Debugging

Everyone’s code has bugs. Part of learning to program is learning how to fix them. This covers some general strategies of how to go about debugging your code.

"IT WENT AWAY"

This is a common statement when debugging code. The error just went away. It must be fine now. But that isn’t always true. In a deterministic space, code doesn’t just randomly get fixed. Something changed between runs. Keeping track of what types of things could have changed is a good mind set when you start debugging.

FIXING THEM BEFORE THEY HAPPEN

The first part of debugging is to fix errors before they happen. You are never going to be perfect but some methods will help make things better.

Simplify your code

Don't make things more complicated than they need to be. Don't do something just because it is fancy or cool. If simple works, do simple.

Show your code to other people

Have someone else look at your code. Having another set of eyes on your code can help catch bugs before your run into them, but it can also help you simplify things. If you have to explain it to another person, sometimes you catch errors yourself.

Comment your code

Every function should have a comment explaining what the function does. Every line of code that makes you pause and say "what does this do?" should have a comment. This gives you a chance to review your code and possibly simplify or catch errors, but it also helps make sure that if you run into errors, you have a clear record of what the code does so you can go back quickly.

Name things well

Name your code well. Name your functions and your variables in a way that makes them easy to understand. A variable named speed is easier to understand than a variable named s. A function should have a name that explains what happens in it. A class should have a name that explains what the code is in that class or its function in the rest of the system. Anything you can do do make your code more readable when you look back it will make your code easier to debug.

COMPILATION ERRORS

Compilation errors typically include a description and a line number, which makes the source relatively easy to track down in most cases.

But, not every compilation error makes sense. When you get compilation errors, what you know is that the compiler cannot understand that line. So it is possible that the compiler cannot understand anything

after that line. So if you want to fix a list of compiler errors that doesn't make sense, start at the first one and then try to compile again. Sometimes after the first one is fixed, the list will change.

TEST PLAN

You should have a plan of what you are going to test. You should know exactly what things to you are going to run before you go about trying to run them. This helps you better use the time you have as well and helps you better write and test code.

Know the expected results

You should know the expected result of every test you plan to run. It is also best if you know some types of errors you can run into and what they might mean about your code. You can't anticipate everything, but knowing what some things mean helps. This plan creation can also help you analyze your code before you run your tests.

Don't just run silly tests

Think about test coverage when you are creating your tests. One way to think about coverage is to make sure that every line of your code is run in at least one test. But a better way to think about it is that every case is run. All possible input sets, all possible cases you can come up with where the code or the robot would behave differently. You may need to push different buttons, place the robot physically differently or give different input on a dashboard. It can be surprisingly easy to accidentally run only tests that don't mean anything for the code you are running.

FIXING RUNTIME ERRORS

Make sure you have the right code

This sounds silly, but it is a common mistake, and you don't want to spend a really long time trying to figure out what is wrong just to realize you are running an old version of the code.

Check your inputs

Check what values you are being given. This can be from hardware or software. Make sure that the values are in the right range that you are expecting. That the API is expecting values in that range.

Rubber Duck Debugging

Talk through your code to someone, or something (like a rubber duck), line-by-line. Know what it should do and what each line means piece-by-piece. Talking through it can sometimes be all you need.

Understand the error message

Know what information you are getting. If you are getting an error message, what part of the code outputs that error message? What does that message indicate? Is there any other information you can get from the robot? Are there log messages, console messages? What are the CANTalon blink codes telling you? What about the temperature of the motors? Or the pressure of the pneumatics? Collect all the information you have to see what the status of the robot is and see what the tells you about the behavior you are seeing.

Debugger

In some environments, you may have access to a debugger. Debuggers are programs where you can run your code and stop it at a particular line and then step through line by line and see what the values of different variables are.

Divide and Conquer

Find a way to make the problem smaller. Look for a way to make the area of code that could possibly be an issue smaller. Try putting output lines in your code that can give your more information. This could be values of critical variables that you can't see otherwise or just making sure that code gets into certain functions or how many times it runs through a particular loop. Or come up with other tests that give you different variations and give you more information about exactly when it does or doesn't work.

Look for familiar bugs

Think about things that are common issues. Things that have broken before. Here are some common ones in general, but there may be more specific ones for your situation. Think about those as well

- Infinite loops: these are usually pretty easy to notice, and to fix.
- Off by one errors: These usually happen when you have something like loop that you meant to go from 1-10 and it goes from 0-9.
- Is the robot on?
- Is the radio/router connected?
- What gear are you in?
- CANTalon blink codes
- Did you upload the correct code?

Focus on recent changes

If you had working code, and now you don't, it is something that changed in that period of time. So check what most recently changed.

Take a break

If you've been staring at a problem too long, it can be hard to find an issue. Take a break. Taking a fresh look is sometimes the best thing you can do. When you come back to the problem, you are often able to think about it in different ways than when you've been staring at it for a while.