Mechanical Engineer Resource Document

Yaroslav Tochinski

Software Required

- Autodesk 2018 or later
    - Search for student version on Autodesk
    - Used because previous teams used it, also free
- LabVIEW 2013 or later
    - **Must be 32-bit**
    - Need application builder
        - Available in LabVIEW Professional
        - The RIT computers can't build executables to my knowledge
    - You may need to get creative to get this software
- VI Package Manager
    - https://vipm.jki.net/
- GFITSIO
    - https://fits.gsfc.nasa.gov/fits_libraries.html#matlab_nz
        - This link shows all the software available
    - Only available in 32-bit, hence restriction on LabVIEW
    - Callisto uses FITS files
- JSON API
    - https://lavag.org/files/file/216-json-labview/
- Websockets API
    - https://forums.ni.com/t5/Example-Program-Drafts/LabVIEW-WebSockets-Library/ta-p/3491490
    - This one was a little janky. I had to create ping and pong vis, which is puzzling, since this isn't an uncommon part of the protocol at all. If your connection suddenly drops before a minute elapses, investigate ping and pong.
- Node.JS
    - For demo server
- GIT
    - For demo server

Contact info

- Yaroslav Tochinski
    - Mechanical Engineer
    - 610-825-2697
    - yaroslav@tochinski.com
- Brandon McDonnell
    - Computer Engineer

    - 267-328-4411
- David Fediaczko
    - Electrical Engineer (Hardware)
    - 484-560-3143
- Kyle Tevis
    - 484-942-5795
    - Electrical Engineer (RF)

100% of the team is currently working full time. We would be willing to answer any questions you have. Dave and Kyle are based in Rochester still. Do not try to contact us via RIT email however, we will almost certainly not check it.

Please contact me ASAP, I will share the google drive with you. Our teams public SVN *should* be up to date, but there's lots of stuff not there, and you need to ask for it. Our google drive contains all our data, and all the previous groups data. I am the owner, and I really don't use the student drive anymore, so it's not going to get deleted unless RIT deletes alumni drives suddenly.

The biggest hurdle we had was too much work, and too few people, mostly because of new action items that kept coming up. This is why documentation is lacking a little, we had a lot of other things to do. To make up for this we are offering our contact, so you can ask us directly. In this document, I tried to highlight and explain the things that I expect you will pull your hair out trying to figure out.

Recommendations

Please contact me ASAP so I can share the google drive with you. It will benefit you to have all the files ASAP. MSD 1 is very intensive in beginning in terms of classroom deliverables. Now, in terms of engineering education, all the material is very beneficial. However, I now wish we had spent less time on the MSD class stuff and more time familiarizing ourselves with the system.

Here are a few things you should start off with:

Team Leader: Get access to the Senior Design Space on the 3rd floor. If there is a superior senior space, feel free to use that. It can get a little cramped up there, and you will have a lot of stuff. Also, try to get a key to the observatory. The garage is great for storage. One thing I pushed for but was unable to get is a 3 in OD pole installed at the observatory. This will allow you to do testing at RIT. If you start immediately, you might be able to get it installed by spring.

Mechanical: Familiarize yourself with the CAD, fix it up a little if necessary. Learn Inventor if need be. One task I wanted to complete, but couldn't is a complete mechanical BOM, especially for all the fasteners. This would be a good objective to familiarize yourself with the system.

LabVIEW developer: If you don't know LabVIEW, learn LabVIEW. Alternately, if working the GUI proves too difficult, feel free to scrap it and build a new one. You won't be able to run the GUI until you have a demo server. If you want, you can try to create a debug mode.

Computer Engineer: Contact Brandon and try to get up to speed on the Suntracker server code. Brandon had to rewrite large chunks of the previous team's code. Try to get a demo server set up so the LV dev can debug his GUI. The only thing he did not write is the PyEphem (Python) module, which is responsible for returning the coordinates of the sun. The author of that code is David. This had to be done because Radio Eyes does not have a headless mode, and the developer is unresponsive to our requests.

Electrical: We had an electrical fault at final installation around finals week. Contact Dave to learn what to fix, if he did not fix it already.

RF: Contact Kyle and talk to Marty to learn what needs to be done here. I'm not really clear.

