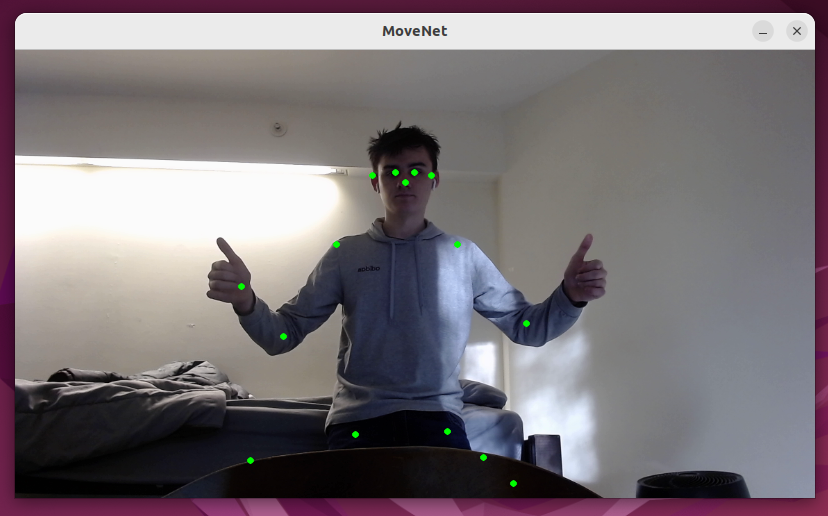
**Assignment #2** – Offloaded DNN Inference

**Part #1** – Offloaded DNN-Based Inference

This part was quite simple as it required solely taking a look at the instructions and following them such that the MoveNet code will compile and allow me to annotate my body accordingly. I did run into a couple of system issues with Ubuntu in that the Internet was briefly not working as well as several update errors. This made it difficult to fetch the project, and building it also took quite some time. However, I was eventually able to get everything working, and down below we can see an annotated image of my body.



**Part #2** – Offload the DNN-Based Inference

This part was quite simple since a lot of the information was already out there on the web and just had to be learned. I began with some quick introductions to networking which explained basic concepts like SSH tunneling, ports, etc. I then followed a three part YouTube guide on creating a web server in Rust by “Let’s Get Rusty.” They implemented a simple server at first as well as concurrency using a ThreadPool which I adapted to this project accordingly. I began by testing on my local virtual machine through the 127.0.0.1:8000 ip/port, but then moved on to testing between two VMs. In the first case, I ran into no issues in getting the annotations to display properly, however in communicating between the two VMs I found that with only a couple of threads, it was resulting in very choppy behavior. As a result, I had to increase the amount as well as implement a delay so that the code doesn’t automatically try and read the stream if it hasn’t finished processing and getting back and forth. Pinging in the two cases was significantly different where within the VM it would take ~0.02ms whereas between the two, it would take ~0.2ms. I made several noteworthy efforts for the mentioned objectives in the assignment: correctness, performance, elegance. For elegance, I split the entire code into three different crates that communicated with one another. You have the client and server sides where the client side would use the other crate to access and perform certain operations such as activating the camera, requesting annotations, setting up the server, etc. This would be very similar to how a client would do it in the real world utilizing some sort of an API. Then, the server facing code would call the TCP handler to perform the calculations over the network. As for performance and correctness, I had a lot of trouble with converting the float types to bytes, and to fix this I ended up utilizing an external library which worked perfectly, resulting correct and elegant annotations. More specifically, for performance, the ThreadPool and delay implementations came very much in handy in optimizing throughput since the connection was rather weak between my two VMs.

