick Ability	20
4.1.7.6	Predictions	20
4.1.7.7	Functionality Without Internet	21
4.2	Hardware	21
4.2.1	Tablet Case	21
4.2.2	Laptop Case	22
4.2.3	Video System Case	22
5	Analysis	24
5.1	Data Accuracy	24
5.1.1	Overall Accuracy at LAN	24
5.1.2	Number of Disagreements Within Consolidation	25
5.1.3	Comparison of Consolidation Methods	26
5.1.4	Data Accuracy with Fewer Scouts	26
5.1.5	Takeaways	27
5.1.6	Accuracy of Inner Goals Regression	27
5.2	Strengths	27
5.2.1	Prioritization and Limiting Scope	27
5.2.2	Code Quality and Review Process	27
5.2.2.1	Code Quality	27
5.2.2.2	The Review Process and Knowledge Transfer	28
5.2.2.3	Code Reusability	28
5.2.3	Documentation	28
5.2.4	Reliability of the Match Collection App	28
5.3	Weaknesses	28
5.3.1	Knowledge Transfer in Offseason and Build Season	28
5.3.1.1	Software General	28
5.3.1.2	Issues With Collaborative Learning	28
5.3.2	Testing and Review Process Issues	29
5.3.2.1	No End to End System	29
5.3.2.2	Issues with Consistence of Tests	29
5.3.3	Work Distribution	29
5.3.4	Data Structure	29
5.3.5	Lack of Milestones	29
5.3.6	Insufficient User Feedback	29

6	Future Improvements	31
6.1	Overview	31
6.2	Offseason Education Revamp	31
6.3	Improved Testing Procedures	31
6.4	Improved Software Features	31
7	Conclusion	33
7.1	Overall	33
7.2	Utilization of Human Resources	33
7.3	Resources	33
7.3.1	2020 Scouting System Software	33
7.3.2	Agile Software Development	34
7.3.3	Past Whitepapers	34
7.3.4	The Blue Alliance	34
7.3.5	Fall Workshops	34
7.3.6	Simbots Seminar Series: Scouting and Match Strategy	34
7.3.7	Overview and Analysis of FIRST Stats	34
7.3.8	Contact Us	34
	Glossary	35

Chapter 1

Introduction

1.1 Bolded Terms

Within this document there are bolded terms, which are defined within the glossary for your convenience. To avoid confusion, please reference the glossary as bolded terms appear. The bolded terms are hyper-linked to their respective definitions. Some common terms are defined here:

- Scouting System - The system that 1678 uses to collect, process, and visualize data in order to aid in match strategy and picklist creation at competition. There are two apps that collect data: the **Pit Collection** app and the **Match Collection** app. The data from these apps is processed by the **Server**, then displayed on the **Viewer** app and the **Picklist Editor**.
- Software Scouting - A subteam of 1678 Citrus Circuits. The purpose of **Software Scouting** is to create the most valuable software possible to aid in 1678's competition picklist and match strategy creation.

1.2 Purpose of Whitepaper

The Scouting Whitepaper is a technical document that describes the electronic **Scouting System** used by FIRST Robotics Competition (FRC) Team 1678: Citrus Circuits. The purpose of the 2020 Scouting Whitepaper is to share our approach to developing the 2020 **Scouting System** and the effectiveness of that approach with other FRC teams.

1.3 Why Scout

Citrus Circuits develops an electronic **scouting system** to be as competitively successful as possible by gaining insight to teams' competitive capabilities. This allows us to develop successful match strategies and picklists that best suit our strategic needs. By using electronic devices to collect, process, and visualize scouting data, the data can be continuously updated and sent to users. This allows for more complex calculations and enables a wider range of data visualization. Most importantly, it provides faster communication between components of the **Scouting System**.

1.4 History

Citrus Circuits has used a custom-built electronic **scouting system** since 2013. Every year since then, we have drastically improved the **Scouting System** and the training we provide to new members. The evolution of the **Scouting System** can be observed across the annual scouting whitepapers since 2013.

The resources available to us have also greatly increased each year. This year, **Software Scouting** has grown to be the largest subteam on Citrus Circuits with 24 members.

1.5 System Overview

The Citrus Circuits **Scouting System** is used to collect, process, and visualize data about teams at FRC competitions. **Software Scouting** develops five main software elements: the **Match Collection** app, the **Pit Collection** app, the Match Strategy **Viewer** app (**Viewer** app for short), the **Picklist Editor**, and the **Server**.

At competition 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:

- **Objective scouts** use the objective mode of the **Match Collection** app.
- **Subjective scouts** use the subjective mode of the **Match Collection** app.
- **Objective pit scouts** use the objective mode of the **Pit Collection** app.
- **Subjective pit scouts** use the subjective mode of the **Pit Collection** app.

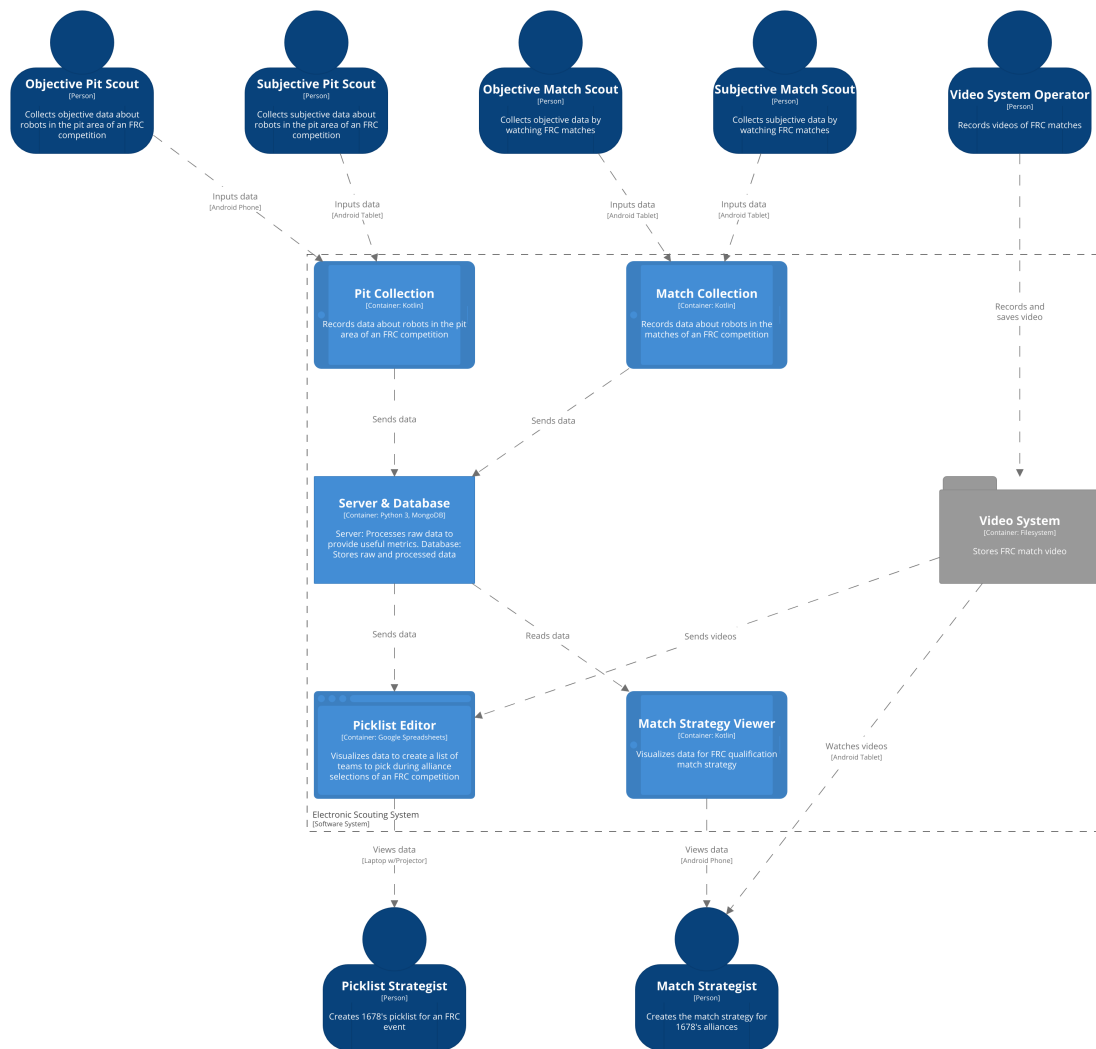


Figure 1.1: Diagram of the **Scouting System**, with competition roles

As users enter data into collection apps, those apps send data to the **Server**, which uses that data to compute various metrics of teams' abilities. Those computed metrics are then sent to the **Viewer** app and the **Picklist Editor**, which display data used for match strategy and picklist creation, respectively.

1.6 Subteam Structure

1.6.1 Crews

Software Scouting is divided into two **crews**: front-end and back-end. Front-end is responsible for the user-facing software (e.g. the **Match Collection** app and the **Picklist Editor** spreadsheet). Back-end is responsible for the **Server**, which manages the database, and handles computations. The two **crews** provide a crucial separation of tasks in a large subteam and enable members to specialize in specific programming languages and concepts.

1.6.2 Scouting Leadership Team

The **Scouting Leadership Team** is a group of 6 veteran members within **Software Scouting** that helps with the management and leadership of the subteam. Since **Software Scouting** has 24 members, the **Scouting Leadership Team** was created to increase communication between the **crews** (and within the subteam as a whole), as well as to help the subteam lead with management and leadership tasks for the large subteam. The **Scouting Leadership Team** mainly helps with organizing discussions and prioritization of tasks.

1.7 The Game

Objective and **subjective scouts** collect data about robots that participate in the 2020 FRC game, Infinite Recharge, at competitions. The following video gives a brief overview of the game: <https://www.youtube.com/watch?v=gmiYWTmFRVE>.

The official rulebook for the Infinite Recharge is available at: <https://firstfrc.blob.core.windows.net/frc2020/Manual/2020FRCGameSeasonManual.pdf>.

Chapter 2

Principles

2.1 Organizational Principles

2.1.1 Team Culture About Scouting

Citrus Circuits fosters a positive culture around scouting. We dedicate many hours during competition season to train **objective** and **subjective scouts**, and dedicate over half of our 44-person competition team to scouting. Additionally, the entire team, including mentors, emphasizes the importance of scouting on our team and how it contributes to our competitive success.

2.1.2 Subteam Culture

We believe that the most effective teams provide their members with an environment in which team members feel valued as people and are encouraged to support each other and work together. This environment has three key elements: prioritizing the health of our members, promoting a highly collaborative environment, and providing every member the opportunity to participate in large-scale improvements to our subteam.

2.1.2.1 Health

We believe that the health and well-being of our members is more important than their contributions to our team. As such, we frequently encourage all of our members to take care of themselves (e.g., getting enough sleep, taking breaks) and remind each other of this commitment when we see someone not taking care of themselves.

