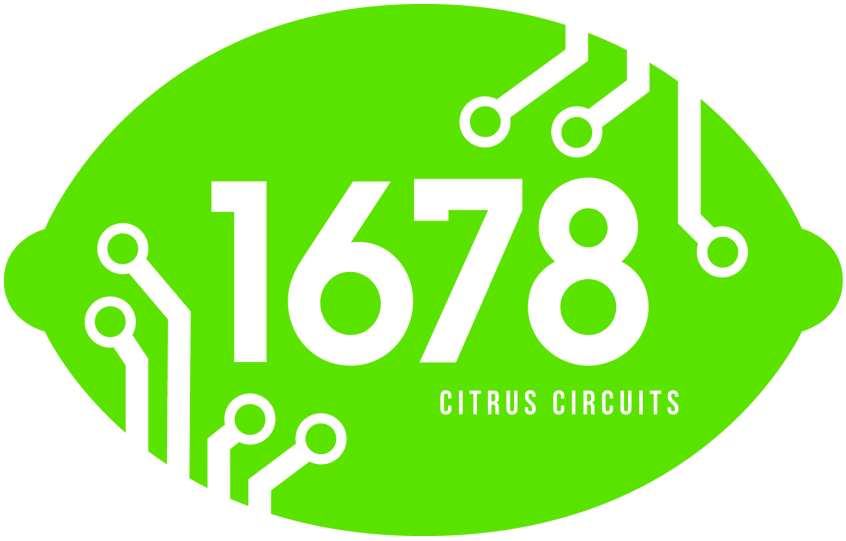


ELECTRONIC SCOUTING SYSTEM

DONALD PINCKNEY

SEASON OF 2012-2013

Introduction

Original Goal of the system

The goal of Citrus Circuits's scouting system is to allow for seamless input of data by scouts in the stands into a central database, which we can later use to very quickly calculate alliance selection lists. With our older pencil and paper system from last year, we found ourselves scrambling to effectively use all the data collected by our scouts.

Our new system effectively achieved its above stated original goal at the Central Valley Regional, allowing us to pick team 840 and 295 and win the regional as 7th seed Alliance Captain. At the Central Valley Regional, the system aggregated the data, and let us quickly create our alliance selection lists in about 10 minutes. But in our two following competitions, the Sacramento Regional and Championships, we took the system a step further by providing nearly real-time data and stats to mobile devices handled by our coaches. This allowed our coaches and drive team members to far more effectively strategize for matches and browse data for teams, along with the primary purpose of alliance selection.

Figure 1 Team members scouting with the app

Overview

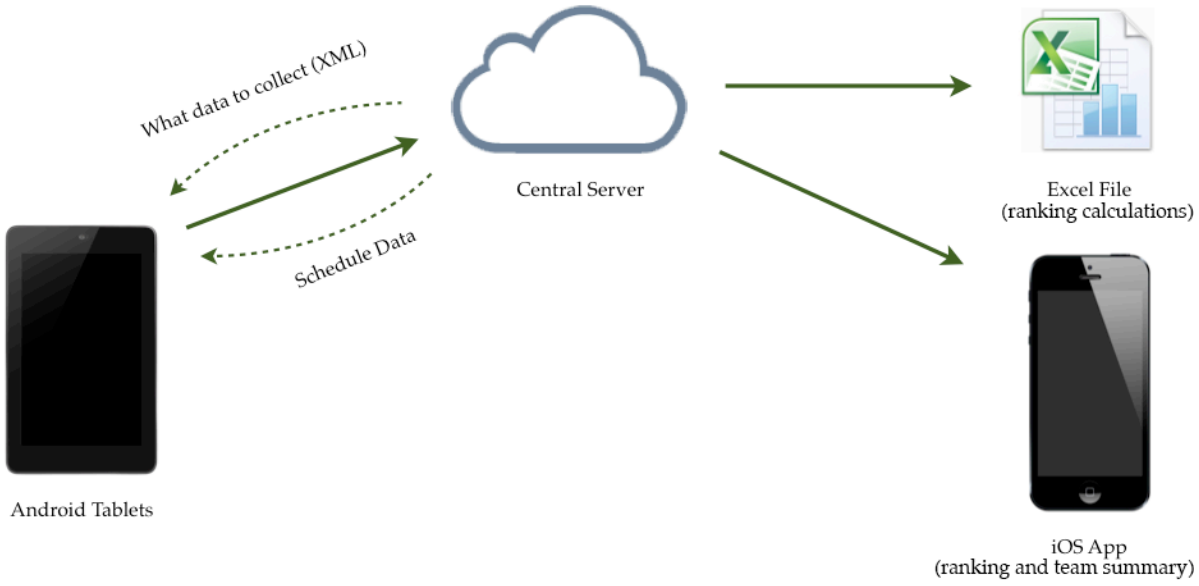
Currently our scouting system consists of four main parts:

