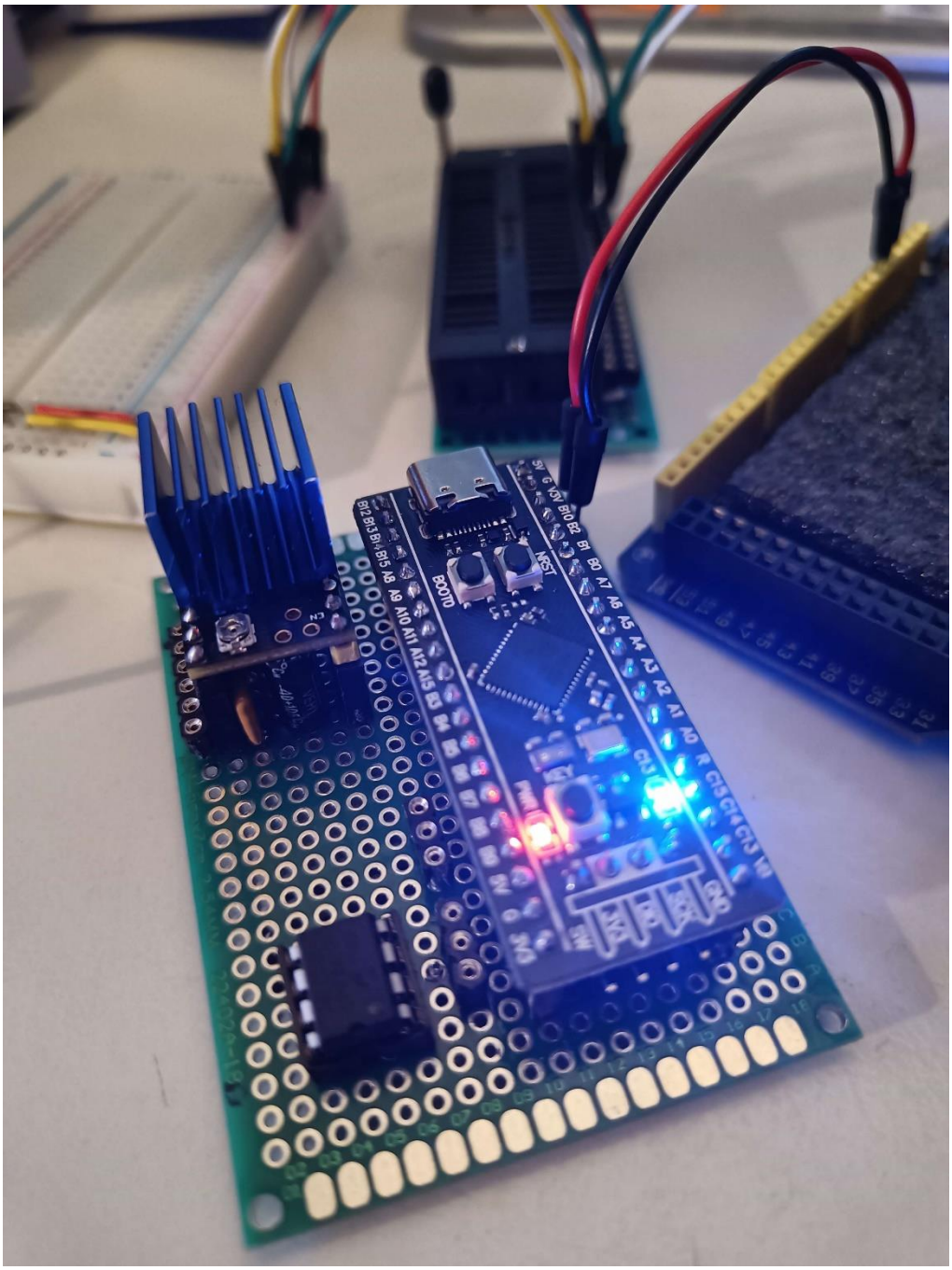
Robot Arm Project