2.1.2.2 Collaboration

Developing valuable, working software at the scale of the **Scouting System** is a difficult task that cannot be accomplished by a single person. Effective collaboration is essential for the success of our subteam. To facilitate this, we supplement our subteam culture with two key ideas: idea sharing and knowledge transfer.

Idea Sharing

We encourage everyone to share all of their ideas, regardless of whether they think it is a good or bad idea, as we believe that the best ideas come from people building off of the ideas of others. Furthermore, we strive to understand the intention and background behind ideas, rather than shutting them down based on perceived flaws or concerns with implementation.

We stress the importance of this because historically, some of our best ideas initially seemed absurd. For example, we joked about using QR codes for data transfer in the beginning of the 2018 season. Two years later, QR codes continue to be the most reliable and convenient method of data transfer from the collection to processing components of the **Scouting System**.

Knowledge Transfer

In order to collaborate effectively, every member of our team needs to have access to the knowledge that they need to contribute to our team. Important knowledge includes, but is not limited to:

- Our users' needs
- What our subteam has tried in the past and how well it worked
- Which tasks should be worked on next

- How a piece of software will fit in with the rest of the system
- How to implement a specific software concept
- Unforeseen problems or delays with a task
- Known issues with our subteam or the **Scouting System**

2.1.2.3 Leadership Opportunities

In order to improve as quickly as possible, we strive to give every member of our subteam the opportunity to suggest and implement improvements. Scouting Leadership meetings are the primary method we use to encourage members to participate in making large-scale improvements. We also try to delegate (see Scouting Leadership Team, Meeting Summaries, and Meeting Leads) to engage as many members as possible in the leadership of our subteam and to foster a focus on the success of the team over that of individuals.

2.2 Software Principles

2.2.1 Purpose

The purpose of the **Scouting System** is to aid in 1678's competition picklist and match strategy creation. In the development of the **Scouting System**, picklist creation is a higher priority than match strategy creation.

2.2.2 Prioritization

When developing the **Scouting System**, we prioritize at several primary scopes:

Most broadly, we prioritize accuracy and reliability over features – it is better to have less data than to have inaccurate or inaccessible data. To achieve this, we prioritize the features of the visualization software in order to limit the scope of the **Scouting System**. This limitation makes it easier to deliver accuracy and reliability by providing a clear set of features to implement.

While technically not a scope of prioritization, reusability is another main principle we consider when developing the **Scouting System**. If the core of the **Scouting System** can be reused from year to year, it can be improved and built upon by multiple generations of students. This reuse also allows students to focus more on year-specific features to ensure the system as a whole is as reliable and accurate as possible, making reusability a key component of the first level of prioritization.

2.2.3 Prioritization of Tasks

We prioritize individual tasks based on the difficulty of the task and the value we estimate the task to provide. Prioritizing visualization features enables us to estimate the relative value that tasks will provide. The processes that we use to prioritize visualization features and individual tasks are Prioritized Data Fields and Task Management, and are described in the next chapter.

When prioritizing, every item is ranked ordinally. Additionally, we focus more on the order of the highest priority items because they will be implemented sooner, while the priorities of lower items will often be shifted around multiple times before the features are implemented.

2.2.4 Limiting Scope

In order to develop a reliable and accurate **Scouting System**, we limit the scope of our software as much as possible while still providing the functionality needed to assist in picklist and match strategy creation. To do so, we limit the number of programming languages used and the amount of data fields visualized (and as a result, data fields collected).

2.2.5 User Interface Optimization

When designing the UI for collecting match data, our goal is to minimize the amount of time a user spends entering data to maximize the time the user is watching the match. This makes users less likely to miss data that should be recorded. We also limit the amount of data collected to maximize the accuracy of the data that is collected.

Pit data collection does not require the same speed of input as match data collection, so the UI is optimized for accuracy (e.g. confirmations, ability to edit data), followed by ease of input.

Both match strategy and picklist visualization are optimized for speed of understanding data. Match strategy is done between qualification matches, and picklist creation is done in a short meeting after the competition venue closes—in both situations, the speed at which data can be understood is the limiting factor for how well the data can be utilized.

Chapter 3

Processes

3.1 Scouting Leadership Public Meetings

The **Scouting Leadership Team** holds both private and public meetings. Private meetings include discussions about specific individuals and the product management of **Software Scouting**, while the public meetings revolve around broader discussions with more feedback and ideas (e.g. productivity, training). Any interested member of **Software Scouting** is encouraged to come to the public meetings and can submit topics for discussion to a Slack channel.

3.2 Subteam Procedures

3.2.1 Importance of Documentation

To enable knowledge transfer between members of our team, we strive to document as much information as possible in a written, publicly accessible format. We primarily use public Slack channels and the 1678 Google Drive to document procedures specific to **Software Scouting** and other project specific documents, such as meeting notes, discoveries, and external resource links.

3.2.2 Meeting Summaries

To keep all members of our subteam updated about our work and progress, meeting summaries are posted in Slack after each weekend meeting and every two weekday meetings. They summarize what progress was made and the next steps for a specific **crew**. Any member of **Software Scouting** can sign up to write a meeting summary.

3.2.3 Meeting Leads

Any member of **Software Scouting** can sign up to lead an official subteam meeting. They take on responsibilities such as giving announcements, preparing the workspace and equipment, taking attendance, and making sure that our area is clean after the meeting ends.

3.2.4 Development

3.2.4.1 Development of Prioritized Data Fields

Working with 1678's Strategy subteam, **Software Scouting** creates a spreadsheet of prioritized fields of data to be visualized. This allows us to include the necessary data points, focusing on their accuracy, instead of developing software to get less important data points. Prioritizing our data fields avoids feature creep, which happens when a system has many new features added to it beyond what was originally planned for. Feature creep causes overcomplication and leads to a worse product. Additionally, prioritizing our visualized data fields lowers the likelihood of initially developing collection for data fields that turn out to be of little importance.

3.2.4.2 Task Management

In order to organize software tasks, the **Scouting Leadership Team** utilizes GitHub's project board feature as a kanban board. This board contains the prioritized list of tasks, which are created according to the principles listed under Prioritization of Tasks. As features are assigned, worked on, and merged into the **codebase**, the **Scouting Leadership Team** and GitHub automation keep the kanban board up-to-date. Scouting developers are expected to keep track of their tasks and check the kanban board throughout the season.

3.3 Testing Processes

3.3.1 Reviews

When a **Software Scouting** member has written code that is ready to be put into the master **codebase**, they will need reviews to ensure all master code is functional at all times. This both catches issues with code and increases knowledge about the code in the subteam. All code written needs both of the following reviews:

- **Buddy Review:** A line-by-line code review process done with someone who is a member of the same **crew**. This is often helpful with catching syntax errors or other common problems.
- **Peer Review:** A summary of what the code does and the logic behind each action completed with someone who is not on the same crew. This helps identify logic issues or confusing code.

3.3.2 Tests

Code must also be tested before being put into the master **codebase**. A front-end developer needs both a user test and an edge test, while a back-end developer only needs an edge test because the same functionality is covered in the edge test.

- An edge test is completed by someone in the same crew as the code developer, and is specifically used for testing edge cases. The tester intentionally tries to crash the app to discover uncommon bugs. This is critical for catching bugs before they are introduced into the master **codebase**.
- A user test is completed by someone in a different crew as the developer to see if the general functions and specific features of the app work. This simulates normal usage of the app and provides information on how easy the app is to use.

3.3.3 Schema Changes

In our software, the **schema** is the outline for our data fields and data structures. Throughout the build season, the **schema** files are updated and edited. **Schema** changes have a separate review process as they are a crucial part of our system's function and cannot be easily tested. **Schema** review changes consist of one review from a **Scouting Leadership Team** member of each crew. Reviewers read through the changes and evaluate if the **schema** should be changed and if the proposed change is the best data format. They also look for areas that may require comments.

3.3.4 Field Tests

As part of the testing process, we schedule multiple tests of the entire **Scouting System** before each competition that aim to simulate the competition environment as realistically as possible. We collect data by scouting match videos to allow calculations to be tested with realistic data. As the data is collected, we process it and display the results as we would at competition, allowing us to identify bugs that occur when we integrate the individual parts into one system. Field tests are also used to gain feedback on the user interface and user experience. All members of the **Software Scouting** subteam participate in these tests as users.

3.3.5 Daily Product Tests

We regularly test each **codebase** to catch bugs that went unnoticed during the review process. These tests run on the latest version of each product. We test each app, document any errors, and run the **server** on the output of the match collection apps. These tests are meant to catch errors that reduce app functionality to allow field tests to be more effective in testing the overall system.

3.4 Competition Process

3.4.1 Before Competition

Preparing the **Scouting System** for a competition is critical, as we want it to be reliable at competition to aid in picklist and match strategy creation. If any issues exist within our team's robot or **Scouting System**, a competition is sure to find them. It is with this in mind that **Software Scouting** follows three processes prior to any competition.

The first is **scout training**, where scouts practice using the app and submitting data. Experience and exposure is critical to data accuracy, and **scout training** is often the first experience that scouts have with the **Match Collection** app. This training usually takes place the week before a competition. While the training for **subjective scouts** is developed by the mentors of the strategy subteam, the training for **objective scouts** is routine and simulates a much more common competition experience. Prior to Objective **scout training**, all **objective scouts** are required to read documentation for Objective **Match Collection** (linked here), which they are quizzed on later.

The physical devices that operate the **Scouting System** are also prepared prior to the competition. All devices have their test data deleted and the latest version of all apps installed.

Finally, all **Software Scouting codebases** have a competition-specific copy made prior to competition. During competition, hotfixes are administered into the copy instead of directly to the main **codebase**. This separation allows fixes to be reviewed and merged to the main **codebase** after the competition.