- An **Android app** running on eight tablets, which scouts sitting in the stands input data into. The tablets run two programs, one for quantitative measures, and a second for strategic scouting.
- A **server**, which receives data from the Android app, stores it, and then outputs it to an Excel file.
- An **iOS app** which would receive nearly real-time data from the server and display it to our coach and team captain in the pits. This app has two primary functions: to substitute for previous calculations done using Excel in earlier iterations of the system, and to provide our coach and captain hard data to use to strategize in qualification matches.

So far, the lines of communication are: Android tablets send data to server, server sends data to Excel file, and server sends data to iOS app.

However, there are also additional lines of communication not directly related to the collected data. To make the system flexible for future competitions, we made it easy to change what data is collected on each robot and across matches. To achieve this, the server communicates a XML file to each scouting tablet, which dictates what data to collect. In addition, the server sends qualification match schedule data to each tablet.

To clarify, below is a diagram of the system, with solid arrows representing robot data communication, and dashed arrows representing other communication from the above paragraph.



Implementation Details

Internet Problem

Initially at the Central Valley Regional we connected the tablets sequentially to a Wi-Fi network when we needed to upload data. This allowed us to create rankings lists for alliance selection, but this did not allow for real-time data. To provide real-time data to the iOS app requires the Android tablets in the stands to have a constant Internet connection. However, rule T04 in the FRC Game Manual states that "Teams are not allowed to set up their own 802.11a/b/g/n/ac (2.4GHz or 5GHz) wireless communication (e.g. access points or ad-hoc networks) in the venue."

This rule made it quite difficult to provide a constant Internet connection to all eight Android tablets, as this rule specifically outlaws a simple Wi-Fi hotspot. We were not able to finance eight 3G tablets and data plans, and we were committed to building a fully legal system, so we had to get creative.

At the Sacramento Regional we used a single 3G dongle with a router and a network switch to create a purely wired hotspot. We then used Ethernet to USB adapters to provide all eight tablets with a wired USB Internet connection.

Figure 2 3G dongle, router, and part of the case

We also used a lead-acid robot battery to provide power to the router, switch, and simultaneously charge all eight tablets. However, the Ethernet cables and the power cables to the tablets were too cumbersome, and the Ethernet to USB adapters did not work consistently. We were unable to deliver real-time data to the iOS app, which was our objective.

In preparation for the World Championship, we completely redesigned out Internet setup. As before, we used a 3G dongle with a router to provide a wired Internet connection. This Internet connec-

tion connected to only a single Raspberry Pi (http://www.raspberrypi.org/). Then all the tablets connected to the Raspberry Pi via USB cables and USB hubs. However, this did not provide the tablets with a direct source of an Internet connection. We programmed the Raspberry Pi to receive data from the tablets purely over USB, and then upload it to the server. Specifically, C/C++ code on the Raspberry Pi implemented the Android Open Accessory Protocol (http://source.android.com/tech/accessories/aoap/aoa2.html) using **libusb** and then uploaded that data to the server. The USB cables in this setup were far more reliable than the Ethernet to USB adapters, and were far easier to manage. In addition, a single USB cable to each tablet carried both data and power, unlike at the Sacramento Regional. We worked to create a simple and elegant result: we packaged the Raspberry Pi, battery, router, and USB hubs into a single aluminum case. This made it easy and comfortable to carry into the arena.

