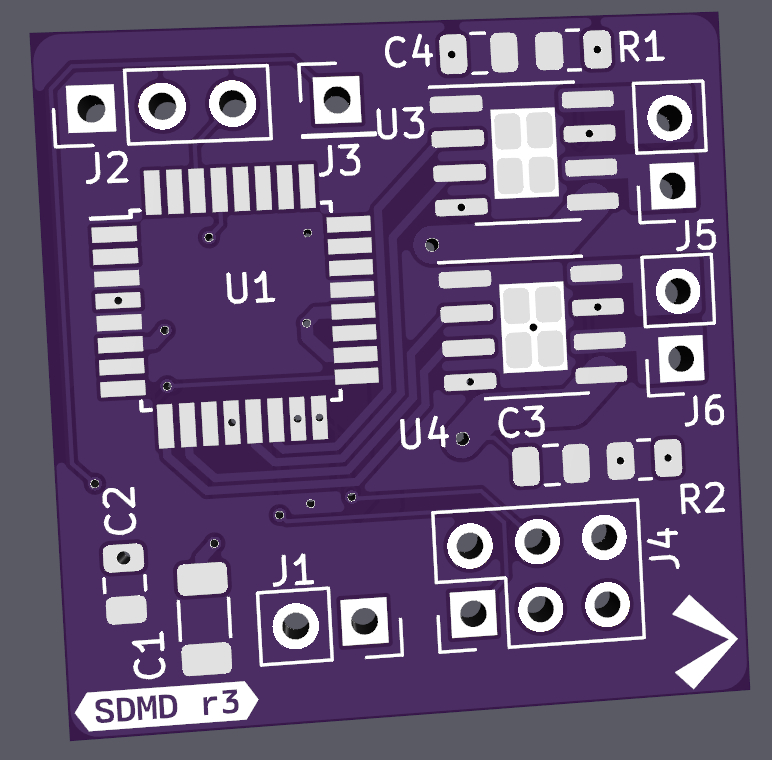
SDMD was a small project I did around when I built Reflection 2. I wanted my own motor controller as I was dissatisfied with current offerings. Through Hack Club’s Onboard program, I developed two prototypes, and then took over funding of revisions 3, 4, and now 5. SDMD is recently more of a software project.

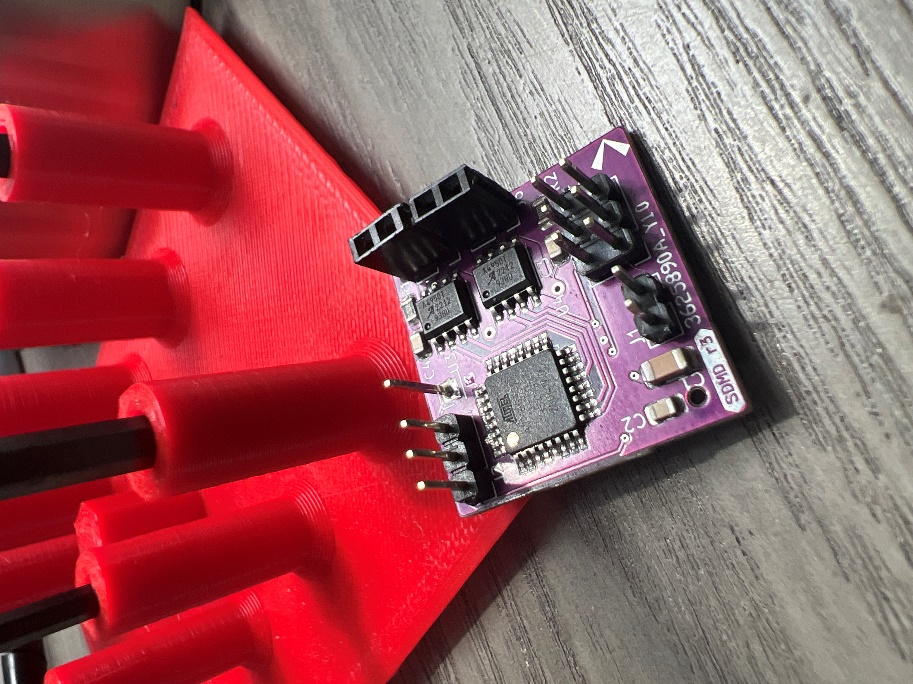
My design requirements were not very complicated. I wanted the device to be small, first and foremost, and cheaper than other available drivers. My size limit was 3x3cm, and my cost limit was $12. I decided to base the driver off the ATmega328 microcontroller because of availability, ease of use, and few required supporting components. I also settled on 0805 size SMT components because they are large enough to be easily hand soldered.