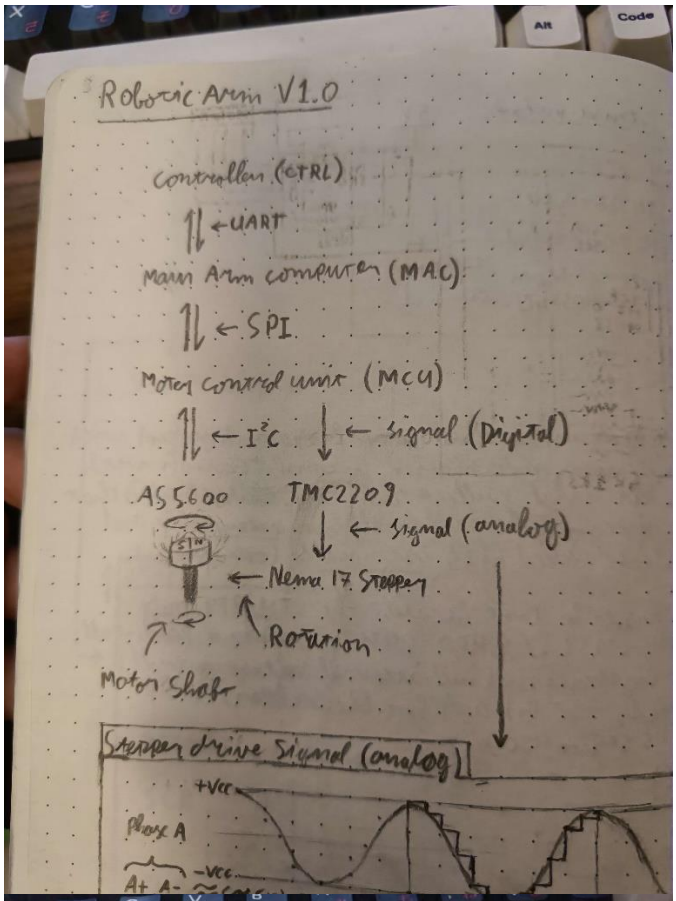
Studentnaam: Marijn Verschuren
Studentnummer: 510936

