Reste à faire

Au soir du 13/2, les indispensables :

- Interfacer les deux TOF avec le hub I2C.
- Tester IX, en démarrage à `FX 10`
- Ajouter la butée de carotte
- Router correctement le câble d'alimentation et l'USB.
- Redesigner le support du hub et l'imprimer.

le facultatif :

- Ré-imprimer boîter
- Vidéo montrant l'interface et les déplacements
- Changer les noms de VX en FX et VZ en FZ.
- Ajouter le fin de course Z bas (imprimer support)
- Ajouter le fin de course X gauche (concevoir et imprimer support)

On voit le bout !

XRF_bench_v1.ino

Dernière version du programme :

```
#include "M5Unified.h"
#include "Module_Stepmotor.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

#include <Wire.h>
#include <VL53L0X_mod.h>      // VL53L0X sensor library
#include <SparkFun_VL53L1X.h>  // VL53L1X sensor library

// I2C address for the TCA9548A multiplexer
#define TCAADDR 0x70

// LEDC PWM definitions for motor control
const int pwmChannelX = 0;    // PWM channel for X axis
const int pwmChannelY = 1;    // PWM channel for Y axis (mirror of X)
const int pwmChannelZ = 2;    // PWM channel for Z axis

const int pwmPinX = 16;       // PWM pin for X axis
const int pwmPinY = 12;       // PWM pin for Y axis
const int pwmPinZ = 15;       // PWM pin for Z axis

int Xspeed = 10000;
int Zspeed = 15000;

// *****************************************************************************
// I2C Multiplexer Setup
// *****************************************************************************
void tcaselect(uint8_t i) {
  if (i > 7) return;
  Wire.beginTransmission(TCAADDR);
  Wire.write(1 << i);
  Wire.endTransmission();
}

// *****************************************************************************
// Sensor Objects (each on its own multiplexer channel)
// *****************************************************************************
VL53L0X_mod   Zsensor;  // Connected on multiplexer channel 0
SFEVL53L1X    Xsensor;  // Connected on multiplexer channel 1

// *****************************************************************************
// Global Variables and Objects
// *****************************************************************************
String inputString = "";
bool stringComplete = false;
static Module_Stepmotor driver;

// *****************************************************************************
// Serial Event - Called when new data arrives on Serial
// *****************************************************************************
void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}

// ********************************************************************************
// Helper Functions for Motor Control
// ********************************************************************************

// Reads and prints the current sensor values for the X and Z axes.
void status() {
  // Read sensor values:
  // For the X axis sensor (VL53L1X), we assume getDistance() returns the distance in millimeters.
  tcaselect(1);
  int xDistance = Xsensor.getDistance();
  // For the Z axis sensor (VL53L0X), we assume readRangeContinuousMillimeters() returns the distance in millimeters.
  tcaselect(0);
  int zDistance = Zsensor.readRangeContinuousMillimeters();

  driver.getExtIOStatus();

  // Print sensor values to the Serial monitor
  // Human readable:
  Serial.print("X: ");
  Serial.print(xDistance);
  Serial.print(" mm, Z: ");
  Serial.print(zDistance);
  Serial.print(" mm, Z stop: ");
  Serial.println(driver.ext_io_status[1]);