3.4.2 During Competition

At competition, all 1678 students have specific roles that they are trained to fulfill. These roles include:

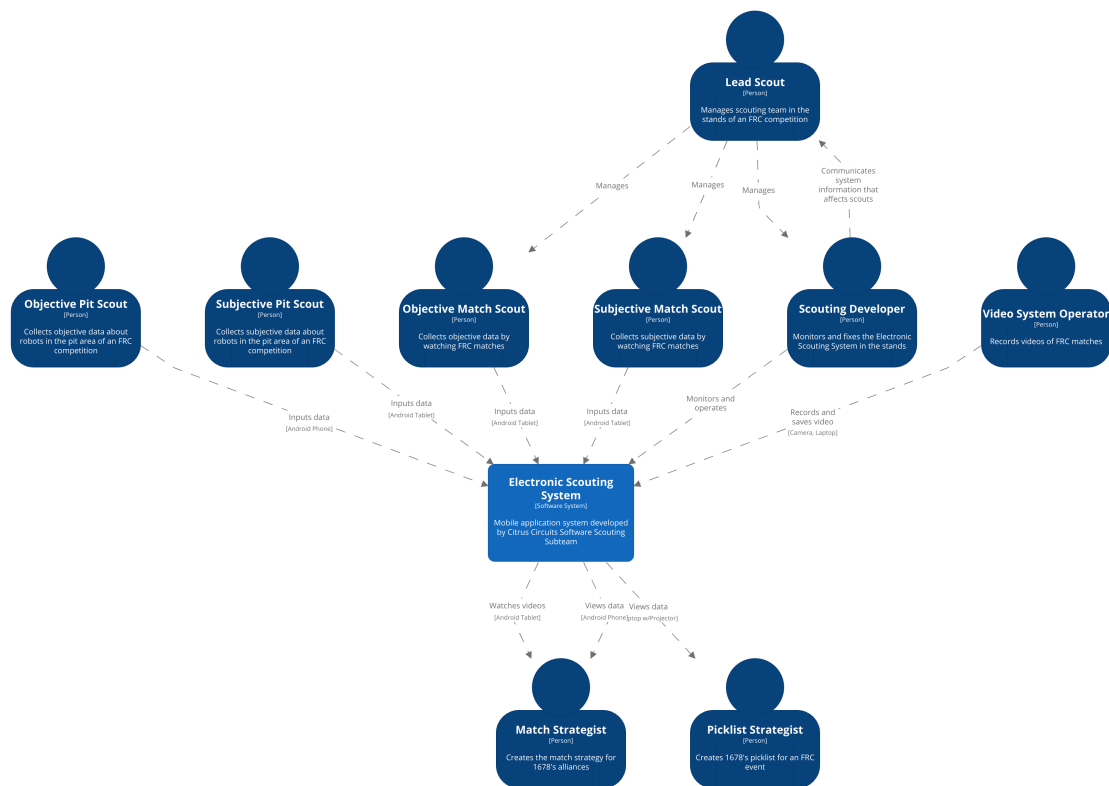


Figure 3.1: Diagram of competition roles

- **Objective scouts** (enough to fill empty spots on the travel team) and **subjective scouts** (4-6)
 - Utilize the **Match Collection** app to collect data about robot performance in the qualification matches.
- **Objective pit scouts** (1-2) and **subjective pit scouts** (2-4)
 - Collect data about individual robots prior to qualification matches using the **Pit Collection** app.
- **Video system operators** (2)
 - Operate the **video system** in order to record qualification and playoff matches. Matches that 1678 played in are viewed by the 1678 drive team in order to review their performance.

- Lead scout and assistant lead scout (1 of each)
 - Relays operational information to the scouts (objective and subjective) in order to improve data accuracy and to ensure smooth operation of the **Match Collection** app. Also responsible for scout break rotations.
- **Software scouting developers** (2-4)
 - Fix any bugs found at the competition and ensure the system is operating correctly. At competition, **software scouting developers** actively monitor the **Scouting System**.

Upon arrival to the stands on competition day, the **Scouting System** is set up by the **software scouting developers**. The **video system** is set up within this time as well.

Once qualification matches begin all **objective scouts** and **subjective scouts** utilize the **Match Collection** app to collect data about the matches that they view. During competition, the scouting developers monitor the transfer of data and administer fixes to the system if and when the system encounters an error.

Objective scouts and **subjective scouts** are given breaks once every two hours or so for approximately thirty minutes.

3.4.3 After Competition

After a competition, **Software Scouting** holds a debrief about the **Scouting System** including mentors and students that wish to participate. Each member gives an answer to the current question one at a time. All answers are recorded by a predesignated note taker. The questions usually include:

- What went well?
- What could have gone better?
- What are we going to do to improve?

Having a debrief of this nature has been shown multiple times to produce meaningful feedback that has benefited our subteam tremendously. Without this debrief, many issues could easily be left unaddressed.

Meetings after the debrief are used to create fixes for issues discovered during the competition and continue developing features for either the next competition or season.

3.5 Offseason

3.5.1 Offseason Training

In order to prepare all new software students for build season, new software students go through a training called Software General. Veteran members of 1678's two software teams, **Software Scouting** and **Software Robot**, teach new software students about the principles of programming. The concepts that students learn within Software General can be applied to every programming language and aid them in developing code for our robot and **Scouting System**. Students, whether they are veteran or new developers, are expected to work together during Software General. This encourages knowledge transfer and the development of collaboration skills. Once the training is complete, the new students choose between one of the two software subteams to continue their season on.

Once a week, veteran developers present a basic programming concept (e.g. operators, control flow, functions, classes), after which the new software students receive an assignment to practice said concept. For the remainder of the weekly meetings, the students work on the assignments, and veteran members help the students and answer questions.

All submitted assignments are scored by members of both software subteams for code quality and style. All incoming software members are expected to demonstrate an understanding of the concepts covered on every assignment, and to revise their assignments until this proficiency is demonstrated.

We also use the offseason to continue development of the **Scouting System**, which helps prepare both new and veteran developers for build season. Offseason development also allows us to make improvements to the **codebase**, reducing the work needed in the next season.

Chapter 4

Product

4.1 Software

4.1.1 Purpose of Our Software

The goal of our software product is to provide useful data in order to form effective match strategy and aid in our picklist creation. Our software is composed of three components—collection, processing, and visualization. Collection is the software used to collect data for robots in matches and in the pits. The **server** then processes this data by running computations on the data and pushing raw data and the results of these computations into the database. This data is visualized for the creation of our picklist and match strategy.

4.1.2 Visualization

4.1.2.1 Picklist Editor

Search the menu (Alt+V)																		
100% 123- 18 100% 123- 18																		

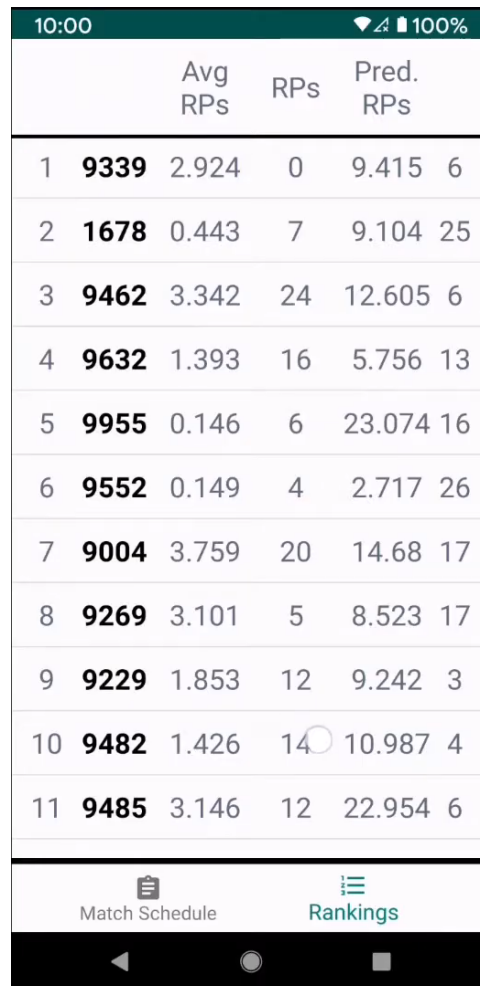
Figure 4.1: The Picklist Editor

At regionals, Citrus Circuits creates its picklist in the evening of the first day of qualification matches, where **strategists** use data to rank the teams in order of ability to contribute to our alliance. During the alliance selection process, we pick the highest available team on our picklist. The **Picklist Editor** displays scoring abilities, driver ability, match videos, robot information, images, and graphs for each team using data exported from the **Server**.

Code for the **Picklist Editor** can be found here: <https://github.com/frc1678/picklist-editor-public/tree/2020>

Picklist Editor video: <https://youtu.be/WLi4QmJkbb4>

4.1.2.2 Match Strategy Viewer



		Avg RPs	RPs	Pred. RPs	
1	9339	2.924	0	9.415	6
2	1678	0.443	7	9.104	25
3	9462	3.342	24	12.605	6
4	9632	1.393	16	5.756	13
5	9955	0.146	6	23.074	16
6	9552	0.149	4	2.717	26
7	9004	3.759	20	14.68	17
8	9269	3.101	5	8.523	17
9	9229	1.853	12	9.242	3
10	9482	1.426	14	10.987	4
11	9485	3.146	12	22.954	6

Figure 4.2: The Viewer app (data is faked)

The Match Strategy **Viewer** app, or **Viewer** app, allows **strategists** to access scouting data on a smartphone. The **Viewer** app is used to create our team's qualification match strategy. It displays the match schedule of our team, predicted rankings and scores, team details, and current rankings.

Code for the **Viewer** app can be found here: <https://github.com/frc1678/viewer-public/tree/2020>

Viewer video: <https://youtu.be/WLi4QmJkbb4?t=96>

4.1.3 Collection

4.1.3.1 Match Collection



Figure 4.3: The Objective Match Collection Mode

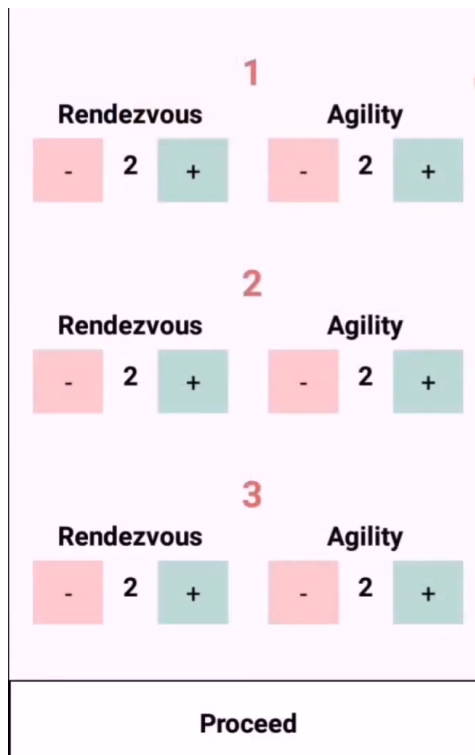


Figure 4.4: The Subjective Match Collection Mode

To collect data about robots in a match, the objective **Match Collection** and subjective **Match Collection** modes are used by their respective scouts. Both modes are contained in the **Match Collection** app. Objective **Match Collection** is used by scouts to collect quantitative data for one robot, whereas subjective **Match Collection** is used by **subjective scouts** to collect qualitative data for an alliance of 3 robots. These apps use primarily toggle buttons, counters, and timers to record data, as well as QR codes for data transfer. Both modes have four screens: Team Assignment, Data Collection, Edit Information, and QR Display. Other than the Data Collection Screen, all of these screens are mostly the same.

