Code Review

Code: Olaf Goudriaan
Reviewer: Tim Leijssen

macro.h:2-11:

```
#define uint16 unsigned int
#define int16 int
enz.
```

Op zich is dit goed, maar het is misschien veiliger om types uit stdint.h te gebruiken (als dat beschikbaar is natuurlijk), dan zijn deze groottes niet meer hardware afhankelijk. Bijv.:

```
#include <stdint.h> // misschien moet dat <cstdint> zijn
#define int16 int16_t
```

Of anders int16_t direct gebruiken.

macro.h:39-51:

```
#define IDLE 0 /** System State */
#define RUNNING 1 /** System State */
#define ERROR 2 /** System State */
#define INTERNAL_ERROR_RESTART_REQUIRED 3 /** System State */
```

Beter om hier een enum te gebruiken:

```
enum SystemState {
    IDLE,
    RUNNING,
    ERROR,
    INTERNAL_ERROR_RESTART_REQUIRED,
}
```

Of misschien nog beter, om eventuele naam conflicten te vermijden kan je een prefix toevoegen zoals je later bij STEPPER_ en AXIS_ wel doet:

```
enum SystemState {
    SYSTEM_IDLE,
    SYSTEM_RUNNING,
    SYSTEM_ERROR,
    SYSTEM_INTERNAL_ERROR_RESTART_REQUIRED,
}
```

macro.h:73:

```
#define X_STEP_PIN PB0
```

Dit is misschien mijn persoonlijke voorkeur maar ik zou de naamgeving andersom doen.

```
#define PIN_STEP_X PB0
```

Dan weet ik bijv. dat alle pin id's met PIN_ beginnen.

main.cpp:28:

```
while (!(systemState == INTERNAL_ERROR_RESTART_REQUIRED)) {
```

Kan natuurlijk ook:

```
while (systemState != INTERNAL_ERROR_RESTART_REQUIRED) {
```

system.cpp:11-13

```
static char line[MAX_LINE_SIZE];
static uint8 size = 0;
static bool lineComment = false;
```

Moeten deze static zijn? De while-loop hieronder wordt alleen onderbroken als line vol is (size == MAX_LINE_SIZE). In dit geval gaat hij de functie uit, maar bij de volgende aanroep blijft size gelijk aan MAX_LINE_SIZE (omdat deze static is), dus hij zal geen data meer verwerken.

system.cpp:18:

```
    if(c == 32) continue;       /** filter spaces */
```

Beter:

```
    if(c == ' ') continue;
```

system.cpp:19

```
    if(lineComment == true) {
```

Kan natuurlijk gewoon zijn:

```
    if (lineComment) {
```

system.cpp:24-76

Misschien dat het duidelijker is als je het verwerken van de line-buffer niet in dezelfde loop doet als het inlezen ervan, bijv.:

```
    // Inlezen:
    while (size < MAX_LINE_SIZE) {
        unsigned char c = uartRead();
        // Alle verschillende waardes van c hier afhandelen
        if (c == ...) ...
        if (c == '\n' || c == '\0') {
            line[size++] = '\n';
            break;
        }
    }
    // Verwerken
    if (line[0] = '$') {
        ...
    } else if (...) {
        ...
    } // Enz.
```

system.cpp:58-61

```
    size = 0;
    for(int i = 0; i < MAX_LINE_SIZE; i++) {
        line[i] = '\0';
    }
```

In principe is het met '\0' overschrijven van line overbodig, omdat je de size toch los bijhoudt. Ik zou hooguit alleen het eerste karakter op '\0' zetten:

```
    size = 0;
    line[0] = '\0';
```

system.cpp:113:

```
    mantissa = round(100 * (value - intValue));
```

Deze wordt zo te zien niet gebruikt.

system.cpp:118-133

```
      switch (intValue) {
      case 0: case 1: case 2: case 3: ...
      case 17: case 18: case 19: ...
      }
```

Misschien hier even in commentaar bij zetten wat deze commandos betekenen.

system.cpp:151-175:

```
switch (letter) {
case 'A': break;
case 'B': break;
case 'C': break;
...
default: break;
}
```