Klas: PD11
Vakdocent: Rop Pulles

Versie: V1.0
Datum: 02-01-2023

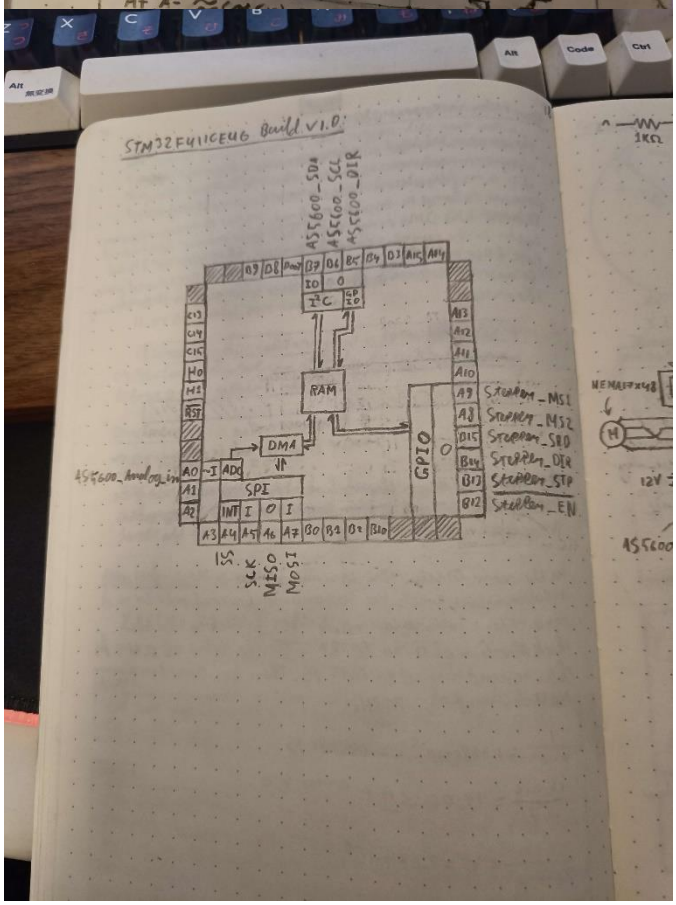
Contents

1. Design Iteraties:

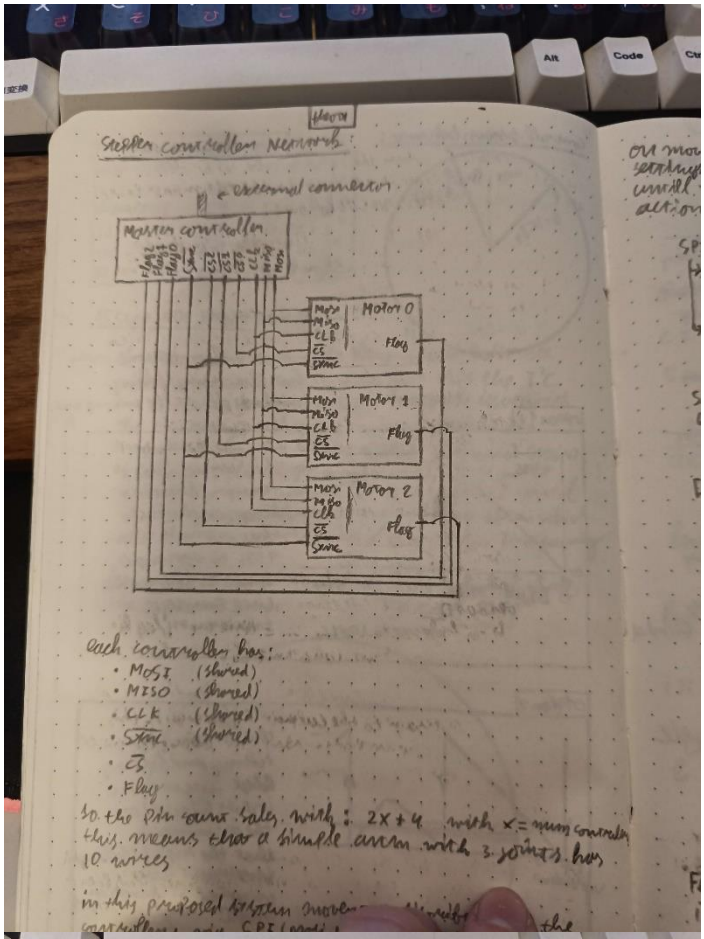


Vanaf het eerste design is het basis idee niet veel veranderd dit idee gaat als volgt:

Op de pc (controller) word een opdracht gemaakt, deze opdracht word vervolgens via de uart doorgestuurd naar de arm computer waarna deze opdracht gechecked en als het nodig is gerepareerd word. Als de opdracht in orde is word deze doorgestuurd naar de correcte motor control unit deze unit heeft in deze versie een sensor (magnetic angle encoder) een stepper motor driver.