  // For IHM:
  Serial.print("K ");
  Serial .print(driver.ext_io_status[1]);
  Serial.print(" ");
  Serial.print(xDistance);
  Serial.print(" ");
  Serial.println(zDistance);

  // Optionally, update the M5Stack LCD to display the status:
  M5.Lcd.fillRect(0, 60, 320, 30, TFT_BLACK); // Clear a portion of the display for status info
  M5.Lcd.setCursor(0, 60);
  M5.Lcd.print("X: ");
  M5.Lcd.print(xDistance);
  M5.Lcd.print(" mm, Z: ");
  M5.Lcd.print(zDistance);
  M5.Lcd.print(" mm");
}

// Reads and prints the current X sensor value.
void statusX() {
  tcaselect(1);
  int xDistance = Xsensor.getDistance();

  Serial.println(xDistance);
}

// Resets PWM outputs (motor control signals) to 0 duty cycle
void resetMotorOutputs() {
  ledcWrite(pwmChannelX, 0);
  ledcWrite(pwmChannelY, 0);
  ledcWrite(pwmChannelZ, 0);
}

// Moves Z axis: if 'positive' is true, moves upward; otherwise, downward
void moveZ(bool positive) {
  resetMotorOutputs();
  digitalWrite(0, positive ? 1 : 0); // Set Z direction pin (pin 0)
  ledcWrite(pwmChannelZ, 1);
}

// Moves the Z axis until the Z sensor reports the desired distance (in millimeters)
void moveZto(int targetDistance) {
  const int tolerance = 5; // Tolerance in mm
  // Select the multiplexer channel for the Z sensor (channel 0)
  tcaselect(0);
  int currentDistance = Zsensor.readRangeContinuousMillimeters();  // Get current distance (mm)
  bool movePositive = (currentDistance < targetDistance);
  // Start moving in the correct direction
  moveZ(movePositive);

  // Loop until the sensor reading is within the tolerance
  while (abs(currentDistance - targetDistance) > tolerance) {
    tcaselect(0);  // Select channel 0 before each sensor read
    currentDistance = Zsensor.readRangeContinuousMillimeters();
    delay(50);  // Small delay to allow sensor update

    Serial.print("Z distance: ");
    Serial.println(currentDistance);
  }
  // Once target is reached, stop the motor
  stopMotion();
}

// Homes the Z axis: moves upward until a limit switch is triggered, then stops.
void homeZ() {
  resetMotorOutputs();
  // Start moving upward (+Z)
  digitalWrite(0, 1);
  ledcWrite(pwmChannelZ, 1);
  driver.getExtIOStatus();
  while(driver.ext_io_status[1]) {
    driver.getExtIOStatus();
  }
  // Stop movement and reverse direction if needed
  resetMotorOutputs();
  digitalWrite(0, 0);
  M5.Lcd.drawString("Reached Z=0", 0, 40, 2);
}

// Sets speed for the Z axis by reconfiguring the PWM frequency (speedKHz in kHz)
void setZSpeed(int speedKHz) {
  // Reconfigure PWM channel Z with the new frequency
  ledcSetup(pwmChannelZ, speedKHz * 1000, 8);

  Zspeed = speedKHz;
}
```

```
// Moves X axis: if 'positive' is true, moves in the positive direction; otherwise, negative.
// Direction is set using digital pins 17 and 13.
void moveX(bool positive) {
  resetMotorOutputs();
  digitalWrite(17, positive ? 0 : 1);
  digitalWrite(13, positive ? 1 : 0);
  ledcWrite(pwmChannelX, 1);
  ledcWrite(pwmChannelY, 1);
}

// Smooth acceleration
void smoothXMove(bool positive) {
  resetMotorOutputs();
  digitalWrite(17, positive ? 0 : 1);
  digitalWrite(13, positive ? 1 : 0);

  ledcSetup(pwmChannelX, 1000, 8);
  ledcSetup(pwmChannelY, 1000, 8);

  ledcWrite(pwmChannelX, 1);
  ledcWrite(pwmChannelY, 1);

  for(int i=1000 ; i<Xspeed ; i+=300) {
    ledcSetup(pwmChannelX, i, 8);
    ledcSetup(pwmChannelY, i, 8);

    delay(20);
  }
}

// Moves the X axis until the X sensor reports the desired distance (in millimeters)