Nevertheless, this system was not without its flaws: We did not find out until we were in competition that the Raspberry Pi could only handle six or seven simultaneous USB connections (despite having eight physical USB ports), and we had electrical failures with one of the USB hubs. However, the system did still work well enough to provide the data necessary for the iOS app on a fairly instantaneous basis. It was sufficiently reliable that our driveteam dubbed the iOS app the "BS detector"!

Android App: Native App, Programmed in Java

Key Functions:

Automatic scheduling: The Android app receives the schedule data from the server, and using that automatically keeps track of the schedule. It tells the scout using the tablet which match it currently is, and what robot to watch. The schedule is transmitted as **JSON** data, and only needs to be transferred to the tablets once, often the night before qualification matches. Thus, this data is not transmitted through the Raspberry Pi discussed above. The schedule is originally provided to the server by automatically parsing the HTML of the online FIRST match schedule, or by manual entry.

Easily configurable data collection: The Android app receives a **plist** file (a glorified XML file) from the server, which dictates what data should be collected. The app dynamically generates a different UI based on the plist file, without any changes to the source code. As with the schedule data, this only needs to be transmitted to the tablets once before competition. The end result is that we can update the Android app for next year's competition in a few minutes, just by changing a single file. To help illustrate, to the right is a sample from our data description plist file. Note the **type** value: this determines what kind of UI control will appear in the app for a given piece of data. Specifically "int" will produce a +/- stepper (for counting Frisbee shots), "BOOL" will produce a switch (can a robot pickup Frisbees?), and "radio" will produce a radio group (0, 10, 20, or 30 pyramid points).

```
<key>auto</key>
<dict>
    <key>highShots</key>
    <dict>
        <key>label</key>
        <string>High Shots</string>
        <key>type</key>
        <string>int</string>
    </dict>

    <key>medShots</key>
    <dict>
        <key>label</key>
        <string>Med Shots</string>
        <key>type</key>
        <string>int</string>
    </dict>

    <key>lowShots</key>
    <dict>
        <key>label</key>
        <string>Low Shots</string>
        <key>type</key>
        <string>int</string>
    </dict>
</dict>
```

Figure 3 Sample data description

Uploading: As discussed above in the Internet Problem section, we used USB cables to transmit data from the tablets to the Raspberry Pi, which then uploaded the data to the server. The format of how the data is uploaded is very simplistic: it is simply a very long URL, with parameters for the data. Below is an example URL.

http://example.com/upload.php?match_teamNumber=1678&match_matchNumber=6&match_auto_highShots=5

password, **match_teamNumber**, and **match_matchNumber** are required parameters, and the parameters following vary, depending on what is the in data description plist (see above). The **key** for each data piece in the plist matches with the parameter name in the URL. If we had a data piece with the key "climbPoints" in the teleop section, that parameter name would be: **match_teleop_climbPoints**.

Each time a match completes, the Android app attempts to transfer the data to the Raspberry Pi via USB, and if that fails, we store the data locally on the tablet. Over time, a large batch of data that failed to upload may form, which one can later upload together via Wi-Fi, e.g., during lunch outside of the arena.

Server: Programmed in PHP

Key Functions:

Storing uploaded data: The server stores all the uploaded data in a MySQL database. There is a separate table for each team, and inside each table rows correspond to matches, and columns correspond to data pieces.

Storing match schedule: The match schedule is either entered by hand or by automatically parsing the HTML of the online FIRST match schedule, and is then stored in a single MySQL table.

