**Mar 1**

The slow servo was caused by my LiFe battery which had one dead/disconnected cell and so was only outputting 3-ish volts instead of 6. Running on the 7.2V LiPo it works fine, albeit a bit quick. I have seen online documentation for the PyBoard that includes a servo class for Micropython that includes a speed parameter, as well as other goodies, like calibration of the end positions for each servo. This seems like a handy thing to have, but I don't think its available for the M5Stack, so I will roll my own:

Servo

\_angle

boolean enabled

int min angle

int max angle

setAngle(int angle)

int getAngle()

moveToAngle(int angle, int speed)

setAngle commands the servo to the desired angle at its maximum speed. The moveToAngle function divides the movement into increments of (currently) 1 degree with delays inserted after each one to slow down the servo. The speed is a percentage of maximum, so at the default of speed=100, the delay is 0 and the servo should move almost as fast as with setAngle() (there may be slight differences because of acceleration and deceleration in the servo controller board). If you set speed=50, it will add delays to slow the servo to roughly half the rated speed for the servo. Ideally this would be done in a way that isn't blocking. We'll see if that's too complicated to do.

**Feb 9**

I now have a simple menu running on the M5 that lets me select one of 4 different axes - twist, nod, tilt and jaw and then set a servo angle for each one. There is no constraint checking yet, nor does it take into account that the twist axis must move two servos simultaneously (in opposite directions? not sure).

Testing this with the simplest/safest axis (the head twist) it seems to move extremely slowly. The servos I connected yesterday move at normal speeds, so either the servo on blubot isn't getting enough current, there is too much friction from the 3d-printed gears, or it is damaged in some way. The simplest way to test this is to take the servo off and run the test again on the naked servo. If the servo is damaged, it is probably just the servo board, so I may be able to salvage it using the spare servo board left over from the abdomen servos (which run two servos off a single board). If it's a current supply issue, I'll have to figure out some way to beef up the supply. This will be trickier, so I hope not.

**Feb 8**

I have bought an M5Stack Fire. The plan is to use this to control the head servos. Eventually I would like this to be used as the low-level controller that can receive commands form (say) a raspberry Pi running image recognition software (and maybe speech recognition too). This would allow the head to be safely commanded by students because the M5stack would prevent any of the servos being driven to positions that could damage the hardware.

I'm using the M5 with the Servo Module, in order to have enough ports to connect the 5 or 6 servos needed.

I managed to get micropython working using VSCode quite easily by following the instructions on the M5Stack YouTube channel. But this did not seem to allow me to access the i2c bus. So I embarked on a very hairy process of reflashing the M5 using the command line tools at https://github.com/loboris/MicroPython\_ESP32\_psRAM\_LoBo/wiki/winsetup. This was a complete waste of time and I end up reflashing back to the UIFlow fork of MicroPython that I originally installed by following the M5Stack YT tutorial. The problem turned out to be that the i2c library is called i2c\_bus, not I2C, as mentioned in various online sources (I suppose this is just a different library).

The M5Stack Servo module has i2c address of 53h and is initialised by declaring:

i2c = i2c\_bus.easyI2C((21, 22), 0x53)

You can then set the angle directly by using

i2c.write\_u8(servoChannel + 0x10, angle)

The ServoModule supports 12 servos and these are labelled 0 - 11 on the hardware. But you need to add 16 to this when passing the channel number to the write command.

If you just pass the channel number itself, it doesn't do the pwm needed to control the servo. I haven't experimented with how this works though.

Challenges:

Write text to the LCD

Send data via i2c to one of the servo ports

Use pwm to actually drive the servo to a known position

Connect multiple servos to test power draw during simultaneous use

Create some kind of model to relate head position to servo angles

Build a constraint system that prevents the head being commanded to a damaging position.

Add a pretty UI that provides status info for the servos

Allow serial/BLE communication with something else that can issue high-level positioning commands