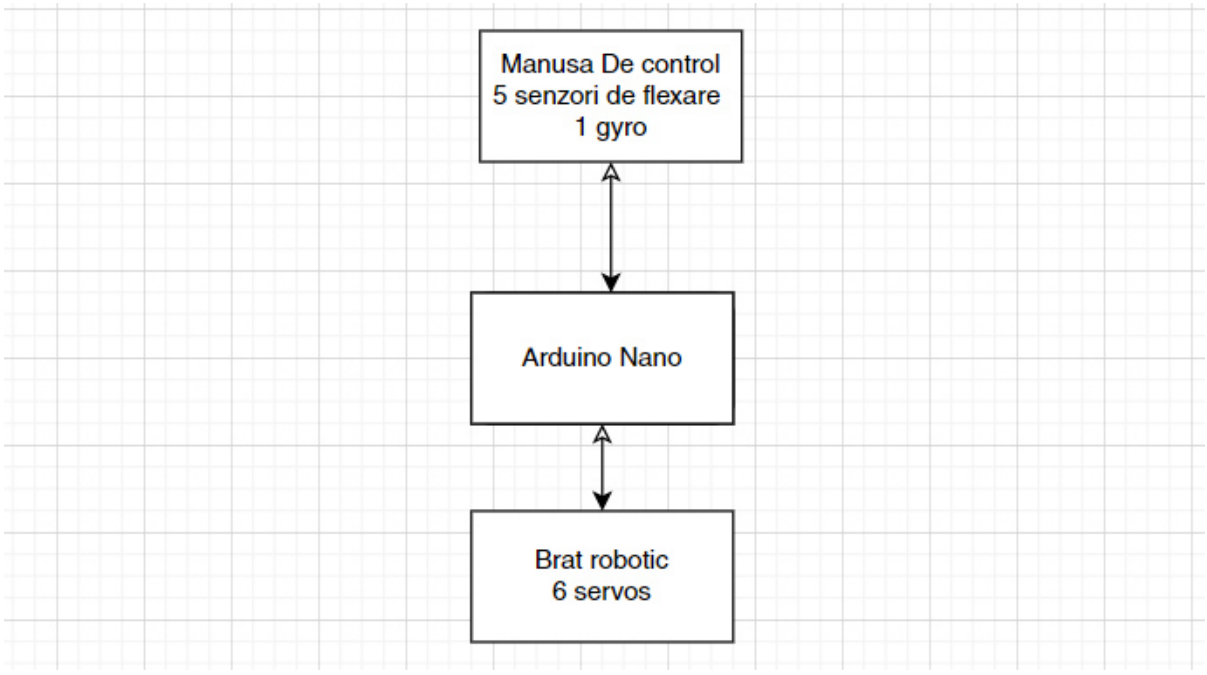
Brat Robotic

Introducere

Un brat robotic care isi poate misca cele 5 degete si sa isi roteasca palma in functie de miscarile omului care il controleaza

Descriere generală

Cel care controleaza robotul isi pune o manusa care are 5 senzori de flexie si un giroscop. Senzorii si giroscopul trimit senzorii pe o magistrala proprietara catre un Arduino Nano care printr-o magistrala controleaza cele 6 servomotoare care alcatuiesc bratul robotic. 1 pentru fiecare deget si unu pentru rotire.