void moveXto(int targetDistance) {
  const int tolerance = 3; // Tolerance in mm
  // Select the multiplexer channel for the X sensor (channel 1)
  tcaselect(1);
  int currentDistance = Xsensor.getDistance();  // Get current distance (mm)
  bool movePositive = (currentDistance < targetDistance);

  // Start moving in the correct direction
  // moveX(movePositive);
  smoothXMove(movePositive);

  // Loop until the sensor reading is within the tolerance
  while (abs(currentDistance - targetDistance) > tolerance) {
    if( tolerance*15 > abs(currentDistance - targetDistance) > tolerance*10 ) {
      ledcSetup(pwmChannelX, int(Xspeed/2), 8);
      ledcSetup(pwmChannelY, int(Xspeed/2), 8);
    }
    if( tolerance*10 > abs(currentDistance - targetDistance) > tolerance*5 ) {
      ledcSetup(pwmChannelX, int(Xspeed/3), 8);
      ledcSetup(pwmChannelY, int(Xspeed/3), 8);
    }
    if( tolerance*5 > abs(currentDistance - targetDistance) > tolerance*2 ) {
      ledcSetup(pwmChannelX, int(Xspeed/4), 8);
      ledcSetup(pwmChannelY, int(Xspeed/4), 8);
    }

    tcaselect(1);  // Select channel 1 before each sensor read
    currentDistance = Xsensor.getDistance();
    delay(50);  // Small delay to allow sensor update (adjust as needed)
  }
  // Once target is reached, stop the motor
  stopMotion();
}

// Sets speed for the X axis by reconfiguring the PWM frequency on both X and Y channels
void setXSpeed(int speedKHz) {
  // Reconfigure PWM channels X with the new frequency
  ledcSetup(pwmChannelX, speedKHz * 1000, 8);
  ledcSetup(pwmChannelY, speedKHz * 1000, 8);

  Xspeed = speedKHz;
}

// Stops all motor movement
void stopMotion() {
  resetMotorOutputs();
}

// **************************************************************************************
// Command Processing
// *********************************************************************************************

// Processes an array of tokens (parsed from serial input)
// Supported commands: IZ, IX, +Z, -Z, VZ, VX, +X, -X, and S (stop)
void processCommand(String tokens[], int tokenCount) {
  String cmd = tokens[0];
  if (cmd == "IZ") {
    homeZ();
  }
  else if (cmd == "+Z") {
    driver.getExtIOStatus();
    if (driver.ext_io_status[1]) {  // Only move +Z if limit switch is free
      moveZ(true);
    }
  }
  else if (cmd == "-Z") {
    moveZ(false);
  }
  else if (cmd == "VZ" && tokenCount > 1) {
    setZSpeed(tokens[1].toInt());
  }
  else if (cmd == "VX" && tokenCount > 1) {
    setXSpeed(tokens[1].toInt());
  }
  else if (cmd == "+X") {
    // moveX(true);
    smoothXMove(true);
  }
  else if (cmd == "-X") {
    // moveX(false);
    smoothXMove(false);
  }
  else if (cmd == "S") {
    stopMotion();
  }
  else if (cmd =="X") {
    moveXto(tokens[1].toInt());
  }
  else if (cmd =="Z") {
    moveZto(tokens[1].toInt());
  }
  else if (cmd =="E") {
    status();
  }
  else {
    Serial.println("Unknown command");
    M5.Lcd.drawString("Unknown command", 10, 300, 2);
  }
}

// ********************************************************************************
// Setup Function: Initializes sensors, motor driver, LEDC channels, and I/O pins
// ********************************************************************************
void setup() {
  delay(500);
  M5.begin();
  Wire.begin();
  Serial.begin(115200);
  Serial.println("Starting up...");
  delay(1000); // Allow sensors to power up

  // ----- Initialize Zsensor on multiplexer channel 0 -----
  tcaselect(0);
  if (!Zsensor.init()) {