The code for the 2020 **Match Collection** app can be found here: <https://github.com/frc1678/match-collection-public/tree/2020>

Objective **Match Collection** Video: <https://youtu.be/WLi4QmJkbb4?t=151>

Subjective **Match Collection** Video: <https://youtu.be/WLi4QmJkbb4?t=219>

4.1.3.2 Pit Collection

The screenshot shows the 'Objective Collection' mode of the Pit Collection app. At the top, a teal header bar displays 'Version: 2.0.2, Objective Collection'. Below this, the number '1678' is prominently displayed in a large, dark font. To the right of the number is a green square icon containing a white camera symbol. Below the number and icon are two red rectangular buttons with white text: 'CAN NOT CROSS TRENCH' and 'CAN NOT GROUND INTAKE'. Further down are two light gray rectangular input fields labeled 'Drivetrain' and 'Drivetrain Motor Type'. At the bottom, there is a label '# of Motors' next to a horizontal line for text entry, and a large gray 'SAVE' button to its right.

Figure 4.5: The Objective Pit Collection Mode

The screenshot shows the 'Subjective Collection' mode of the Pit Collection app. At the top, a teal header bar displays 'Version: 2.0.2, Subjective Collection'. Below this, the number '1678' is prominently displayed in a large, dark font. Underneath the number is the text 'Climber Strap Installation Difficulty' followed by two columns of radio button options numbered 1 through 10. Below the radio buttons is a large, empty rectangular text area labeled 'Climber installation notes'. At the bottom of the screen is a large gray 'SAVE' button.

Figure 4.6: The Subjective Pit Collection Mode

The **Pit Collection** app is a data collection app used to collect physical properties of robots in the pit area of an FRC competition. During the practice and qualification matches, **subjective pit scouts** and **objective pit scouts** record data by visiting each team in the pits and asking them questions about their robot. The app has two modes: **Objective Pit Collection** and **Subjective Pit Collection**. In **Objective Pit Collection**, the pit scout records objective information about the robot and takes photos of the robot. **Subjective Pit Collection** records qualitative details about a team's climb and their ability to participate in a buddy climb with 1678. This data is then saved and sent to the **Server**. The **Pit Collection** app has 4 main screens: Mode Selection, Team List Screen, Objective Collection Screen, and Subjective Collection Screen.

The code for the 2020 **Pit Collection** app can be found here: <https://github.com/frc1678/pit-collection-public/tree/2020>

Objective **Pit Collection** video: <https://youtu.be/WLi4QmJkbb4?t=238>

Subjective **Pit Collection** video: <https://youtu.be/WLi4QmJkbb4?t=283>

4.1.4 Data Transfer

Before competitions, we send the match schedule and team list from the **server computer** to tablets and phones using **Android Debug Bridge (ADB)**. The **server** uses **ADB** to pull data from all devices after they are connected to the **server computer**. **ADB** is the only way we pull data from the **Pit Collection** app, and is a secondary method for pulling data collected through match scouting. The primary way that data is transferred from the **Match Collection** app to the **server** is through QR codes containing match data. With QR codes, there is no need to connect all tablets to a central port which is plugged into the **server**, which is difficult to do quickly in the stands. Instead, all that needs

to be handed around is a wireless QR scanner, which eliminates the use of a wired system.

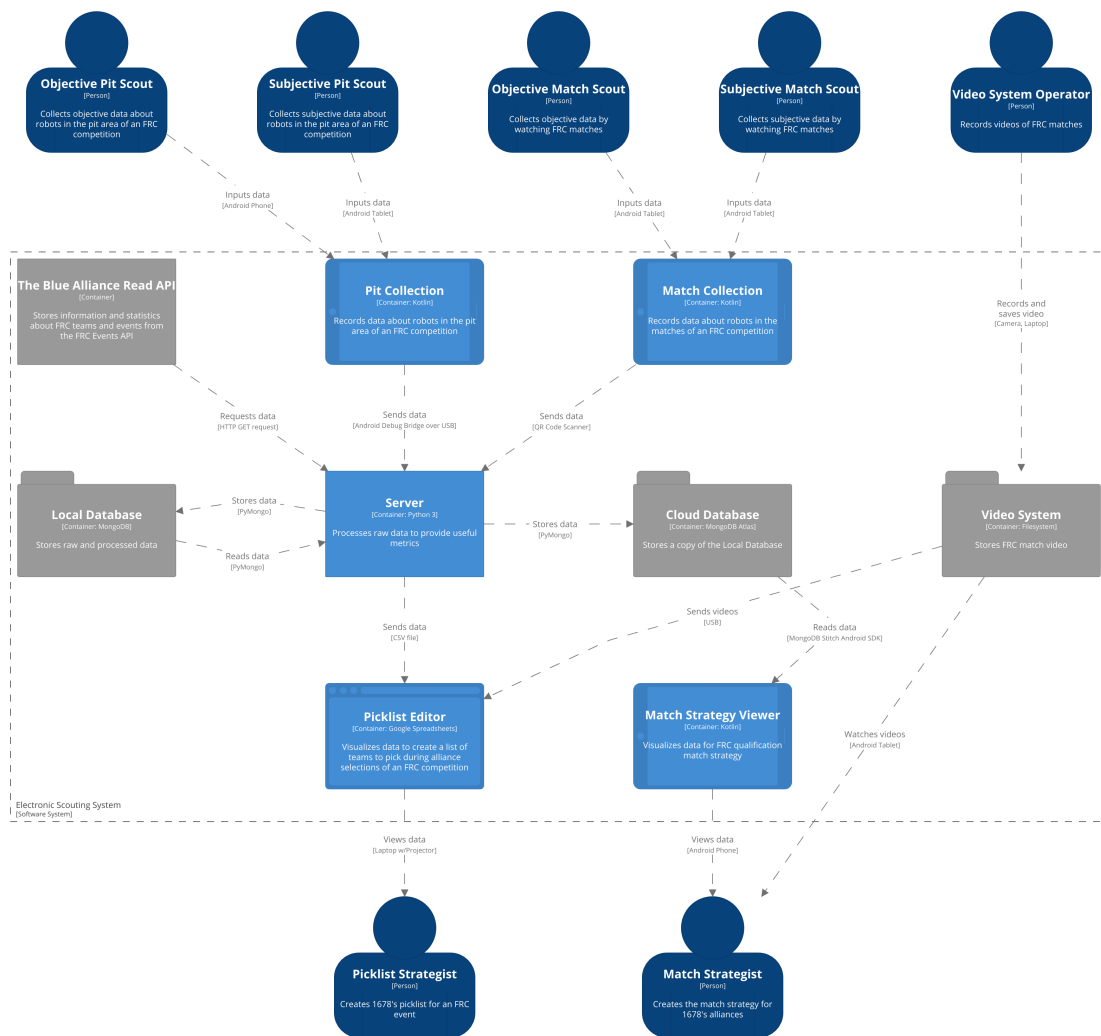


Figure 4.7: Data transfer of the **Scouting System**

When **subjective scouts** and **objective scouts** finish scouting their robots for the match, the **Match Collection** app takes all entered data and compresses it into a string, which is then generated into a QR code and scanned by the **objective** and **subjective scouts** to be scanned into the **server** console. **Match Collection** also saves the compressed string to a local file on the tablet in case the QR code is not scanned. Once all QR codes are scanned for a match, a **server** cycle runs.

The **Viewer** app then pulls and caches the cloud database locally. In order to transfer data to the **Picklist Editor**, a CSV file is generated by the **server** upon request and imported into the **Picklist Editor**.

4.1.5 Schema

In our software, **schema** is the blueprint for our data structure. We use **schema** files to store the standardized data structure that is used by all apps.

Schema files give our developers the ability to change data fields in a single location and thus increases code reusability between different FRC games and their changing data fields. They also associate most calculated data fields with a basic type of calculation such as averages or counts, allowing developers to quickly create new calculations without writing new code. Developers can also easily change the weighting for calculations which combine other metrics without altering **Server** code, making it easier for picklist **strategists** to change the value of weights during

competition. A separate GitHub repository was used to version control the **schema** files.

The 2020 **Schema** repository can be found here: <https://github.com/frc1678/schema-public/tree/2020>

4.1.6 Server

4.1.6.1 Local and Cloud Databases

All of our data is stored on a MongoDB database. Processing is done on data stored in the local database, and the **server** pushes changes to the local and cloud databases. Both databases contain the exact same data. For testing purposes, we can pull data from the cloud database to update the local database we test on. To efficiently write many changes to the cloud database, we use bulk writes, which are multiple write operations made in a single call.

Documentation for MongoDB: <https://docs.mongodb.com/manual>

4.1.6.2 Communicaitaion with TBA

We use The Blue Alliance's Read APIv3 (<https://www.thebluealliance.com/apidocs/v3>) to pull the match schedule, match results, team list, and rankings. In the database, we store a cache of all the data we receive from TBA, as well as a timestamp of when the data last changed on TBA. Caching allows us to reduce the number of requests made, and the timestamp allows us to not download data in the event it has not changed since the previous request.

4.1.6.3 Logging

Terminal output is used at competition to give scouting developers a clearer understanding of what the **server** is doing. Logging is also used to track information about the processes of the **server**. Essentially every process that the **server** performed was logged to a file. Most processes performed by the **server** are logged to a file, which makes it easier to trace issues in **server** processes or major errors to assist in debugging.

4.1.7 Computations

4.1.7.1 QR Handling

All our data is transferred from the tablets used by scouts to the **server computer** in a compressed format. This allows us to fit more data into the QR codes, as our scanners can only store 500 characters for each QR. We use the same compressed format for the secondary method, pulling from files over **ADB**. The end result is that **Match Collection** data needs to be decompressed into a usable format before anything else can be done with the data. This decompression is done according to the same **schema** file that the QR is compressed with.

We also have the ability to blocklist or modify QRs. We do not correct errors in the data because there is no way to ensure that the correction would be valid. Instead, we assign three scouts to each robot, and use a **consolidation** algorithm (see Consolidation and Objective Tim Calcs) to decide on a value between all three scouts. However, we do have the capability to fix errors such as the wrong team or match number being recorded, as this has the potential to severely diminish the accuracy of the **consolidation** algorithm.

The capability to blocklist QRs is generally used only in the event of a match replay. We do not delete any raw data to prevent mistakes leading to data loss. Instead, if it is determined that data should not be used, the raw QR text is added to a blocklist which then prevents it from being used in future calculations.

4.1.7.2 Consolidation and Objective TIM Calcs

In order to have the most accurate data possible, we assign three **objective scouts** for every robot in each match. This redundancy ensures accurate data even if one **objective scout** makes a mistake. **Consolidation** is the process of taking the **unconsolidated Team In Match data (TIM)** and figuring out "what actually happened" based on those. All the categorical data fields we **consolidate** are boolean (data that is true or false, e.g. whether a robot crossed the initiation line) or numerical (data represented by a number, e.g. the number of goals scored in the upper goal).