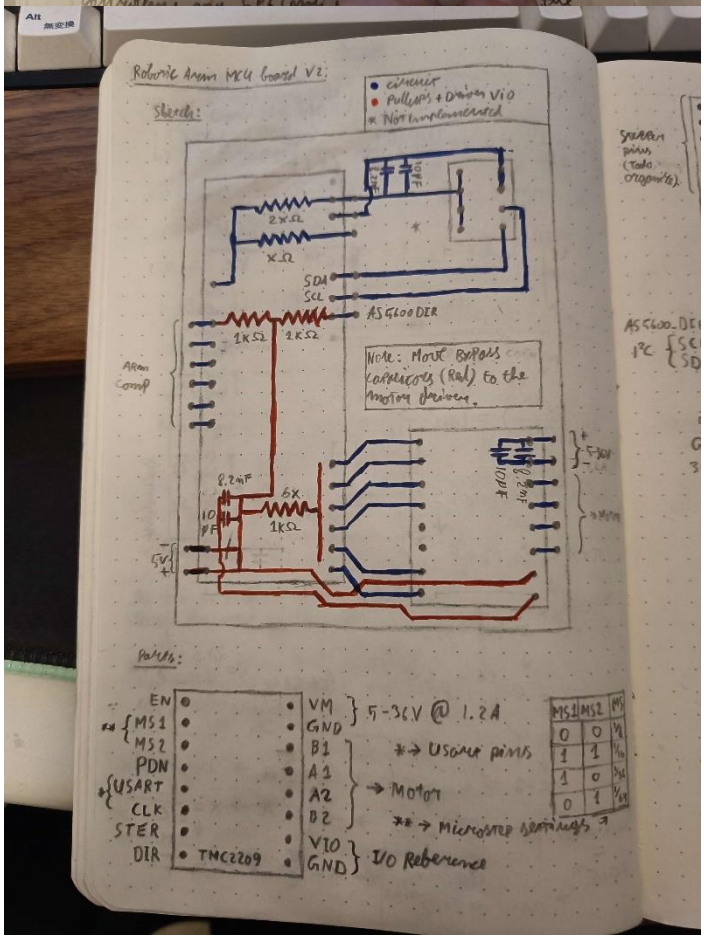


De volgende disign stap focust zich op de motor control unit en heeft maar één grote verandering en dat is de connectie aan de sensor. De sensor was in de vorige versie alleen met I^2C aan de unit aangesloten, in deze versie is de analoge functie van de AS5600 ook aangesloten aan de unit. Om de analoge functie aan te zetten moet de R4 pullup resistor van de AS5600 verwijderd worden.



Vanaf hier begint een soort tweede versie met de volgende veranderingen:

De arm computer heeft een SPI-bus die shared is over alle motor units met elk een chip select. De enige verandering is dat er een Flag en later een Interupt pin is toegevoegd. De Flag word gebruikt als een verificatie van het ontvangen van de opdracht en de Interupt word gebruikt om de volgende opdracht (in queue) uit te voeren.



En hier is de huidige versie van de motor control unit. Veranderingen:

Rom chip toegevoegd om arm segment vorm (als vector) op te slaan

Bypass capacitors toegevoegd om de power supply voltage consistent te houden

2. Huidige Arm_Computer code:

```
RX_data = new_uart_ibuf(&huart2, 1024);  // starts receiving
```

Er word een UART buffer aangemaakt die door de DMA gevuld word

(de handshake code is nog niet uitgewerkt dus gaat waarschijnlijk veranderen)

```
uint32_t baud = 9600;  // default baud
CTRL_Handshake init;
```

Dan word er een handshake aangemaakt en de baud-rate ingesteld (dit word nog niet gebruikt)

```
memset((void*)&init, 0, 6);
uint16_t crc = 0xffff;
```

De handshake word gereset en de crc op 0xffff zodat deze niet overeen komen.

```
while (init.crc != crc) {  // handshake
        if (uart_ibuf_align(RX_data, SYNC_BYTE)) { continue; }
        if (uart_ibuf_read(RX_data, &init, 6)) { continue; }
        crc = crc16_dnp(&init, 4);
}  // TODO: add CRC error correction
```

Hier word de handshake uitgevoerd totdat de crc uitkomt. Hier gaan waarschijnlijk wat veranderingen plaatsvinden omdat er nog vanaf een SYNC_BYTE afgelezen word. Dit vind ik niet zo handig omdat er floating point nummers doorgestuurd worden en de SYNC_BYTE daar vrij snel tussen zit.

```
init.motor_count = motor_count;  // set motor count and send message back
HAL_UART_Transmit(&huart2, (uint8_t*)&init, 6, 10);
```

Nadat het ontvangen gelukt is word de handshake aangepast door de motor count toe te voegen. Dan word deze terug gestuurd.

```
uart_ibuf_reset(RX_data);
MCU_Instruction instruction;
MCU_State state;
uint8_t return_code;
```

