Applicant Questionnaire: Software Developer

Instructions

For some of our positions, we refer to portfolios or code samples when determining if interested people will fit with our team. We have found that some people's previous experience doesn't lend itself towards the creation of an online presence, so we have prepared a brief questionnaire.

Please answer the following questions to the best of your abilities. Good luck; may the force be with you!

Applicant Name:	Francisco Granda

Our main programming languages at Clearpath are Python and C++. What factors would you consider when choosing between these for a given project?

Important factors to consider when choosing a programming language:

- Speed: If speed is required to be optimized, then C++ is the best choice over Python.
- Application: The specific application for a given project is usually the most important factor as fields like machine learning, simulation and modelling are better handled in Python because of the existing libraries and tools. However, if the project includes embedded systems or external target hardware, C++ is the better choice.
- Background: In my experience, going through different projects in your career usually isolates you to one or more specific programming languages for a given time. For this reason, I also factor in the programming language I have been using the most in the past time to choose the language for a next project as it is easier to bridge between one application to another.

What is the most interesting problem you had to resolve through specialized tools, whether in-house or commercial? How were they used to explore and resolve the issues? Tell your debugging war story.
During my M.Eng program, I had to complete the software development for an unmanned aerial vehicle (UAV) in the form of a quadrotor. The objective was to complete the design of controllers, planners and multiple algorithms for computer vision and data association that would allow autonomous flight for the UAV. However, the interface with the quadrotor was a new problematic for me and specialized tools in the form of ROS and Gazebo were explored and used to complete simulations as well as real testing with the quadrotor. With the addition of the new tools, the new problematic became the interface of the original scripts containing the algorithms with the new simulation and testing environment. The debugging war that came next was focused on correctly aligning these interfaces and accomplishing total functionality with the Gazebo simulation environment as well as IRL testing through ROS and the quadrotor.

Team Project:

Please focus on a recent project you completed as part of a larger team.

Please describe your contributions to the project. What components were you personally responsible for, and what were their interfaces to the other components? How did those interfaces get defined?
My experience with aUToronto as the lead electrical engineer allowed me to work with a big multi-cultural team from UofT. My responsibilities included the design of a synchronization system for the sensor suite, organizing the team to handle hardware installation, and collaborating with the other departments to understand their requirements. The synchronization system was the most ambitious goal in the form of an FPGA based system where it interfaced with all the available sensors and the main ECU to reach sub-microsecond accuracy. Multiple interfaces were defined according to each sensor's electrical specifications. (Communication protocol/signals, voltage level and logic level compatibilities, physical connectors/wires, data transfer/message standards) The interface for each sensor also considered the synchronization tools available in each sensor's driver and software.

What development environment and programming language did you use? What costs and benefits did this choice have to the overall project outcome?
The synchronization project used the Vivado Design Suite produced by Xilinx along with Hardware Description Languages as Verilog and SystemVerilog. The choice of using an FPGA was heavily leveraged as the best option because it allowed to easily interface with the available sensors without the need of major additional hardware. By using a Xilinx FPGA board and Vivado, the debugging of HDL code was easy and testbenches were also implemented to observe early functionalities and bugs. The prototyping of the synchronization system also benefited from the simplicity of testing the board in a lab environment with oscilloscopes as well as the real sensors and equipment. Finally, this solution offered modularity and the possibility to continue adding features to the sync board in the future. The project was a success and was part of the system that won 2022 SAE's AutoDrive Challenge II.

What steps did you take to future-proof your code? Were there areas where you (and the team) chose specifically not to (or not to attempt to) future-proof? What was the reasoning in each case? Is there an example where this strategy paid off, or you got bitten?
Most of the code used in the development of the synchronization system was future-proofed as the focus was modularity and robustness. Each sensor interface was defined separately but interacted with the main scripts in the same way. If required, new sensors can be added to the scheme without the need to change the main framework. However, in the early development stages one specific component was a Phase Lock module that allowed the synchronization of external clock signals. This module initially was not future-proofed because it supported specific values of signal frequencies. The reason for this was the complexity of implementing a module that supported a range of values instead of singular frequencies. As new sensors were introduced to the project, new frequency values were required, and the module was revamped. These modifications pushed the deadlines of the project, and one non-crucial feature of the system was not implemented due to time constraints.

Individual Project

Please focus on a project completed individually (or mostly individually). Side projects, open source contributions, stuff for school, and stuff for previous employers are all valid.

What was the project? What was the motivation and what did you accomplish? How did it turn out differently from your plan going in?

My participation with the University of Toronto Aerospace Team allowed me to work on an attitude determination project for a cube satellite. My motivation followed my curiosity for space related projects and my desire to test my skills in a real satellite mission. Still in early stages of development, my project was related mostly to implementation and simulation of state estimation algorithms. I accomplished the modelling of dynamics and kinematics of the cube satellite as well as the implementation of an Extended Kalman Filter for attitude determination based on gyroscope telemetry.
When I started, I planned to follow a similar process to my experience in state estimation for robotics. However, the link of this application to the knowledge of orbital dynamics and satellite sensors added a level of complexity that I did not consider in my first plan. I was required to learn about and use multiple tools from the Aerospace Toolbox available in MATLAB and Simulink.

What is something that you know now that could have helped you with this project if you had known it at the time? Please focus on technical knowledge rather than personal development (eg, time management).

If I could give my past-self one piece of knowledge before starting the project, it would be to familiarize better with the different aerospace standards and conventions that are used in the industry. My experience with my master's program gave me an insight into these standards but my application was limited to specific settings. However, multiple standards and conventions can be defined for frames of reference, attitude/orientation, order of rotations and many more. The Aerospace toolbox from MATLAB used different conventions that made it difficult to use it with my own code and scripts. If I had known it at the time, I would have started by correctly identifying the standards and adapting accordingly since the beginning.

How did you perform system testing and verification?

The main framework of the project was completed in MATLAB mainly relying on my own code. The system testing and verification was mainly done in Simulink by creating a simulation environment where the Aerospace Toolbox could shine.
By replacing different sections of my models with the ones available from the toolbox, the system could be validated by the comparison of results to different scenarios.

Lightning Round

What is your favorite network protocol debugging tool and why? If you don't have one, what might you try out next? Where would using one be effective in troubleshooting a problem with an application or service?

I personally don’t have a favorite debugging tool. But I would start by looking at available tools based on visual reporting availability, cloud compatibility and of course pricing. These tools can be useful when you have many objects subscribed to multiple services. (i.e. nodes to a topic/service, components to a framework, sensors to a robot) A network analysis tool would be effective to isolate a specific node and perform troubleshooting.

Have you ever reverse engineered a protocol or some other aspect of a piece of unknown hardware or software? What was your process and what did you discover?

An important piece of hardware I needed to reverse engineer was a RS232 Splitter Box. My process was mainly oriented to hardware input/output identification and voltage logic. In this process I learned a lot about the COM hardware interface, the DC fan-out problematic and methodologies to avoid it.

Have you ever successfully evangelized a tool or practice to a team you were working on?

At aUToronto, as a master’s student I was tasked to teach the rest of my team, who were mostly undergrads, my approach to solve the synchronization problem. For this I planned various tutorials regarding FPGA’s and the whole process of prototyping with it. From HDL, to synthesis, implementation, and testing, I used simple examples like blinking an LED to go from coding to actually observing the functionality on the board.

What is your favorite robot, real or fictional, and in one sentence tell us why?

My favorite robot is NASA’s Perseverance rover simply because when it touched down on Mars, I was tearing up watching the livestream just in awe from what humans can do when collaborating with each other.

Data from Star Trek TNG is in close second though…