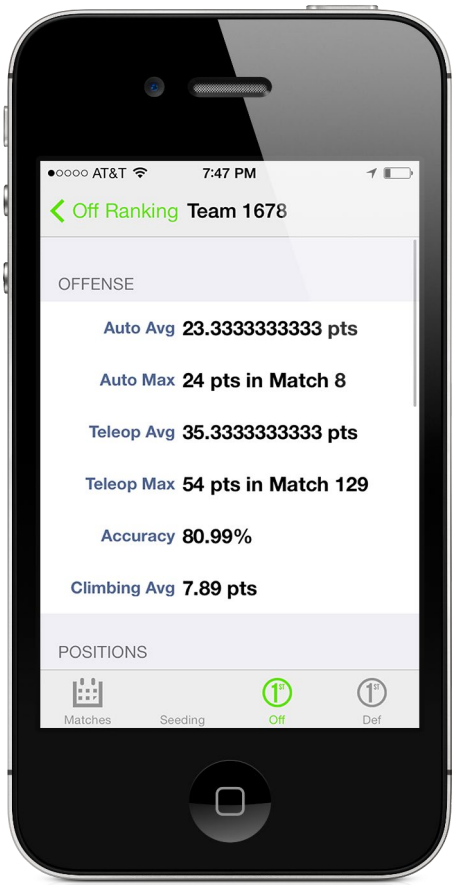


Figure 4 iOS app displaying our stats

Export data to an Excel file: The stored data can be exported and downloaded as an Excel file in the .xls format via a single click. The server uses the library PHPExcel (http://phpexcel.codeplex.com/) to quickly generate an Excel spreadsheet from the current data. The server also automatically sums all the data when it produces the Excel spreadsheet, as this is very useful for calculating ranks. Downloading the data as a spreadsheet is most useful for overnight or post-competition analysis.

Perform detailed, flexible calculations for the iOS app: The calculations and statistics that appear in the iOS app (see below) are all calculated on the server. Like the data description plist, we wanted the calculations to be extremely flexible. To maintain calculation speed and flexibility, while allowing for more complex calculations, we allow customization of the calculations via insertable PHP code on the admin webpage. This allows you to write fully-featured PHP code for the calculations, and modify it at competition simply by editing it on a webpage, no FTP access needed.

Admin webpage: The server also includes a simple but effective admin webpage. On the webpage, one can change the data description plist, the match schedule, and the PHP code for the calculations for the iOS app, all without having to connect via FTP. The admin webpage is also where the data can be downloaded as an Excel spreadsheet.

iOS App: Native App, Programmed in Objective-C

Key Functions:

Eliminates the need for FRC Spyder / Tracker app: The app consolidates scouting data, match schedule, and seeding all into one app.

Brief overview of other robots in upcoming qualification matches: The driveteam in the pits can quickly see what the best offensive and defensive robots are in upcoming qualification matches, and strategize with their alliance accordingly.

Provides more detailed calculations for each team: This is all calculated on the server, and has dynamically generating UI based on what the server calculates. Example calculations include autonomous average and maximum scores, teleop accuracy, and Frisbee pickup data.

Raspberry Pi Uploader: Programmed in C/C++

As discussed in the Internet Problem section, the Raspberry Pi simply receives data from the tablets via USB, and immediately uploads it. However, the implementation was far from simple to achieve. To communicate with our Android app over USB, we had to use **libusb** to implement the Android Open Accessory Protocol. Libusb was more low-level than we were used to, and the implementation of the Android Open Accessory Protocol on the tablets (out of our control) seems to exhibit bugs relating to garbage collection. To get started quickly with Android USB accessories, check out the link http://android.serverbox.ch/?p=262, which we based most of out libusb code off of.

Miscellaneous

Why Android for the tablets, and then iOS for the data receiving app? One reason: cost. It is not possible (as far as we know) to get eight $100 iPads in new condition. But, it is possible to get eight $80 Android tablets. We chose iOS for our platform for the data receiving app simply because iPhones rather than Androids happen to be available to our driveteam. We plan on implementing an Android version this coming season.

Future Goals

Improving the Raspberry Pi Uploader

The Raspberry Pi uploader was a complicated but clever solution to the Internet problem. However, it was less than perfect. We plan to keep the same system, but modify it to make it more robust. We plan to get more reliable USB hubs, and use two Raspberry Pi's rather than one, to solve the problem of a maximum of 6 USB connections per Raspberry Pi.

Robot Photos

Next year we would like to implement a robot photo database into our system. One team member would be dedicated to taking photos in the pits, probably with a smartphone, which would then be synced to the iOS app.

Visual Graphs

We think that we can do a lot more data processing with the iOS app. We look forward to implementing beautiful, visual graphs of robot performance over time.