To **consolidate** boolean data, we use the result that the majority of scouts obtained. If only two scouts are assigned to a robot and they disagree about an event, we say that the event did not happen.

To **consolidate** numerical data, such as action counts or time spent climbing, we first look to see if scouts agreed. If there is one value agreed on more than others, we say that is the number that happened. Otherwise, the **server** calculates a weighted average of the values, where the weight is the squared reciprocal of the distance (in standard deviations) from the mean of these values.

The code for Objective Team in Match Calculations can be found here: https://github.com/frc1678/server-public/blob/2020/calculate_obj_tims.py

4.1.7.3 Inner Goals Regression

Since it is very difficult for scouts to tell whether a robot is scoring inner goals or outer goals, we had them report only the total number of high goals. We then assumed that the proportion of high goals that go into the inner goal is constant for each team. Using the total number of inner goals for each alliance, which TBA gives us, and the total number of high goals scored by each team, which our **objective scouts** give us, we can approximate each team's proportion of inner goals to high goals.

We find a “best solution” that will minimize the total square error, running only on teams that have scored high goals. This method, least squares, is also used for calculating **Offensive Power Ranking (OPR)**. Finally, we clipped the result for each team to be between 0 and 1 so that it is more usable for calculations and makes more sense for **strategists**.

The code for the inner goals regression can be found here: https://github.com/frc1678/server-public/blob/2020/inner_goals_regression.py

Information about how **OPR** is calculated can be found here: <https://blog.thebluealliance.com/2017/10/05/the-math-behind-opr-an-introduction/>

4.1.7.4 Subjective Team Computations

Driver ability is a statistic for teams that is designed to represent driver skill. This metric forms an important part of the assessment of the defensive ability of a team because many teams do not play defense during qualification matches, but driver ability is very strongly related to how well a team plays defense. This metric is based on the agility and rendezvous rankings **subjective scouts** give to teams in each match. These scores are combined into the driver ability statistic based on weights determined by **strategists**.

4.1.7.5 Pick Ability

Strategists begin developing our picklists by using computer-sorted lists of teams and editing them after the first day of qualification matches. For each robot, the **server** calculates two values to summarize how much they would contribute to our elimination-round alliance: first and second pick ability, which are used to create first drafts of our first and second picklists, respectively. These scores are important because they reduce the amount of work and discussion required during the **picklist draft** by giving a preliminary sort of teams. Each ability score is a weighted average of a team's offensive and additionally, for second pick ability, defensive capabilities. Pick ability roughly represents how much of a point swing a team will contribute to our alliance, either by scoring points or preventing points from being scored. Throughout the season and during competitions, we adjust the weights used to calculate pick ability.

4.1.7.6 Predictions

Using the data that we collect during a competition, we try to predict what will happen later in the competition. By looking at the averages of scoring abilities for each team in an alliance, we can calculate their chance of scoring enough power cells to get a ranking point or the chances of the alliance getting the climbing RP. As we collect more data, we are able to accurately predict scores and ranking points for future matches, as well as teams' final seedings. These help us develop our match strategies and our picklists.

4.1.7.7 Functionality Without Internet

Another important feature is the ability for the **server** to run without Internet, albeit with limited functionality. Without Internet, we can still collect and store data from QRs and tablets attached over USB, perform objective calculations that do not require data from TBA, and run calculations involving subjective data. While recent hardware acquisitions reduce the need for this functionality by improving the reliability of Internet access, it is still essential to be able to collect and store data to prevent loss.

4.2 Hardware

Because the **Scouting System** is so specialized, it requires a unique and diverse range of hardware. The goal of this section is to explain how the different physical parts of the system work together, as well as some of the reasoning behind choosing specific devices and a few of their shortcomings. However, it is certainly not necessary to have a complicated system in order for it to be effective, and most scouting systems are smaller while still being effective. Our **Scouting System** has reached this point by steadily improving since 2013

Every device in the **Scouting System** must be safely stored and transported. To do this, the hardware components of the system are placed into different cases depending on their use. There are two tablet cases, a laptop case and a **video system** case. Our main priority for the cases was the ability to act as carry-on luggage for an airplane, because they would hold lithium ion batteries and therefore could not be checked. To ensure that the devices are well-protected and easy to transport, we looked for cases which were waterproof and had wheels. All of these features are found in the Nanuk 935 cases we chose.

4.2.1 Tablet Case

The tablet case (Figure 4.8) is used to keep track of tablets and phones and keep them charged during the whole season. The devices are all plugged into a powered USB hub, which allows all of them to be charged or updated at once. The USB hub is connected to a power supply. The cases have a built in charger for each tablet. We use Lenovo Tab E7 tablets for the **Match Collection** app and the objective mode of the **Pit Collection** app. Pixel 3a mobile phones run the **Viewer** app and the subjective mode of the **Pit Collection** app. The phones are included so that all **strategists** have an Android phone to use the **Viewer** app on, since we discontinued the iOS version of the **Viewer** app to more effectively develop the other apps.



Figure 4.8: The Tablet Cases

4.2.2 Laptop Case

The laptop case carries server laptops for competition. It also holds three Tera Wireless QR scanners, each with a charging base. This case also carries the Netgear 4G LTE Modem and two hotspot antennas to connect to the Internet.

The wireless scanners are used to get data from the **objective scouts** and **subjective scouts** tablets by scanning QR codes generated after each match based on what the scouts inputted. During the season, we noticed some issues and limitations with the scanners, such as the 500-character limit for QR codes stored on the scanner before uploading, and a bug where scanning certain length QR codes with less delay settings cause problems uploading.

The case also holds the laptop used to run the **server**, which is a Lenovo Thinkpad E590 running Ubuntu 18.04 LTS.

The case contains a hotspot to allow the **Scouting System** to access the Internet at competitions. It is a NETGEAR 4G LTE Modem attached with an antenna. It connects to a NETGEAR 5-port network switch, which connects to the server laptop and allows it to communicate with the **Picklist Editor** laptop and the cloud database. The hotspot and network switch do not have batteries and are powered by a DC-UPS (Direct Current Uninterruptible Power System) so that we can maintain an Internet connection even without a power outlet. An inverter stored in the **video system** case powers the UPS and the laptops.

4.2.3 Video System Case

During competition, the **video system** is used to record match videos for reference by **strategists**. The main components of the system are the camcorder, tripod, and inverter.

The camcorder is a Canon Vixia HF100. It uses a wide-angle conversion lens (Raynox HD-5050 Pro Wide Angle Conversion Lens 0.5x) to widen its field of view to fit the whole field. The camcorder is mounted on a tripod to hold it up to the appropriate height.

For power supply during competition, the **video system** case contains a Potek P1500 inverter which is connected to a robot battery to power the camcorder and charge its extra batteries. A problem discovered was that the inverter's leads would not always properly connect to the battery, which resulted in occasional power loss. This was fixed after the LAN Regional.

From the camcorder, videos were transferred via SD card to the **Picklist Editor** laptop, and occasionally to a tablet for a **strategist** (see Figure 4.9).

Finally, the Video System case holds miscellaneous items such as gaff tape and a sandbag.

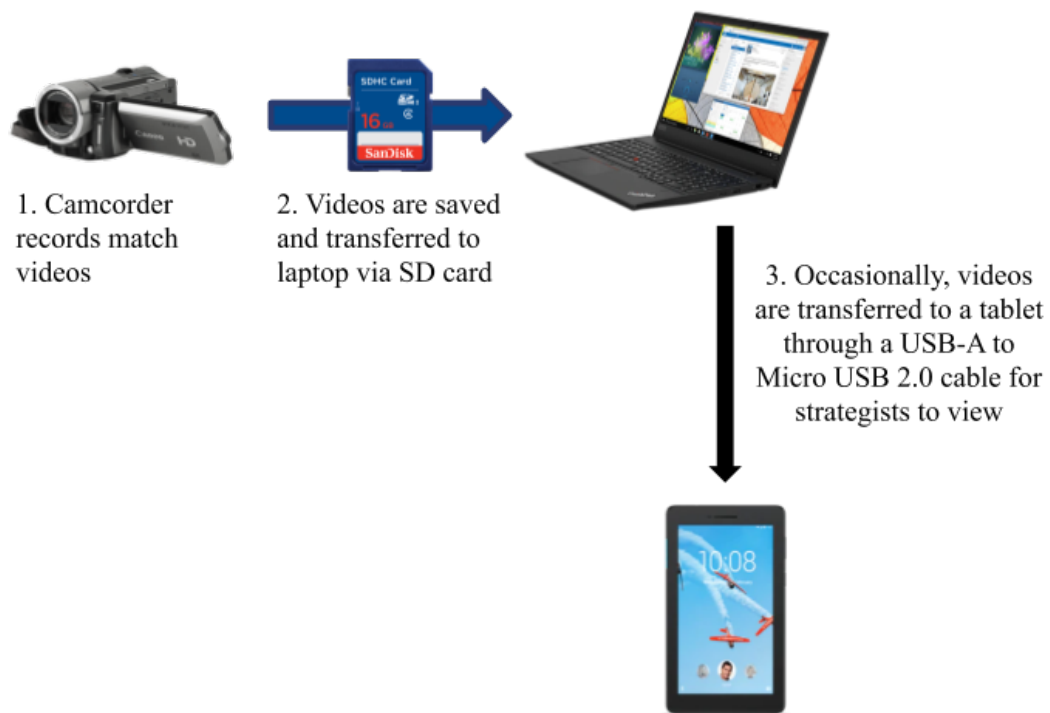


Figure 4.9: Diagram of Data Transfer within the **Video System**

Chapter 5

Analysis

5.1 Data Accuracy

We obtain a rough estimate of how accurate our data is by comparing the total number of balls scored by an alliance with the sum of the number of balls scored by each team on that alliance according to our scouts. If our **objective scouts** were perfect, there would be zero difference, but the farther off they are, the more they will disagree with TBA's total alliance counts – so ideally, we want to minimize the average error. Below, we look at how the mean absolute difference changes if we have fewer scouts or if we use a different **consolidation** algorithm.