Revision 1 was about 2x3cm and used a Toshiba motor driver that I don’t remember the name of. Choosing motor drivers was difficult, I mostly searched through the motor driver section on Pololu until I found a chip I liked. The Toshiba driver was a dual H-bridge and could handle the current I wanted. I had this design made via Onboard, but it did not flash.

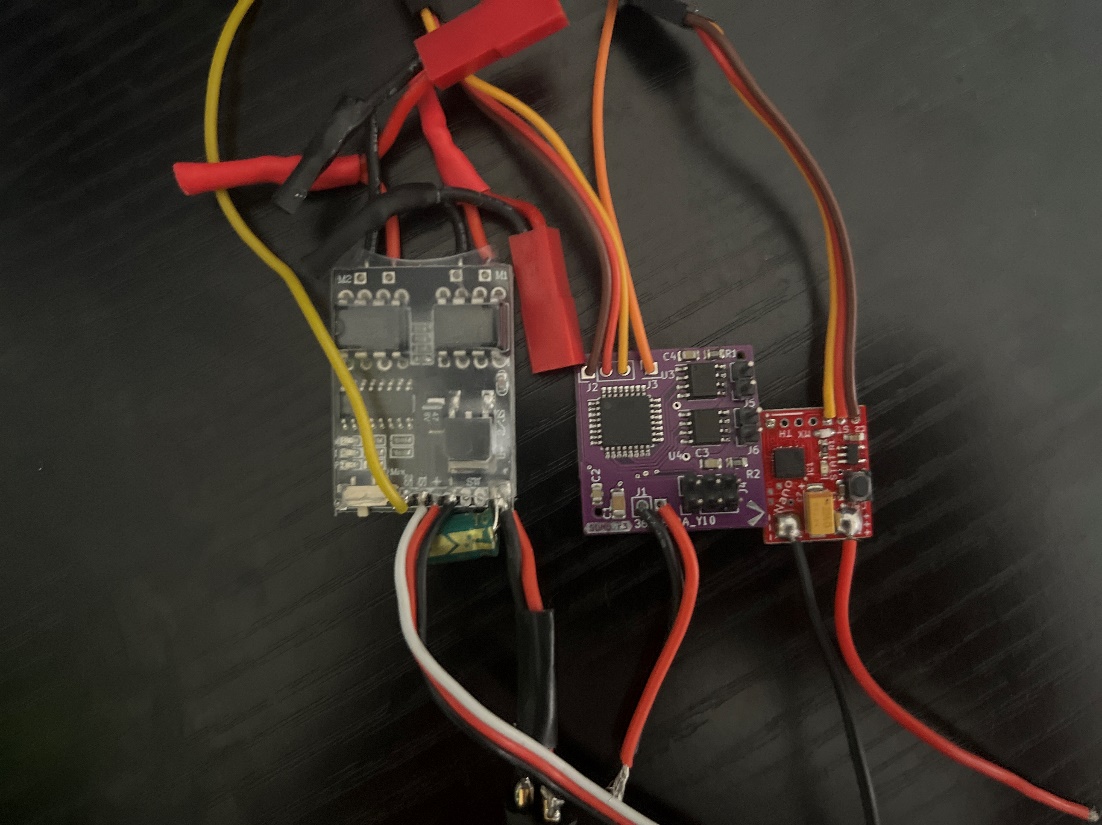
The second revision was much larger, bigger than 3x3cm, and very expensive, failing both of my basic design requirements, but it was more fully featured, with overcurrent protection. Soon after designing this, however, I found the A4950 motor driver IC.