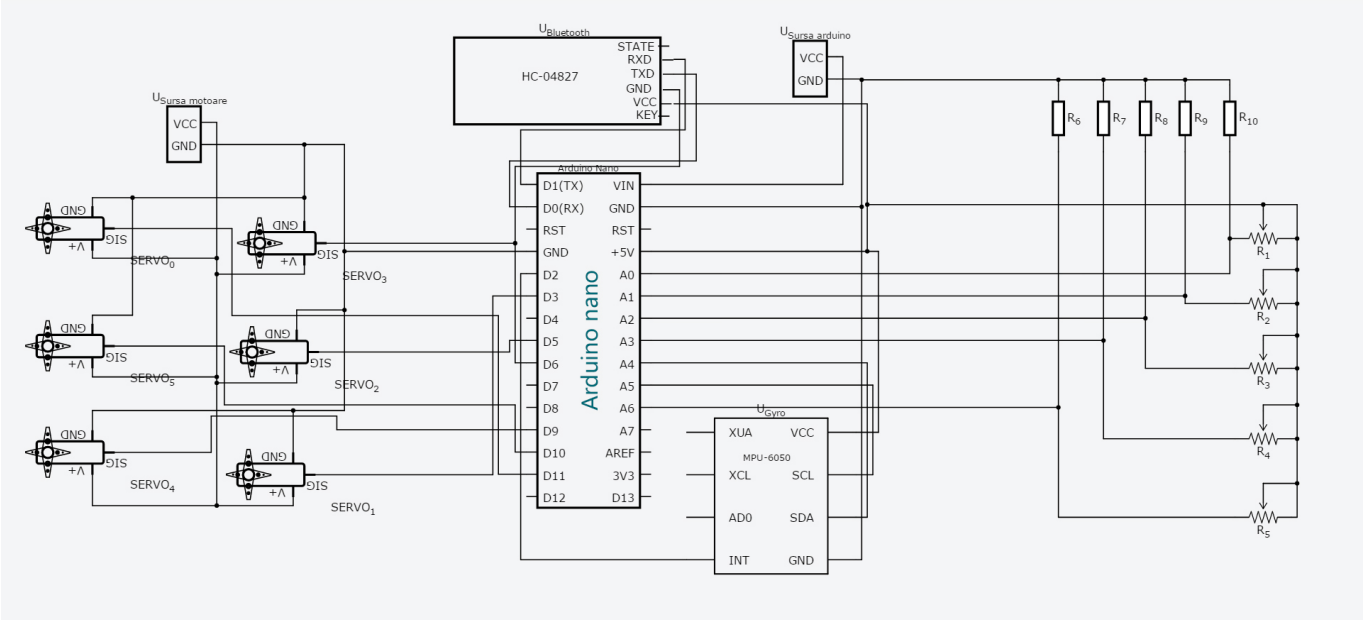


Hardware Design

Componente folosite:

- Modul microcontroler Arduino nano;

- Shield conectică Arduino nano;
- Unordered List ItemServomotoare PWM-MG 995 – 6 buc.
- Senzori flex rezistivi – 5 buc.
- Rezistori 10K – 5 buc.
- Accelerometru+Gyroscope MPU-6050;
- Bluetooth HC-05;
- Sursă CC 7,5V;
- Sursă CC 5V.



Software Design

Sistemul electronic are la bază un modul Arduino Nano (v3.0) – bazat pe microcontrolerul ATmega328, ce permite controlul şi comanda servomotoarelor cu funcţie PWM (Pulse Width Modulation). Servomotoarele sunt conectate pe ieşirile digitale PWM ale microcontrolerului (D3, D5, D6, D9, D10, D11). Traductorii de mişcare a degetelor – Senzorii de flexare rezistivi sunt conectaţi la microcontroler la intrările ADC (Analog to Digital Converter) A0, A1, A2, A3, A6, prin divizoare rezistive. Modulul accelerometru+giroscop este conectat la porturile SDA(A4), SCL(A5), iar Modulul de comandă Bluetooth la porturile digitale RX(D0), TX(D1).

