IMU Report

Intro

Over a four week internship, I was tasked with developing an automated IMU testing platform. This platform would be used for full system verification and testing of flagship GPS receivers. Through the use of two linear actuators, a pole with the receiver attached is able to be tilted in 3D space along the X and Y axis ±30° [1]. The embedded system utilizes an Arduino Mega microcontroller that communicates with the linear actuators, GPS receiver, and computer in real time via Serial connections. The system was built with the intention of succeeding manual tests done previously which were more inaccurate and time intensive. Upon my successful completion of the IMU tester, the verification process which previously took six months could now be completed in a matter of two weeks.

How testing works

To conduct a tilt test config data must first be loaded from the computer to the microcontroller. I accomplished this via a Python script where an .ini file contains data (trigonometric data, tilt angle instructions, baud rates, etc.) which is then parsed and sent via a Serial connection. Arduino code has a distinct quirk with code execution in that the main function called loop() continuously loops until either power is lost or new code is loaded onto the board. With this in mind, I utilized start characters and while loops to force the arduino to wait for computer input before taking action. On the arduino side of things, the config data is parsed again from the Serial line and set to the appropriate global variables[2]. Now that everything is loaded, the arduino waits for an API command[3] to be sent from the computer. A separate python script prompts the user to input an ID which is then sent to the arduino to execute. The main command which I will be focusing on is 'Start Test.' The command involves establishing a secondary Serial line[4] to the receiver and setting the necessary pins moving the linear actuators to tilt the receiver by a specified pitch and roll. Each position is considered a step and the program is designed to execute a series of actions for each step. For the full logic flow, refer to the flowchart[5] in the footer.

Design Intricacies

Writing the microcontroller code posed a number of challenges. The only option for tracking distance extended/retracted was to utilize the interrupt based Hall Sensors embedded into the linear actuators. On top of this, the pitch/roll instructions are recorded in degrees which have to then be translated into linear distance for the actuators. The translation is accomplished through some trigonometric functions. We assume the receiver starts perpendicular to the ground so the

---

[1] https://i.imgur.com/8G3GKmI.png
[2] Due to the limited memory and CPU resources, it makes more sense to use global vars for arduino
[3] The API commands vary from read config, extend/retracting custom lengths, read current roll/pitch, etc.
[4] This is to allow the arduino to communicate with the receiver via custom generated packets
[5] https://i.imgur.com/jkJ3BVk.png

hypotenuse (initial actuator length) is calculated via the Pythagorean theorem[6]. To get the final actuator length, we can sum the pitch/roll angle offset with ninety degrees and use the law of cosines to obtain the new hypotenuse. Now that the initial and final actuator length is known, all that remains is moving the actuators. I attached interrupts to the Hall Sensor pins that trigger whenever the pins change state[7]. Each pin has its own Interrupt Service Routine (ISR) which determines whether the actuator is currently extending or retracting and updates the actuator length accordingly. Communicating between the microcontroller and the receiver requires the use of DCOL packets; a standardized data type for Trimble receivers. These packets involve sending/reading specific characters along a Serial line along with a manually calculated checksum value.

Issues I ran into

- When testing the ISR, there was an issue where the actuators kept extending/retracting without ever reaching a break condition. After some debugging, it turns out that the actuator length variables weren't designated as volatile which was causing the issue
- I noticed that the extend/retract length compared to the expected length was wrong by a considerable factor; initially I thought it was related to the issue described above where variables were not updating in time however after some back and forth discussion with the motor manufacturer, they informed me that their datasheet had a typo
- When setting up the serial connections, some of the connections required the use of a null modem (Specifically the arduino-receiver) otherwise no data would be transmitted
- When dealing with reading/writing over serial lines, there were several times where there would be garbage data left on the Serial buffer that would interfere with future actions leading to system failure. I resolved this through a flush command that ensured each Serial buffer was completely empty at function termination.

[6] The two non-hypotenuse lengths are measured and specified in the config file before the test
[7] Low to High or High to Low