5.1.1 Overall Accuracy at LAN

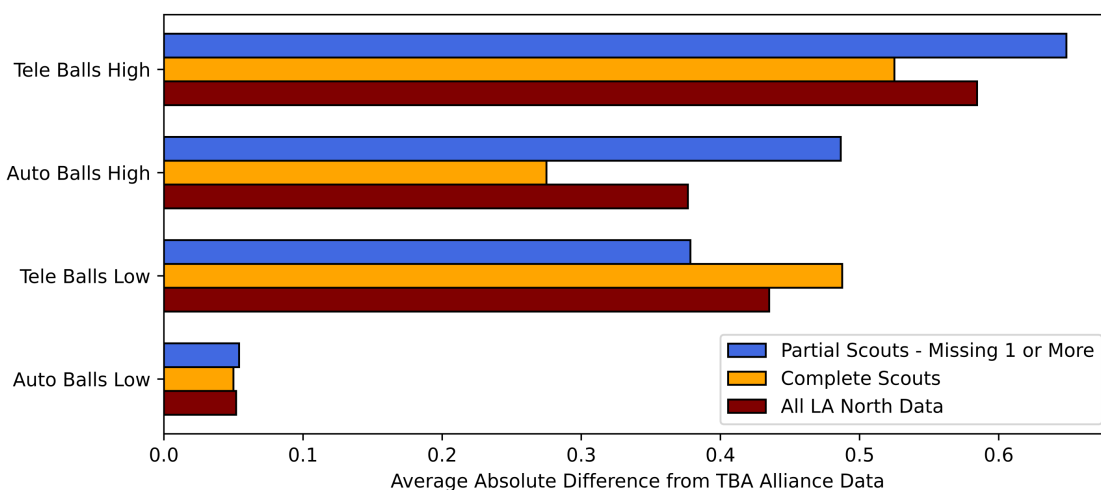


Figure 5.1: Mean absolute error with complete and incomplete sets of **objective scouts**

At LA North, there was an unknown issue that led to data loss, resulting in missing data for most matches from a single tablet. This issue led to us not having a full set of **objective scouts** on a single robot in slightly under half of all matches scouted, which in turn had a negative impact on data accuracy. This impact can be seen when we compare the sum of the scouting data for each robot on an alliance to the scoring data stored on TBA for that alliance. In general, the Mean Average Error for alliances with at least one robot missing an **objective scout** was higher than for alliances with complete **objective scouts**. Power cells scored in the lower port during teleop do not meet this trend, however, that type of scoring happened infrequently compared to power cells being scored in either the inner or outer port. We think it is likely that this discrepancy is due to noise or other factors unrelated to the number of **objective scouts**.

5.1.2 Number of Disagreements Within Consolidation

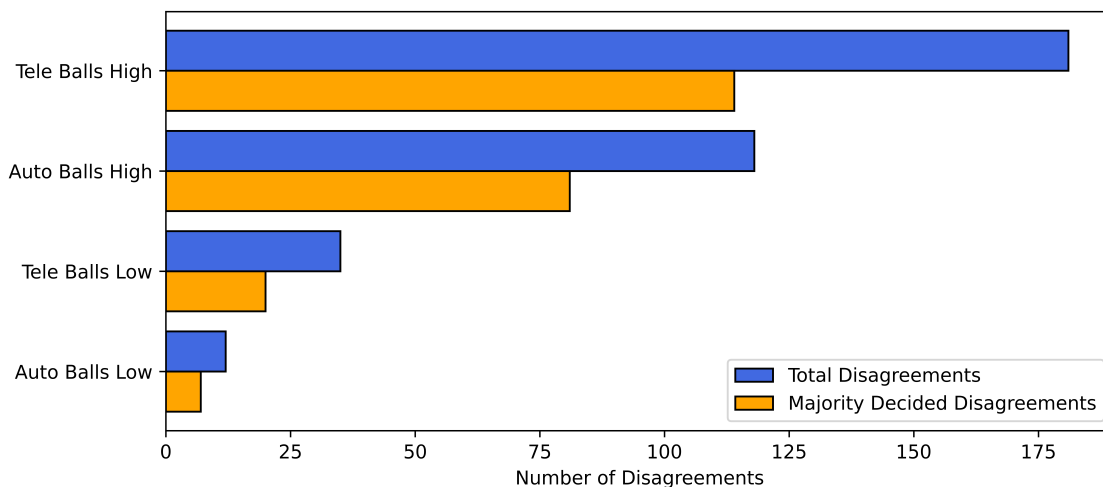


Figure 5.2: Number of Disagreements by Type and Datafield

This data on the number of disagreements between **objective scouts** demonstrates how crucial a good **consolidation** system is. At LA North, there were 352 total instances for which not all **objective scouts** agreed on the scoring of one robot. Some of these disagreements were across different data fields for the same robot, but there were still disagreements in a large majority of matches, showing the influence the **consolidation** algorithm had on data accuracy. Also important is that in the majority of disagreements, we were able to **consolidate** based on the agreement of the majority scouts, shown in Figure 5.2 under “Majority Decided Disagreements”. **Objective scout** disagreements are also at least roughly related to the frequency of the action:

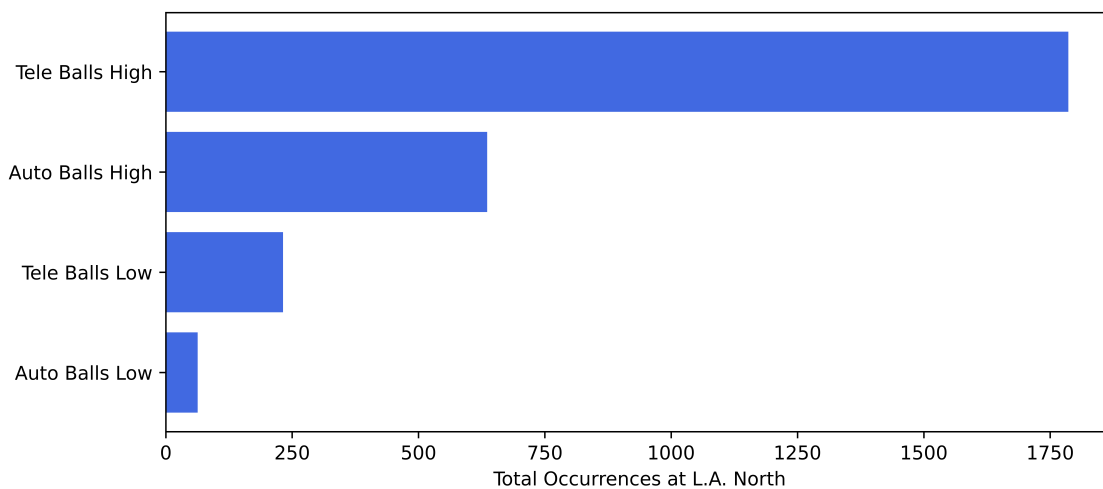


Figure 5.3: Total Occurrences of Common Scoring Types in Qualification Matches at LA North

We can see that actions that happen more frequently are disagreed on more, which makes sense. The more often an action happens, the more often errors recording it should happen assuming a constant error probability.

5.1.3 Comparison of Consolidation Methods

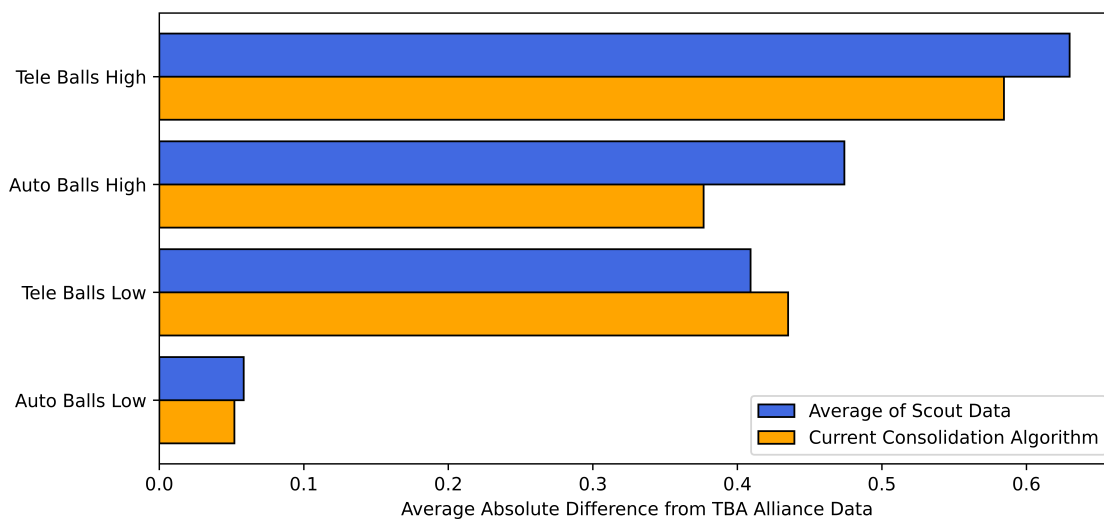


Figure 5.4: Comparison of Mean Absolute Error Using Our **consolidation** Algorithm and **consolidation** by Averaging **Objective Scout** Data

The graph above shows that our **consolidated TIMs** were closer to TBA's alliance totals than they would have been if we just averaged **unconsolidated TIM** data. Although averaging appears to have worked slightly better for Teleop low balls, our **consolidation** algorithm, which puts more weight on **objective scouts** that were closer to the average, worked better overall.

5.1.4 Data Accuracy with Fewer Scouts

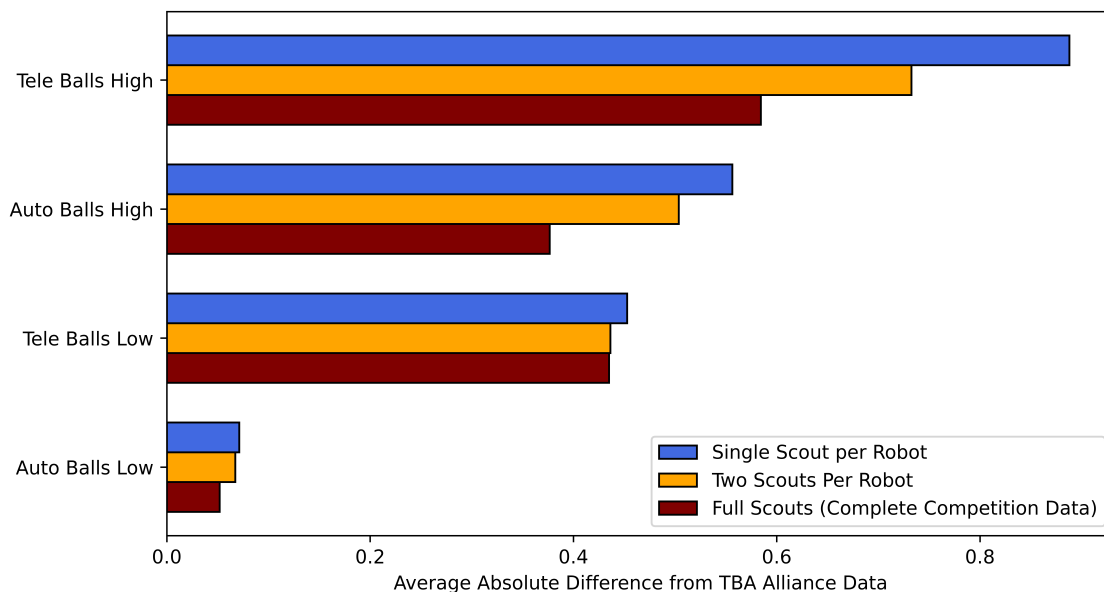


Figure 5.5: Average Absolute Difference From TBA Alliance Data with 1, 2, and 3 Scouts

One of the greatest contributors to the accuracy of the system is the number of scouts. More **objective scouts** means that after **consolidation**, the data should be less vulnerable to instances of human error leading to inaccurate data. To demonstrate this, we used the results from each individual **objective scout** to determine the expected

accuracy of both one and two **objective scouts** per robot. Both took the average of every possible combination of the data we acquired using three **objective scouts** to give as accurate a comparison as possible. For two scouts, we used our existing **consolidation** method to **consolidate** each data point. It is clear that for the high goal, three **objective scouts** is significantly better than two both in autonomous and during the teleoperated period. However, the low goal sees little difference between the two, although more variance is expected from scouting results for the low goal because significantly less data was collected.