Hier word de UART buffer leeg gemaakt en de variables voor: state, instructie en return_code aangemaakt.

```
while (1) {
    if (uart_ibuf_align(RX_data, SYNC_BYTE)) { continue; }
    if (uart_ibuf_read(RX_data, &instruction, 32)) { continue; }

    return_code = 0;
    if (instruction.crc != crc16_dnp(&instruction, 30)) {
        // TODO: error correction
        // if not fix-able \/
        return_code |= RETURN_CRC_ERROR;
        HAL_UART_Transmit(&huart2, (uint8_t*)&return_code, 1, 10);
        continue;  // do not execute instruction
    }
```

In de mainloop word de instructie opgehaald uit de buffer en gecheckt of deze klopt.

```
return_code |= RETURN_OK;
HAL_UART_Transmit(&huart2, (uint8_t*)&return_code, 1, 10);
```

Als deze klopt word een ok signaal terug gestuurd.

```
do {
    HAL_GPIO_WritePin(CS_0_GPIO_Port, CS_0_Pin, 0);
    HAL_SPI_TransmitReceive(&hspi1, (uint8_t*)&instruction, (uint8_t*)&state, 32, 100);
    HAL_GPIO_WritePin(CS_0_GPIO_Port, CS_0_Pin, 1);
    HAL_Delay(10);  // give mcu time to react
} while (HAL_GPIO_ReadPin(TEST_CF_GPIO_Port, TEST_CF_Pin));
if (instruction.action & OVERRIDE) {
    HAL_GPIO_WritePin(C_INT_GPIO_Port, C_INT_Pin, 0);
    HAL_Delay(5);  // give mcu time to react
    HAL_GPIO_WritePin(C_INT_GPIO_Port, C_INT_Pin, 1);
}
if (instruction.action & POLL) {
    return_code = SYNC_BYTE;
    HAL_UART_Transmit(&huart2, (uint8_t*)&return_code, 1, 10);
    HAL_UART_Transmit(&huart2, (uint8_t*)&state, 32, 100);
}
```

Dan word de instructie doorgestuurd naar de motor control unit.

3. Huidige Motor_Control_Unit code:

```
int main(void)
{
  /* USER CODE BEGIN 1 */
    sensor = AS5600_new();
    sensor->i2c_handle = &hi2c1;
    sensor->dir_port = AS5600_DIR_GPIO_Port;
    sensor->dir_pin = AS5600_DIR_Pin;
    sensor->positive_rotation_direction = AS5600_DIR_CW;
```

De sensor word ingesteld.

```
// disable motor driver
HAL_GPIO_WritePin(STEPPER_NEN_GPIO_Port, STEPPER_NEN_Pin, 1);
```

De stepper driver word uitgezet.

```
// initialize AS5600 sensor
while (AS5600_init(sensor) != HAL_OK) {}
```

De settings worden geschreven naar de sensor.

```
state.vel = 0.0;
state.acc = 0.0;
state.instrution_id = 0;
state.queue_size = 0;
state.queue_index = 31;
state.micro_step = 0;
state.srd_mode = 0;
state.lock = 0;  // TODO:
```

De state word gereset

```
AS5600_get_angle(sensor, &state.raw_angle);
state.pos.angle =   state.raw_angle;
AS5600_pos_f64 =    state.raw_angle;  // set
```

De hoek van de motor word afgelezen via I^2C (dit word later gedaan via ADC met DMA naar raw_angle)

```
HAL_TIM_Base_Start(&htim2);  // start timer_2 (for delays)
HAL_TIM_Base_Start(&htim5);  // start timer_5 (for simulation time keeping)
```

Timers 2 en 5 worden aangezet

```
HAL_SPI_TransmitReceive_DMA(&hspi1, (uint8_t*)&state, (uint8_t*)&instruction_input, 32);

// start receiving ADC data
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)&state.raw_angle, 1);
```