Pentru a se realiza mişcarea cinematică, Modulul microcontroler (Arduino nano) – ce este controlat pe baza unui sketch Arduino (program C++ scris în memoria nevolatilă) – interpretează informaţia digitală preluată de la cei 5 senzorii de flexare şi de la senzorul accelerometru+gyroscope, şi o transmite prin comenzi PWM celor 6 servomotoare. Programul (sketch-ul Arduino) a fost scris folosind mediul de programare Arduino IDE, pe baza unui algoritm, care a rezultat în urma unor serii de teste: de calibrare a senzorilor, de stabilire a valorilor ce limitează intervalele de mişcare a servomotoarelor, de optimizare a programului ce controlează sistemul giroscopic+accelerometru şi de programare a modulul Bluetooth de comandă

Cod Arduino:

#include “I2Cdev.h” #include “MPU6050_6Axis_MotionApps20.h” #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE

```
  #include "Wire.h"
```

#endif MPU6050 mpu; #define OUTPUT_READABLE_YAWPITCHROLL #define LED_PIN 13 *(Arduino is 13, Teensy is 11, Teensy++ is 6) bool blinkState = false;* MPU control/status vars bool dmpReady = false; *set true if DMP init was successful uint8_t mpuIntStatus;* holds actual interrupt status byte from MPU uint8_t devStatus; *return status after each device operation (0 = success, !0 = error) uint16_t packetSize;* expected DMP packet size (default is 42 bytes) uint16_t fifoCount; *count of all bytes currently in FIFO uint8_t fifoBuffer[64];* FIFO storage buffer

orientation/motion vars Quaternion q; [w, x, y, z] quaternion container VectorInt16 aa; *[x, y, z] accel sensor measurements VectorInt16 aaReal;* [x, y, z] gravity-free accel sensor measurements VectorInt16 aaWorld; *[x, y, z] world-frame accel sensor measurements VectorFloat gravity;* [x, y, z] gravity vector float euler[3]; *[psi, theta, phi] Euler angle container float ypr[3];* [yaw, pitch, roll] yaw/pitch/roll container and gravity vector volatile bool mpuInterrupt = false; *indicates whether MPU interrupt pin has gone high void dmpDataReady() { mpuInterrupt = true; } #include <Servo.h>* create servo object to control a servo Servo myservo0; Servo myservo5; Servo myservo4; Servo myservo3; Servo myservo2; Servo myservo1; *analog pin used to connect flex sensor int potpin5 = 6; int potpin4 = 3; int potpin3 = 2; int potpin2 = 1; int potpin1 = 0;* variable to read the value from the analog pin int val0; int val5; int val4; int val3; int val2; int val1;

void setup() {

```
myservo0.attach(11);
myservo5.attach(10);
myservo4.attach(9);
myservo3.attach(6);
myservo2.attach(5);
myservo1.attach(3);
```

join I2C bus (I2Cdev library doesn't do this automatically) #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE Wire.begin(); TWBR = 24; 400kHz I2C clock (200kHz if CPU is 8MHz)

```
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
      Fastwire::setup(400, true);
  #endif

 Serial.begin(38400);
  while (!Serial); // wait for Leonardo enumeration, others continue
immediately
```

initialize device / Serial.println(F(“Initializing I2C devices…”));

mpu.initialize(); *verify connection* / Serial.println(F(“Testing device connections…”)); / *Serial.println(mpu.testConnection() ? F(“MPU6050 connection successful”) : F(“MPU6050 connection failed”));* wait for ready *Serial.println(F(“\nSend any character to begin DMP programming and demo: ”));* / while (Serial.available() && Serial.read()); *empty buffer* while (!Serial.available()); *wait for data* while (Serial.available() && Serial.read()); *empty buffer again* load and configure the DMP

```
///  Serial.println(F("Initializing DMP..."));
  devStatus = mpu.dmpInitialize();
```