5.1.5 Takeaways

Scouting the low goal, while more accurate than the high goal, was much more error prone during the teleoperated period than might have been expected based on frequency. This is likely because the low goal was harder to scout than the high goal: most **objective scouts** during **scout training** found that the low goal was more difficult to scout because robots would drive up to it, making it hard to tell how many power cells were scored. This could possibly be addressed by more training as it could make **objective scouts** more consistent in determining the number scored. High Goals in Autonomous were also likely more disagreed upon and more inaccurate due to the short time of autonomous, which often lead to multiple robots scoring in the high goal at once.

Despite this, the overall accuracy of the system was very good, with an average error of less than one game piece per alliance per match for every scoring location. Per robot, the average absolute error for all scoring locations combined was well under one game piece per match, meaning that the data for each robot at the end of competition was very close to the actual number. This met our prioritization of data accuracy over data quantity, in keeping with the philosophy that bad data is worse than no data. Using multiple well trained scouts for each robot and strong **consolidation** algorithms allowed us to maintain significantly higher data accuracy than would have been possible with a single scout per robot.

5.1.6 Accuracy of Inner Goals Regression

The regression 1678 used for calculating inner goals scored by each team treats teams' inner goal accuracy rates (inner goals scored divided by high goals scored) as constant between matches, even though they are not, which results in inaccurate estimates for how many inner goals teams score in each match. This inaccuracy can be measured by looking at the difference between how many inner goals the alliance scored in a match (according to TBA) and how many inner goals the alliance would have scored if their accuracy rates were constant. The standard error for alliances' number of inner goals in autonomous was .834, and 1.533 for teleop. This indicates that the regression method was off by a fair amount, but it was probably the best option available, since **objective scouts** can not be expected to reliably tell which goal a ball enters. Nonetheless, the calculated accuracy rates for each team were reasonably good representations of the overall average accuracy rates across their matches.

5.2 Strengths

5.2.1 Prioritization and Limiting Scope

This year, we simplified our system by identifying collection fields early in the season, and focusing on not duplicating data that could be collected from The Blue Alliance (TBA). By collaborating with the strategy subteam, we were able to determine the data we needed to visualize, and then determine what data we needed to collect. This involved making the decision to not collect data on whether a high goal was scored in the inner or outer port, as it was too difficult to scout and we prioritized data accuracy. Another improvement we made was limiting the number of platforms used, which saved developer resources and made it easier for developers to share knowledge. This also allowed programmers to use the same tools, increasing productivity and the ability of people to help others.

5.2.2 Code Quality and Review Process

5.2.2.1 Code Quality

Additionally, this year developers focused more on code quality. Namely, style was more consistent across each code base, assisted by increased implementation of style guides. Overall, code was also better commented and more readable than in the past. These improvements helped make parts of the system more robust, particularly those completed earlier in the season as these sections were most carefully written and reviewed.

5.2.2.2 The Review Process and Knowledge Transfer

These code quality improvements were facilitated by an improved review process. Although our review process was not ideal, it was significantly improved from last year, and was somewhat successful in its goals of catching errors and helping developers understand more of the system.

5.2.2.3 Code Reusability

We also emphasized code reusability this season. The use of **schema** allowed us to minimize code modifications, define the weights of variables in the **schema** rather than in the code, and modify or create some calculations without code changes. Developers working on all parts of the system utilized **schema** as much as possible, and worked to make calculations, QR decompression, the displaying of data, etc. more game-agnostic. Programmers working on the **Match Collection** app and the **Pit Collection** app also defined sizes in resource files, and used styles to allow changes for different elements to be made in a single place, utilizing reusable superclasses. The **Viewer** app uses configuration files to change what is displayed, and a “translation file,” which maps database name to the displayed name. Finally, the Objective and Subjective **Pit Collection** app's code is very reusable and works on phones and tablets, through the use of the same view for objective and subjective data collection, and it does not change much from year to year.

5.2.3 Documentation

Consistent documentation also helped simplify knowledge transfer about subteam processes, for example using the 2019 whitepaper as a reference, particularly to teach new members more about our subteam. We also gave regular meeting summaries to assist people in understanding what progress is made during each meeting.

5.2.4 Reliability of the Match Collection App

Finally, our **Match Collection** app's code this year was very reliable. During Los Angeles North (which was the only competition we were able to attend this year), the **Match Collection** app only experienced one minor bug where minimal match data was lost, partially due to scouts forgetting to advance the data collection mode to teleoperated. The **Match Collection** app was tested the most out of all our code this year, and showed the benefits brought by our subteam's strengths this year. This the first time in the **Match Collection** app's history that there were little to no performance issues at the first competition of the season.

5.3 Weaknesses

5.3.1 Knowledge Transfer in Offseason and Build Season

5.3.1.1 Software General

Software General training's goal is to teach new software members, from both **software robot** and **software scouting**, the essential skills they need in order to learn subteam-specific skills during the offseason and then contribute to the subteam's goals during the season. Unfortunately, the development of the 2019 Software General curriculum was very rushed and the students did not gain the critical skills for build and competition season. The curriculum was too focused on the technical aspects of writing code, rather than the essential skills of critical thinking or the collaboration that we wanted members to gain during the offseason.

The lessons were taught by veteran software members, some of whom were not always engaged with their peers, not interested in teaching, or not fully comfortable with the content being taught. Additionally, assignments affiliated with each lesson were too large to allow students to properly practice the concepts taught in the lesson. Due to the differing experience levels of new members, there were often students who finished assignments early and were left to either help others or do nothing until everyone was finished with the assignment.

5.3.1.2 Issues With Collaborative Learning

Learning how to develop the **Scouting System** is a very collaborative process, one which we had some difficulties with. A prominent issue in our subteam was due to an imbalance of skill level and technical confidence, which led to difficulties in communication, asking for help, and completing tasks. This year, we had 12 new developers, which led to a strain on veteran developers in both training and in the workload. This caused veterans to have more difficulty

learning and training themselves in preparation for the season and slowed down progress on some tasks. There were also issues in the willingness of some subteam members to collaborate on projects, leading to difficulties in productivity on tasks. More, the staggering ratio between veterans and new members led to an even larger technical knowledge gap.

5.3.2 Testing and Review Process Issues

5.3.2.1 No End to End System

One major reason our subteam struggled with testing was the lack of an end to end system. Although we had originally made it our first priority to create a full end to end system, due to inconsistent deadlines, problems with scanners, and both **crews** working at different paces, the end to end system was created much later in the season than expected. Due to this delay, it was much harder to run full system tests. Without these large-scale tests, there were more faults in the code at competition across both ends.

5.3.2.2 Issues with Consistence of Tests

Our review process this year, while improved from in the past, still had many issues. Although it caught many mistakes, it still let through major logic errors and even completely broken code. This flaw primarily stemmed from the extreme inconsistency of tests and reviews, which ultimately completely depended on the reviewer/tester. Many members did not understand the importance of reviews and tests, which led to them staying silent when they did not understand the code. With a wide range of technical experience and knowledge, as well as different confidence levels, the effort and depth of reviews and tests changed widely depending on the person; inconsistency greatly led to countless bugs getting merged into master code.

5.3.3 Work Distribution

The distribution of tasks off the backlog was often problematic. Since our most experienced developers were usually busy, we assigned difficult or high priority tasks to available developers, which often ended up with developers being given tasks that were too difficult for them at the time. Occasionally, we would assign low priority tasks before higher priority tasks, since the developers who were able to do the high priority tasks were occupied. Another issue was that some tasks would have too many people assigned to it, while larger tasks wouldn't have enough people. To make up for these issues, many scouting members were switched from their assignments mid-project, causing a lot of confusion and hindering progress on certain tasks.

There were also issues with prioritizing tasks on the backlog. We didn't always identify which tasks were holding others up, and sometimes those blocking tasks that should've been high priority were lower on the backlog than they should have been.

5.3.4 Data Structure

We established our data structure before any of us understood how to use MongoDB properly, and stuck with that architecture for the rest of the season. Our functions for communicating with the database were convoluted and unreliable, and were all built for very specific interactions.

5.3.5 Lack of Milestones

During the 2020 Season, **Software Scouting** did not have "soft" deadlines, or deadlines of any kind. Consequently, we had difficulty completing all necessary tasks before competition. Our only deadline at the start of the season was LAN, but the feature cutoff we set for it was not early enough and was not enforced. The absence of goals or deadlines within the season led to a lack of urgency and decreased efficiency, as well as causing a late end to end system and therefore less effective testing (see Testing and Review Process Section). Having an end to end system would also have guided us in setting a productive pace by making future steps more clear and attainable, so the lack of an end to end system also became somewhat of a self perpetuating issue.

5.3.6 Insufficient User Feedback

Another key issue in this year's **Scouting System** was that we did not receive enough feedback from our users. Part of this problem was that we did not know who all of our secondary users were until after LAN. Additionally, our

visualization apps were less effective because we did not communicate enough about what the visualization UI (especially the **Viewer** app) was expected to look like, resulting in user dissatisfaction.

One specific issue we ran into at LAN was the scale in the subjective mode of the **Pit Collection** app. In the subjective section of the **Pit Collection** app, we used a different scale than the **subjective pit scouts** wanted: ranging from 1-4 instead of the expected 1-10. If we had collected more user feedback about this app before competition, we would have been able to completely avoid the confusion and potentially less accurate data.

Probably the worst consequence of insufficient user feedback was the lack of app distribution to our users. We were unsure about who all of our secondary users were, resulting in some of our users not getting the **Viewer** app. Although its most important users had the app, the secondary users were not given the app. If we had collected more feedback about the **Viewer** app, we would have been able to distribute the app to everyone who needed it.

Chapter 6

Future Improvements

6.1 Overview

In **Software Scouting**, we strive to continue improving our system and subteam in every way possible. Although the 2020 season was cut short, we were on track to add significant functionality to our system by the end of the season. We plan to continue working on additional features of our system this offseason.