De DMA devices worden aangezet.

```
HAL_GPIO_WritePin(STEPPER_NEN_GPIO_Port, STEPPER_NEN_Pin, 0);
TIM5->CNT = 0;
HAL_TIM_Base_Start_IT(&htim10);  // start timer_10  (sensor interupt) [100Hz]
```

De motor driver word aangezet, de timer counter word gereset (start simulatie) en de simulatie timer word aangezet timer 10. Deze interrupt word 100 keer per seconden uitgevoerd.

```
while (1) {
    if (step_gain < MIN_STEPPER_GAIN) { continue; }  // make sure t
s
    // dir is set in interrupt
    register uint16_t pulse_delay = MIN_STEPPER_DELAY / step_gain;
    HAL_GPIO_WritePin(STEPPER_STP_GPIO_Port, STEPPER_STP_Pin, 1);
    delay_us(pulse_delay);
    HAL_GPIO_WritePin(STEPPER_STP_GPIO_Port, STEPPER_STP_Pin, 0);
    delay_us(pulse_delay);
}
```

In de mainloop worden de stappen gezet. De direction en delay worden vanuit de interrupt aangepast.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim) {
    if (htim != &htim10) { return; }
    if (state.queue_size == 0) { return; }
```

Hier begint de interrupt call. Er word gecheckt of deze call komt vanuit de correcte timer en of er een opdracht is.

```
euler_method();  // update state.pos.angle, AS5600_delta_pos using the selected mode
double target_delta = instruction->target - state.pos.angle;
target_delta = ABS(target_delta) < ABS(target_delta - 4096) ? target_delta : target_delta - 4096;
```

Hier word de afstand vanaf de target uitgerekend (hier moet nog wat werk gedaan worden). Op de “euler_method” kom ik later terug.

```
if (ABS(target_delta) > 10) {  // ~1 deg
    HAL_GPIO_WritePin(STEPPER_DIR_GPIO_Port, STEPPER_DIR_Pin, target_delta < 0);
    step_gain = MIN(1, ABS(target_delta / 1024));  // all deltas greater than 1/8
```

Hier word de stap snelheid uitgerekend (ook hier moet nog aan gewerkt worden)

```
} else if (!state.lock) {
    instruction = get_next_queue_ptr();
    if (!instruction) { return; }
    // HAL_GPIO_WritePin(STEPPER_NEN_GPI
truction is loaded
    set_motor_setting(instruction);
    step_gain = 0;
    state.lock = 1;
    // TODO: build variables for stepper
}
```

Hier word de volgende instructie geladen. De lock word gereset met de instruction interrupt pin. (hier moet nog gewerkt worden aan het laden van de volgende instructie enz…)

```
void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi) {
    // only hspi1 is used so there is no need to check
    void* dst = get_next_empty_queue_ptr();
    if (!dst) { return; }  // exit if queue is full
```

Dit is de SPI callback waarin de instructie word toegevoegd aan de queue

```
HAL_GPIO_WritePin(STATUS_PIN_GPIO_Port, STATUS_PIN_Pin, \
    (instruction_input.crc != crc16_dnp((uint8_t*)&instruction_input, 30))
);
memcpy(dst, (void*)&instruction_input, sizeof(MCU_Instruction));
state.queue_size++;
```

De Flag pin word zo gezet dat de arm computer kan zien of de instructie goed is aangekomen (LOW = success).

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    // this will not instantly load next instruction
    if (GPIO_Pin == NSS_Pin) { HAL_GPIO_WritePin(STATUS_PIN_GPIO_Port, STATUS_PIN_Pin, 1); }
pin until reset from SPI_TxRxCplt callback on success
    if (GPIO_Pin == INSTUCTION_INT_Pin) { state.lock = 0; }
}
```