    Serial.println("Failed to initialize Zsensor on channel 0");
    while (1);
  }
  Zsensor.setTimeout(500);
  Zsensor.startContinuous();
  Serial.println("Zsensor initialized on channel 0");

  // ----- Initialize Xsensor on multiplexer channel 1 -----
  tcaselect(1);
  if (Xsensor.begin() != 0) {
    Serial.println("Failed to initialize Xsensor on channel 1");
    while (1);
  }
  Xsensor.startRanging();
  Serial.println("Xsensor initialized on channel 1");
  inputString.reserve(200);
  driver.init(Wire);

  driver.enableMotor(1);
  Serial1.begin(115200, SERIAL_8N1, 35, 5);
  Serial2.begin(115200, SERIAL_8N1, 34, 26);
  Serial2.setTimeout(100);

  // Setup LEDC channels for motor control using ESP32 API
  ledcSetup(pwmChannelX, Xspeed, 8);  // Setup X axis PWM on channel 0
  ledcAttachPin(pwmPinX, pwmChannelX);
  ledcSetup(pwmChannelY, Xspeed, 8);  // Setup Y axis PWM on channel 1
  ledcAttachPin(pwmPinY, pwmChannelY);
  ledcSetup(pwmChannelZ, Zspeed, 8);   // Setup Z axis PWM on channel 2
  ledcAttachPin(pwmPinZ, pwmChannelZ);

  // Initialize PWM duty cycles to 0
  ledcWrite(pwmChannelX, 0);
  ledcWrite(pwmChannelY, 0);
  ledcWrite(pwmChannelZ, 0);

  // Configure digital I/O pins for motor direction control
  pinMode(17, OUTPUT); // X axis
  pinMode(13, OUTPUT); // "Y" axis (mirror of X)
  pinMode(0, OUTPUT);  // Z axis

  // Initialize direction states (e.g., set Z upward)
  digitalWrite(17, 0);
  digitalWrite(13, 1);
  digitalWrite(0, 1);

  M5.Lcd.setTextSize(2);
  M5.Lcd.drawString("Setup completed", 0, 0, 2);
  Serial.println("Setup completed");
  M5.update();
}

// ******************************************************************************
// Main Loop: Checks for serial commands and button presses to test movements
// *****************************************************************************
void loop() {
  // Array to hold up to 10 command tokens
  static String tokens[10];
  int tokenCount = 0;

  M5.update();

  // Process serial input when a complete command is received
  if (stringComplete) {
    Serial.print("Command received: ");
    Serial.print(inputString);
    M5.Lcd.drawString("Command received:", 0, 10, 2);
    M5.Lcd.drawString(inputString, 0, 30, 2);

    // Tokenize the input command string
    String message = inputString;
    message.trim();
    inputString = "";
    stringComplete = false;

    while (message.length() > 0 && tokenCount < 10) {
      int index = message.indexOf(' ');
      if (index == -1) {
        tokens[tokenCount++] = message;
        break;
      } else {
        tokens[tokenCount++] = message.substring(0, index);
        message = message.substring(index + 1);
      }
    }

    // Execute the parsed command
    processCommand(tokens, tokenCount);
  }

  // ------------------------------
  // Test controls using M5Stack buttons:
  // ------------------------------
  if (M5.BtnA.wasPressed()) {  // Test +X movement
```

```
    moveX(true);
  }
  if (M5.BtnB.wasPressed()) {  // Test -X movement
    moveX(false);
  }
  if (M5.BtnC.wasPressed()) {  // Stop all motion
    stopMotion();
  }

  // Update LCD indicator for the Z-axis limit switch status
  driver.getExtIOStatus();
  if (driver.ext_io_status[1]) {
    M5.Lcd.fillRect(300, 0, 20, 20, TFT_GREEN); // Z up switch is free
  } else {
    M5.Lcd.fillRect(300, 0, 20, 20, TFT_RED);   // End-course switch is pressed
  }
}
```

Revision #7
Created 13 February 2025 20:42:28 by Christian Simon
Updated 15 February 2025 11:16:31 by Christian Simon