LabVIEW

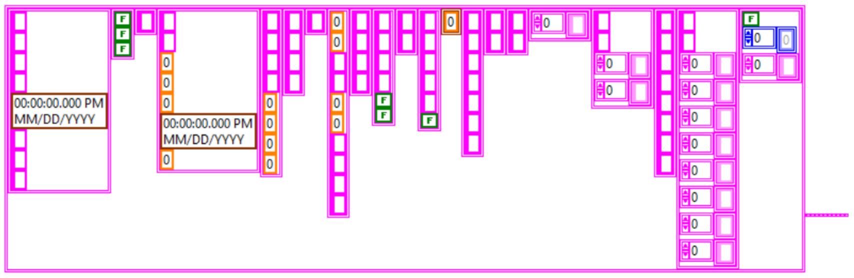
It appears that LabVIEW is not really taught outside the Mechanical Engineering department. So, the Mechanical Engineer is forced to be a software developer. You can also try to have the computer engineer learn LabVIEW, but I expect they will have enough work to do outside of LabVIEW. For the LabVIEW developer, it is recommended to download the LabVIEW tutorials from google drive 20 Resources and do some of them. The GUI kind of blew up in complexity. I do not recommend using ctrl+u to organize it, it is currently organized in somewhat functional groups. In order to learn it, definitely use the developer version, and save a copy of states often.

The SRT does not require the GUI to function. The GUIs function is to receive JSON commands and to display them meaningfully, and to send commands to the unit.

Now the backend of the GUI looks like a rat's nest of parallel loops, because it is. The most important functions are JSON IN and JSON OUT. These are named from the reference point of the SRT; so JSON IN sends messages into the SRT, and JSON OUT takes them out of the SRT.

The JSON protocol is defined in the suntracker driver protocol. For the purposes of the LV developer, it is a very particularly defined string. Warning: In the protocol, the CE sometimes used the wrong {} and " characters for the examples. If the VI appears to stop sending messages, check to make sure the right characters are being used, and all brackets are closed.

Now, how I handled JSON objects is converted them to a LV bundle 2 levels deep. These guys initialize the bundle:



Whenever you want to read/write something, you just use bundle/unbundle by name. In the developer GUI, this is very obvious, as this is displayed in raw form. Only JSON OUT writes to it, as this object stores the status of the SRT. The JSON IN is handled differently. The cluster really doesn't do much other than allow for a complicated case structure, and just generates a formatted string that is written to "Message to be Sent". If the "write" Boolean is enabled, then the message is sent.

The GUI actually communicates with the server via the websockets protocol. The websockets API used was free, but not that great. The critical piece missing was the pong opcode. The server periodically pings the GUI. If the GUI does not pong back, the server terminates the connection. If you see this behavior, check to make sure pong is triggering. Websockets read and write should be included in the API, pong.vi uses some of the subVIs from the library.

The GUI can handle being disconnected and will look for a reconnection opportunity (we have seen this happen). It will also disconnect and reconnect if it finds the connection times out (we have never seen this happen, but I have simulated it).

There is no real good debug mode for the GUI at present, you really need a demo server. The most current demo server was never compiled. You can look at StartSuntracker.bat to see what I referenced, but in full disclosure, I don't fully understand how the CE (Brandon) set this up to work, but I know node.js was required as well as git. You won't be able to run this too much until the demo server is set up.

There is also a subVI called fitsviewer. This was a separate independent vi and was made way before the rest of the GUI. You can mess around with it, and compare it output to the "official" rappviewer, or the other FITS programs. I did not like any of them to be honest.

The "bare" in the name refers to the missing component: the 3d waterfall plot. I tried my best to get this guy to work without crashing LabVIEW, but it really does not behave properly at all. It might be fixed in a later version, but I gave up and gutted it as it was too risky to include.

The CAD

Please use the latest version of Autodesk Inventor. I think you will need at least 2018 SP2, as that is the last version I made edits with.

Your go-to file is going to be Final_Assembly.asm. I tried to clean up the CAD and fix dimensions as best as I could. The assembly is also dynamic, so it you can grab it and actuate it. No driven components though, but you can add them if you want.

Not all dimensions in the real telescope turned out the same as in the model. The most glaring omission is that the Motor in real life does not fit over the pipe and requires shims. This is not reflected in the model, and despite being led to believe this problem was fixed, it was discovered it persists.

There are a few other models, like the pc internals model, and the observatory model, these are much less accurate. I could never find a CAD model of our MOBO, therefore there is a random one off GRABCAD. I never was able to get accurate dimensions of the observatory, I had to extrapolate based off a single stud measurement and pictures I had.

Full disclosure: I never really learned good 3D CAD modeling practice until after I left RIT. Looking back at the models, there’s definitely some things I could have done better.

None of the models have any material properties associated with them that I know of. You will have to go back and assign properties if you need weight information.