Dit is de exti callback code. Als de NSS (SPI non chip/slave select) de interrupt triggered word de Flag op HIGH gezet zodat de arm computer kan wachten totdat deze gereset is zonder dat er een risico is dat de arm computer de motor control unit inhaalt (race condition). Als de instruction interrupt pin de interrupt triggered word de instructie lock weggehaald.

```
void euler_method() {  // typical execution time ~45 us
    register uint16_t raw = ACD_RANGE_CONV * (state.raw_angle - MIN_ADC_IN);
    register uint16_t pos_diff = (AS5600_pos_f64 - raw);  // rotation detection
    state.pos.rotation += pos_diff > 2048; state.pos.rotation -= pos_diff < -2048;
    register double alpha = 1 / ((EULER_TAU / TIM5->CNT) + 1);
    AS5600_pos_f64 = (raw * alpha) + ((1 - alpha) * AS5600_pos_f64);
    state.vel = (1e6 / TIM5->CNT) * ((uint16_t)AS5600_pos_f64 - state.pos.angle) * AS5600_RAD_CONV;
    state.pos.angle = (uint16_t)AS5600_pos_f64;
    TIM5->CNT = 0;
}
```

In de euler method worden de errors die bij analoge signalen komen kijken zo goed mogelijk weggewerkt en de velocity uitgerekend (hier moet nog aan gewerkt worden). Hier zie je ook gebruik van register variables zodat (hopelijk) de compiler de variabelen niet in de ram opslaat wat relatief traag is tegenover registers. (nu weet ik niet wat voor registers op de CORTEX M4 zitten dus dit is niet gegarandeerd)

4. Documentatie en Libraries:

Voledige repository: https://github.com/MarijnVerschuren/Robotic_Arm

AS5600 library repository: https://github.com/MarijnVerschuren/STM32F4_AS5600_library

24LC512IP library:
https://github.com/MarijnVerschuren/EEPROM_Programmer/tree/main/MCU/lib/24LC512IP
Deze moet nog afgemaakt en geisoleerd worden.

Video documentatie (in orde):

https://youtube.com/shorts/iROxng_Yvek

https://youtube.com/shorts/G3Ni2nRQHQA

https://youtu.be/2IKLwBE1TLw

https://youtu.be/rBEvl1Z75_w

https://youtube.com/shorts/V_E33Jzvdlo

5. Controller code:

```
def init(adapter: str, *init_args, baud: int = 9600, time_out: float = 0.1) -> int:
    ser = serial.Serial(
        port=adapter, baudrate=baud, bytesize=serial.EIGHTBITS,
        parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_TWO,
        timeout=time_out
    )
```

Hier word in de init functie de serial port geopend.

```
ser.reset_input_buffer()
ser.reset_output_buffer()

data_o = new_CTRL_Handshake(0, *init_args)
data_crc = get_CTRL_Handshake_data(data_o)[3]
```

Hier worden de in- en output buffer gereset. Ook word de handshake en de crc aangemaakt.

```
while True:
    ser.write(bytes([SYNC_BYTE]))  # sync byte
    ser.write(data_o)
    if ser.inWaiting() < 6: continue  # no response
    data_i = get_CTRL_Handshake_data(ser.read(6))
    if data_i[1:3] == init_args and data_i[3] == data_crc:
        if init_args[1] != baud: return init(adapter, *init_args, baud = init_args[1])
        return data_i[0]  # motor_count
```

In de initloop word de handshake gestuurd totdat er een terug komt.

```
def run(adapter: str, baud: int = 9600, time_out: float = 0.1) -> None:
    motor_count = init(adapter, 1, baud)  # init args: init_0=1, baud=baud
```

In de run function word de init functe gecalled om de motor count te krijgen.

```
while True:
    input("send: ")
    # make instruction
    target = float(input("target (f64, rad): "))
    max_vel = float(input("max vel (f64, rad / s
```

Onder de run loop worden wat inputs gevraagd om een instructie te maken. (dit gaat natuurlijk veranderd worden)

```
return_code = 0
while (not return_code & RETURN_FLAGS.OK):
    ser.write(bytes([SYNC_BYTE]))  # sync byte
    ser.write(instruction)

    return_code = int.from_bytes(ser.read(1), "little")
    print(return_code)
```

De instructie word gestuurd totdat er een ok return code terug komt