supply your own gyro offsets here, scaled for min sensitivity mpu.setXGyroOffset(220); mpu.setYGyroOffset(76); mpu.setZGyroOffset(-85); mpu.setZAccelOffset(1788); 1688 factory default for my test chip

```
  // make sure it worked (returns 0 if so)
  if (devStatus == 0) {
      // turn on the DMP, now that it's ready
      ///  Serial.println(F("Enabling DMP..."));
      mpu.setDMPEnabled(true);
      // enable Arduino interrupt detection
      ///  Serial.println(F("Enabling interrupt detection (Arduino external
interrupt 0)..."));
      attachInterrupt(0, dmpDataReady, RISING);
      mpuIntStatus = mpu.getIntStatus();
      // set our DMP Ready flag so the main loop() function knows it's okay
to use it
      ///    Serial.println(F("DMP ready! Waiting for first interrupt..."));
      dmpReady = true;
      // get expected DMP packet size for later comparison
      packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
      // ERROR!
      // 1 = initial memory load failed
      // 2 = DMP configuration updates failed
      // (if it's going to break, usually the code will be 1)
      ///     Serial.print(F("DMP Initialization failed (code "));
      ///     Serial.print(devStatus);
      ///     Serial.println(F(")"));
  }
```

```
  // configure LED for output
  pinMode(LED_PIN, OUTPUT);
```

```
}
```

void loop() {

```
  // if programming failed, don't try to do anything
  if (!dmpReady) return;
```

```
  // reset interrupt flag and get INT_STATUS byte
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();
```

```
  // get current FIFO count
  fifoCount = mpu.getFIFOCount();
```

```
  // check for overflow (this should never happen unless our code is too
inefficient)
  if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
```

```
      // reset so we can continue cleanly
      mpu.resetFIFO();
      /// Serial.println(F("FIFO overflow!"));
```

```
  // otherwise, check for DMP data ready interrupt (this should happen
frequently)
  } else if (mpuIntStatus & 0x02) {
      // wait for correct available data length, should be a VERY short wait
      while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
      // read a packet from FIFO
      mpu.getFIFOBytes(fifoBuffer, packetSize);

      // track FIFO count here in case there is > 1 packet available
      // (this lets us immediately read more without waiting for an
interrupt)
      fifoCount -= packetSize;
```

```
      #ifdef OUTPUT_READABLE_YAWPITCHROLL
          // display Euler angles in degrees
          mpu.dmpGetQuaternion(&q, fifoBuffer);
          mpu.dmpGetGravity(&gravity, &q);
          mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
          // Serial.print("ypr\t");
         // Serial.print(ypr[0] * 180/M_PI);
         // Serial.print("\t");
         //  Serial.print(ypr[1] * 180/M_PI);
        ///  Serial.print("\t");
        ///  Serial.println(ypr[2] * 180/M_PI);
      #endif
```

```
      // blink LED to indicate activity
      blinkState = !blinkState;
      digitalWrite(LED_PIN, blinkState);
  }
// END GYRO

val5 = analogRead(potpin5);            // reads the value of the flex sensor
(value between 0 and 1023)
val4 = analogRead(potpin4);
val3 = analogRead(potpin3);
val2 = analogRead(potpin2);
val1 = analogRead(potpin1);
val0 = (map(ypr[2] * 180/M_PI, -65, 5, 40, 150));
```

```
int valf0 = constrain(val0, 40, 150);
int valf5 = map(val5, 450, 230, 155, 85);
int valf4 = map(val4, 465, 240, 160, 70);
int valf3 = map(val3, 430, 240, 150, 60);
int valf2 = map(val2, 445, 220, 180, 80);
int valf1 = map(val1, 500, 320, 20, 115);
```

```
valf5 = constrain(val5, 95, 155);
valf4 = constrain(val4, 70, 160);
valf3 = constrain(val3, 60, 150);
valf2 = constrain(val2, 80, 180);
valf1 = constrain(val1, 20, 115);
```

```
myservo0.write(valf0);
myservo5.write(valf5);                  // sets the servo position according
to the scaled value
myservo4.write(valf4);
myservo3.write(valf3);
myservo2.write(valf2);
myservo1.write(valf1);
```

Serial.print("I received gyro: "); Serial.println(val5); *Serial.println("I received servo: ");* Serial.println(valf5); *delay(150);* waits for the servo to get there }

Concluzii

Proiectul m-a ajutat sa dobandesc cunostinte avansate de lucru cu Arduino.

Bibliografie/Resurse

- http://www.arduino.org/learning/reference
- https://inmoov.fr/
- https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/MPU6050_6Axis_MotionApps20.h

Export to PDF