Allegro Microsystems’ A4950 is an expensive but high-quality H-bridge driver capable of high voltage or high current usage. The specifications and the physical size seemed perfect for my usage, so I made revision 3 with them.





I particularly like this picture comparing the board between the large, expensive modules I had used before and the expensive Scorpion Nano motor driver.



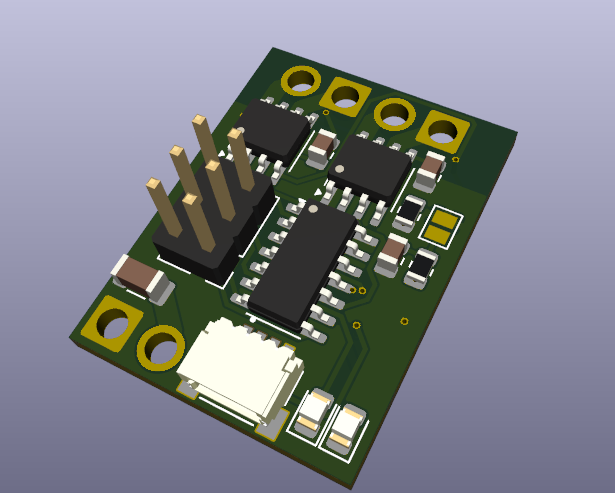
Hack Club’s Onboard grant funded the SMT assembly of the top of the board, and I assembled the bottom and through-hole components. This board also did not flash, until I found out that I had reversed the PICO and POCI pins on the ICSP header in all my schematics. After modifying my 6-pin adapter to the incorrect pinout, I was able to flash the fuse bits on the board and start the software development phase.

Initially, I wanted the device to use two PWM servo pulse signals for input, one for the left motor and one for the right motor. This worked for a time but was highly unreliable and prone to randomness and sudden failure, probably due to inaccuracies in the ATmega’s internal clock. This was not a huge problem, as I had designed the board with the MCU’s serial RX pin exposed as one of the digital inputs. Serial communication was immediately reliable, though not being bidirectional was not fun.

The first communication protocol was based on an integer between 0 and 200. Numbers from 0 to 100 corresponded to motor 1, and numbers from 100 to 200 corresponded to motor 2. At 50 and 150 respectively, each motor would be stopped. At values closer to the upper limit, a motor would spin forwards, and at values closer to the lower limit, a motor would spin backwards. I have yet to determine if this protocol is stupid or not.

I used SDMD R3 mostly successfully in Reflection 2, although the motor braking system caused huge interference problems. When the throttle passed the zero point, the braking system would engage instantly, meaning that repeated movements could cause voltage spikes to travel into the motors and create electrical problems. For Reflection 3, I solved this problem by simply not using braking, but afterwards, I wanted to resolve the issue.

Before I cover that, I want to briefly digress to another project called Klip. Klip is an ongoing attempt to make my connections between boards in my combat robots smaller using Stemma QT connectors that snap into place. I attempted to design a smaller SDMD based on this standard, but inherent flaws with the microcontroller I picked killed the project before it ever got off the ground. Klip has been integrated into revision 5 of the SDMD, replacing the old IO header. And another quick digression, revision 4 is identical to revision 3, but with larger through holes for wires.



Anyways, one day I sat down and drew up some ideas for a set-and-forget firmware for SDMD. I decided to base the firmware off of educational robots, something that, despite having a bit of a distaste for, I do respect the user-friendliness of. The new firmware would rely on commands, allowing it to be truly software-defined.

The commands come in two flavors. The less important commands are the direct commands, ones which directly control the output of the device. These are commands like “brake motor 1” or “failsafe shutdown”. The more complex commands are the interpreted commands. These commands configure the driver to interpret the next piece of data and do something based upon it. Writing to the motors use interpreted commands. First a command is sent, and then a number between 0 and 200 controls the motor to drive either forwards or backwards. There is also a command to set the curve factor, a number which controls how sharp the throttle response is. There is another command to determine if the motor driver will automatically brake, and what the activity timeout is before then.

This new revision is currently awaiting testing in Reflection 3.1.