Alles wat niet gebruikt wordt zou ik hier gewoon weglaten uit de switch-statement, deze worden al door de default branch afgehandeld.

system.cpp:183:

```
codeBlock.endPos[X_AXIS] = prevPos[X_AXIS] + codeBlock.endPos[X_AXIS];
```

Iets korter/duidelijker:

```
codeBlock.endPos[X_AXIS] += prevPos[X_AXIS];
```

system.cpp:205:

```
bool readFloat(char *line, uint8 *char_counter, float *float_ptr) {
```

Zo te zien is dit een hele functie om floats te parsen. Kan je niet gewoon stanaard functies als strtof o.i.d. gebruiken?

system.cpp:291:

```
char* getStatus(int s) {
```

Ik zou deze statusToString() noemen, want dat is in feite wat je doet hier.

datacommunication.cpp:

Ik zie dat hier print-functies zijn om bijv. integers en floats als strings te printen, zijn daar ook geen standaard functies voor? Bijv. itoa/dtostrf?
Afgezien van dat, zie ik hier redelijk wat dubbele code, bijv. er zijn bijna identieke print(int16) en print(int32), misschien dat je die eerste als de tweede kan aanroepen:

```
void print(int16 n) {
    print((int32) n);
}
```

Al is dat misschien iets trager.
Ook:

```
void printHline(uint8 n) {
    printHLine(n, '=');
}
```

stepper.cpp:22:

```
if(stepperData.currentBlock->command == 0 || ...) {
```

Veel namen hier zijn nogal langdradig en komen meerdere malen voor, dat maak het moeilijker leesbaar. Misschien kan je ze in een tijdelijke variabele stoppen om de code wat in te korten.

```
int command = stepperData.currentBlock->command;
if(command == 0 || command == 1) {
```

Dat geldt ook voor de stepperData.xAxisNumber en yAxisNumber.
Ik weet zo niet wat de command getallen 0 - 3 hier betekenen. Misschien dat je deze als enum/define constantes kan doen?

```
enum {
    COMMAND_MOVE, // o.i.d.
    COMMAND_???,
    ...
}
...
if(command == COMMAND_MOVE) {
```

stepper.cp:175-194:

```
    for(uint8 i = 0; i < 5; i++) {
        steps = ...;
        if(steps <= 65535) {
            ...
            i = 10;
        }
    }
```

Die i = 10 komt neer op een break statement, dus dat is natuurlijk duidelijker.
Je kan eventueel de if-statement hier omdraaien:

```
    for(uint8 i = 0; i < 5; i++) {
        steps = ...;
        if(steps > 65535) { continue; }
        ...
        break;
    }
```

stepper.cpp:257:

```
    float toPowerOf2(float f) {
        return f * f;
    }
```

Dit heet een "square" normaal gesproken, "powers of two" zijn 2^x dus bijv 2, 4, 8, 16, 32.

stepper.cpp:306-307

```
    float maxValue = abs(stepperData.formulaValues[X_AXIS]) > ...
    maxValue = ...
```

Ook hier is het zowel korter als duidelijker om het in tijdelijke variabelen te zetten.

```
    float absX = abs(stepperData.formulaValues[X_AXIS]);
    float absY = ...;
    float absZ = ...;
    float maxValue = absX > absY ? absX : absY;
    maxValue = maxValue > absZ ? maxValue : absZ;
```

Het zou goed kunnen dat er ook een max() functie is, dan kan je doen:

```
    float maxValue = max(max(absX, absY), absZ);
```

stepper.cpp:313,320

```
    if((pow(block->R, 2)) - ...)
```

Dit had natuurlijk de toPowerOf2 (of eigenlijk square) functie boven kunnen gebruiken.

```
    if(square(block->R) - ...)
```

Dat is waarschijnlijk nog sneller ook. pow() kan ook niet-gehele exponenten aan (bijv. pow(x, 2.5)) maar de tradeoff is dat het een stuk rekenintensiever is, vergeleken met twee getallen vermenigvuldigen.