6.2 Offseason Education Revamp

For the 2020 offseason, we plan to drastically improve our offseason training. We will prioritize the topics that we teach new members during Software General and during **Software Scouting** specific training in order to better prepare them for the fast-paced environment of the subteam. This includes focusing less on specific programming structures and concepts, and having a greater focus on problem solving and research skills.

6.3 Improved Testing Procedures

Testing not only aids all of software but is a major facet to our scouting success, as it allows us to catch bugs in the system that would otherwise go unnoticed. Having an end to end system at all times is crucial for testing the **Scouting System** and getting helpful feedback from users. Furthermore, not having a fully functioning end to end system led to reliability issues at LAN and complicated development for apps as testing became significantly harder.

One way to improve testing is to create automated tests, which run whenever an app or the **server** is updated. Having both human and automated testing would have caught some bugs that made their way into the **Scouting System**. One particular procedure that may be used in the future is Test Driven Development, where tests are prepared before new code is written. This gives more structure to the process of each code change and allows programmers to have a better idea of what code should do before it is written.

Although we had field tests in the 2020 season, they did not test the whole system. Ideally, several full system tests should have been completed. Full system tests would force the subteam to consistently have an end to end system because the tests rely on the existence of such a system. In addition, a full system test is the only reliable way to regularly test all interactions between parts of the **Scouting System**.

6.4 Improved Software Features

While improving offseason education is our first priority for the 2020 offseason, we also plan on implementing several features to improve the **Scouting System** for future seasons. This year, we focused heavily on writing code that could be reused in future years wherever possible. This was usually successful, however there are still some areas where the code and **schema** could be further generalized and made game-agnostic. One example that affected us even during this season was when we decided to switch **subjective scouts** from collecting driver speed to driver rendezvous agility. It took us significantly longer than it could have because the code assumed that the metric, otherwise calculated the same, was called driver speed.

We also plan to take TBA alliance data into account for **consolidation**. We had planned to have this completed before the Sacramento Regional so that it would be tested during competition before the Houston Championships. However, with the suspension of the season, it turned into something that would get completed during the offseason. **consolidation** using TBA data will improve data accuracy by ensuring that team values add up to be consistent with the final match score.

Another feature that likely would have been completed by Sacramento Regional was database **schema** validation and enforcement. This would help increase reliability by catching instances where incorrectly typed data was put into the database.

Additionally, we would like to add graphs for the **Viewer** app, to help users easily visualize collected data. This element was in the process of being implemented during the season and would most likely have been completed by Sacramento Regional.

Chapter 7

Conclusion

7.1 Overall

The Citrus Circuits **Scouting System** has evolved significantly since 2013 and was the most effective and reliable it has ever been in the 2019-2020 season. This is due to our preseason redevelopments, as well as the new processes, procedures, and debriefs implemented in the **Software Scouting** subteam.

7.2 Utilization of Human Resources

To create our electronic **scouting system**, the **Software Scouting** subteam requires many resources, primarily the time and commitment of many dedicated software developers. During the 2020 season, the **Software Scouting** subteam had 24 members, with 13 working on the **server**, and 11 on the collection and visualization apps. This was significantly more members compared to last year with 15 total developers.

Our season is split into two sections: the offseason and the build/competition season.

- Offseason (August–December):
 - Regular meetings two nights a week (5 hours total)
 - Used for training new and veteran members, as well as discovering new ways to improve our system for the upcoming season
 - Discussed and implemented new **Scouting System** structure for system redevelopment
- Build and competition season (January–April):
 - Four mandatory meetings a week (21 hours total)
 - Used for programming and testing the **Scouting System**

In order to complete our **Scouting System** on time, we often end up working during extra team meetings and outside of meetings to meet deadlines and complete all required work. Scouts also spend several hours before each competition training on their respective apps.

7.3 Resources

7.3.1 2020 Scouting System Software

The 2020 software is available at <https://github.com/frc1678> For individual links:

- **Match Collection** app: <https://github.com/frc1678/match-collection-public/tree/2020>
- **Pit Collection** app: <https://github.com/frc1678/pit-collection-public/tree/2020>
- **Viewer** app: <https://github.com/frc1678/viewer-public/tree/2020>
- **Picklist Editor** code: <https://github.com/frc1678/picklist-editor-public/tree/2020>
- **Server** code: <https://github.com/frc1678/server-public/tree/2020>
- **Schema** files: <https://github.com/frc1678/schema-public/tree/2020>

7.3.2 Agile Software Development

Information for Agile software development can be found here: <https://agilemanifesto.org>

7.3.3 Past Whitepapers

For past whitepapers, please reference our website: <https://www.citruscircuits.org/scouting.html>

7.3.4 The Blue Alliance

Excellent website for viewing data on FRC. Provides an API for easily pulling data from the site for use in custom scouting system software.

<https://www.thebluealliance.com>

7.3.5 Fall Workshops

Citrus Circuits students and mentors hosted several workshops/seminars focused on different aspects of our team.

<https://www.citruscircuits.org/fallworkshops.html>

The most recent workshop on Scouting System Development can be found here:

https://www.youtube.com/watch?v=LYLG_Vdm-QQ

7.3.6 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>

7.3.7 Overview and Analysis of FIRST Stats

Comprehensive overview of mathematical analysis of FRC and FTC teams. Covers concepts ranging from simple analysis such as OPR, DPR, and CCWM to more complex methods such as WMPR, EPR, and MMSE techniques. Some knowledge of linear algebra recommended.

<https://www.chiefdelphi.com/t/overview-and-analysis-of-first-stats/144569>

7.3.8 Contact Us

We are interested in helping 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 softwarescouting@citruscircuits.org.

Glossary

ADB

ADB stands for Android Debug Bridge, a tool used to help with debugging Android apps. In the **Scouting System**, it is used to pull data from all apps to **Server**. 17, 19

Codebase

Refers to a specific software component of the 1678 **Scouting System** (e.g. **Server**, **Match Collection**). 10–13

(To) Consolidate

For each datapoint gathered from the objective **Match Collection**, there are three **unconsolidated** values, one from each **objective scout**. The consolidation process takes these values and uses them to find the single most accurate value. 19, 20, 24–27, 31

Crew

Software Scouting is divided into two “crews”: Back-end and Front-end. Front-end is responsible for developing software that users must interact with. Back-end is responsible for developing the **Server** which connects all of the other pieces of software. 5, 6, 10, 11, 29

Match Collection

The Match Collection app gathers data about robots or alliances in a match. Match Collection has two “modes”: objective and subjective. In objective mode, **objective scouts** collect objective data about a team’s robot in a match (e.g. number of high goals a robot scored in a match). In subjective mode, **subjective scouts** collect subjective data about the three robots in an alliance. The objective data is collected by three **objective scout** for each robot in a match, while subjective data is collected by a **subjective scout** for all three robots in an alliance. 4, 5, 12, 13, 16–19, 21, 28, 33, 35, 36

Objective Pit Scout

Objective pit scouts collect objective data about a team’s robot (e.g. number of motors, and drivetrain type). 5, 12, 17, 35

Objective Scout

A 1678 member that collects objective data about the performance of a single robot in a match (e.g. the number of high goals that a robot scored in a match). At every match in a competition ideally there are 18 objective scouts scouting a match, with 3 objective scouts assigned to each robot. 5–7, 12, 13, 18–20, 22, 24–27, 35

OPR

A team’s OPR is an estimate of the average points a team contributes to each alliance. 20

Picklist Draft

The process during which **strategists** modify the picklist from the original computer-sorted list of teams. This mainly takes place in a discussion on the night of the first day of qualifications. 20, 37

Picklist Editor

The spreadsheet that is used during the Picklist Draft. It’s data is imported by a CSV file that is generated by the **Server**. 4, 5, 14, 18, 22, 33, 36

Pit Collection

The Pit Collection app gathers objective and subjective data about robots in the pit. The users of the objective mode of the Pit Collection app are called **objective pit scouts**, and the users of the subjective mode of the Pit Collection app are called **subjective pit scouts**. 4, 5, 12, 17, 21, 28, 30, 33, 36

Schema

Schema outlines how our data is structured. For example, **Match Collection** Schema details what characters stand for the specific actions that a robot can do within a match of Infinite Recharge so that the data can be compressed. 11, 18, 19, 28, 31–33

Scout Training

A session for scouts to practice the data collection process, usually using match videos from a recent competition. 12, 27

Scouting Leadership Team

The Scouting Leadership Team is a group of 6 veteran members within **Software Scouting** that helps with the management and leadership of the subteam. 6, 10, 11

Scouting System

The system that 1678 uses to collect, process, and visualize data in order to aid in match strategy and picklist creation at competition. There are two apps that collect data: **Pit Collection** and **Match Collection**. The data from these apps is processed by **Server**, then displayed on the **Viewer** app and the **Picklist Editor**. 4, 5, 7, 8, 11–13, 18, 21, 22, 28, 29, 31, 33, 35

Server

Server makes up the totality of the **software scouting** back end. It handles decompression, consolidation, and calculations. 4, 5, 11, 14, 17–22, 31, 33, 35, 36

Server Computer

A specific set of computers that the **Server** is designated to run on. 17, 19

Software Robot

A subteam of 1678 Citrus Circuits. The purpose of Software Robot is to create autonomous and teleoperated systems that work consistently and effectively to maximize the potential of mechanical systems. 13, 28

Software Scouting

A subteam of 1678 Citrus Circuits. The purpose of Software Scouting is to create the most valuable software to aid in 1678's competition picklist and match strategy creation. 4–6, 10–13, 28, 29, 31, 33, 35, 36

Software Scouting Developers

Members of **Software Scouting**. 13

Strategist

A team member, either a student or a mentor, who contributes to the match strategy and picklist of 1678. 14, 15, 18, 20–22, 35, 37

Subjective Pit Scout

Subjective pit scouts rate the ease of buddy climbing with the 1678 robot using the subjective mode of the **Pit Collection** app. 5, 12, 17, 30, 35

Subjective Scout

A 1678 member that collects subjective data about the performance of three teams robots in a match (e.g. the robot's speed). There are two subjective scouts scouting in a match, one for each alliance. 5–7, 12, 13, 16, 18, 20, 22, 31, 35

TIM

Data about a single robot in a single match. Can be objective or subjective. 19, 26

Unconsolidated Data

Objective match data from objective scouts before consolidation. 19, 26, 35

Video System

The system used to collect match videos for **strategists** to refer to during the **picklist draft** night. 12, 13, 21–23, 37

Video System Operator

Operate the **video system** in order to record qualification and playoff matches. Matches that 1678 played in are transferred and viewed by the 1678 drive team in order to review their performance. 12

Viewer

An Android app used by **strategists** to view data. 4, 5, 15, 18, 21, 28, 30, 32, 33, 36