Smart Tennis Racket

Introducere

Ce face

Smart Tennis Racket este un dispozitiv electronic montabil pe racheta de tenis care detectează automat loviturile și oferă feedback obiectiv jucătorului în timp real. Concret, sistemul:

- detectează startul și sfârșitul unei lovituri din semnătura inerțială a swing-ului;
- estimează **intensitatea loviturii** — *slab* / *mediu* / *puternic* — din vârful de accelerație;
- estimează **viteza unghiulară maximă** a swing-ului prin integrarea giroscopului;
- afișează un **contor de lovituri** și statistici live pe un display TFT 1.8”;
- oferă **feedback vizual instant** printr-un LED RGB (verde = lovitură bună, roșu = slabă);
- permite reset printr-un buton fizic.

Scop

Antrenament obiectiv pentru jucătorii amatori. Autoevaluarea unui jucător amator se bazează pe simțuri (“a sunat bine”, “a fost puternic”), nu pe date măsurate. Dispozitivul transformă o ședință de antrenament într-un set de metrici concrete: număr lovituri, raport forehand/backhand, distribuția intensităților, viteză medie de swing — fără cost ridicat și fără dependență de un antrenor sau de o aplicație telefonică.

Ideea de pornire

Soluțiile comerciale (Babolat Pop, Sony Smart Tennis Sensor, Zepp Tennis) sunt scumpe, închise ca platformă și au fost discontinuate de mai multe ori, lăsând utilizatorii fără suport. Plecând de la observația că un IMU 6-DOF de tipul MPU6050 acoperă cea mai mare parte a senzoricii necesare, ne-am propus să construim un echivalent open, ieftin și deschis pentru extensii.

Utilitate

- feedback obiectiv în timp real, fără telefon și fără cont online;
- statistici per ședință, vizibile direct pe ecranul atașat de rachetă;
- alimentare autonomă pe baterie 9V → uzabil pe teren.

Descriere generală

Sistemul este construit în jurul plăcii de dezvoltare **ATmega328P Xplained Mini**, care joacă rolul de coordonator central. Microcontrollerul citește continuu date inerțiale de la senzorul MPU6050 prin magistrala **I2C**, le procesează printr-o pipeline simplă de detecție și clasificare, după care actualizează simultan două canale de output: ecranul TFT (text și statistici) prin **SPI** și LED-ul RGB (feedback luminos) prin trei canale **PWM**. Un buton conectat ca GPIO permite reset-ul contorului.

Schemă bloc

- **MPU6050 ⇄ ATmega328P** — *I2C, 100 kHz*: la fiecare ~10 ms, MCU-ul (master) citește 14 octeți (3× accel + temp + 3× gyro) de la slave-ul MPU6050 (adresa 0x68). Sunt folosite primitivele `twi_start` / `twi_write` / `twi_read_ack` / `twi_read_nack` / `twi_stop` din driverul de la Lab 6.
- **ATmega328P → modul detecție swing** *(software)*: un filtru pe magnitudinea accelerației (||a|| − g) detectează startul și sfârșitul unei lovituri.
- **ATmega328P → modul clasificare** *(software)*: pe baza semnului ω_z (giroscop, axa verticală a încheieturii) și a vârfului de accelerație, decide forehand/backhand și slab/mediu/puternic.
- **ATmega328P → ST7735 TFT 1.8”** — *SPI, master → slave*: după fiecare lovitură detectată, MCU-ul redesenează zona de text și afișează:
  - `HITS: 24`
  - `SHOT: FOREHAND`
  - `POWER: HIGH`
  - `SWING: 320 deg/s`
- **ATmega328P → LED RGB (R+G)** — *PWM*: 2 canale PWM independente, generate de Timer 0 (OC0B = verde) și Timer 2 (OC2B = roșu), modulează intensitatea celor două culori. LED-ul are anod comun → driver-ul software inversează duty-cycle-ul (0 = stins, 255 = maxim).
- **Buton → ATmega328P** — *GPIO + INT0*: tratat în întrerupere externă pe PD2, cu debouncing software. La apăsare = reset contor lovituri.

Module software

| Modul | Sursă | Descriere |

`twi.c/.h`	Lab 6 (preluat, F_CPU adaptat la 16 MHz)	driver I2C low-level
`mpu6050.c/.h`	**scris** (după pattern-ul `mpl3115a2` din Lab 6)	init, citire accel/gyro burst, calibrare offset
`spi.c/.h`	Lab 5 (preluat)	driver SPI low-level
`st7735.c/.h`	Lab 5 (preluat, adaptat pentru pini noi)	driver TFT 1.8”

`timers.c/.h`	Lab 3 (preluat, extins)	configurare PWM 2 canale pe Timer 0 + Timer 2
`rgb.c/.h`	**scris**	wrapper PWM pentru LED RGB (R+G) cu efecte (solid / blink / fade)
`swing_detect.c/.h`	**scris**	detecție swing și clasificare
`main.c`	**scris**	bucla principală, integrare module, FSM

Hardware Design

Hardware Design

1. Componentele folosite și rolul lor în proiect

- **ATmega328P Xplained Mini** — placa de dezvoltare cu microcontrolerul ATmega328P (AVR 8-bit, 16 MHz). Are debugger/programator integrat (mEDBG) și port serial virtual (CDC), deci poate fi programată și depanată direct prin USB, fără programator extern. Rolul ei este de **coordonator central**: citește datele de mișcare de la senzor, le procesează, comandă display-ul și LED-ul și tratează apăsarea butonului.
- **MPU6050 (modul GY-521)** — senzor inerțial 6-DOF (accelerometru 3 axe + giroscop 3 axe, 16-bit fiecare). Comunică prin **I2C** la adresa `0x68`. Rolul lui: **achiziția semnăturii inerțiale a loviturii** — accelerometrul detectează startul și sfârșitul unei lovituri prin praguri de magnitudine, iar giroscopul oferă sensul de rotație pentru clasificarea forehand/backhand.
- **Display TFT 1.8” ST7735** — ecran color 128×160 pixeli, control prin **SPI**. Rolul lui: **afișarea informațiilor în timp real** — contor de lovituri, tipul loviturii, intensitatea și viteza unghiulară maximă.
- **LED RGB 5 mm cu anod comun** — folosim canalele Roșu și Verde. Rolul lui: **feedback vizual instant** — verde pentru o lovitură puternică, roșu pentru o lovitură slabă. Intensitatea fiecărei culori este controlată prin **PWM**.
- **Buton tactil 6×6×5 mm** — rolul lui: **resetarea contorului de lovituri**. Este tratat printr-o întrerupere externă pe linia INT0.
- **Componente pasive și de suport** — rezistori de 220 Ω pentru limitarea curentului prin LED, breadboard MB-102 pentru prototipare, fire de legătură, suport de baterie 9V pentru alimentarea autonomă (în faza de dezvoltare alimentarea se face prin USB).

2. Pinii folosiți și justificarea alegerilor

Microcontrolerul ATmega328P are un număr limitat de pini cu funcții speciale (I2C, SPI, PWM, întreruperi). Alegerea fiecărui pin nu este arbitrară, ci impusă de funcția hardware de care avem nevoie.

Semnal	Pin AVR	Componentă	Justificarea alegerii
SDA	PC4	MPU6050	Pin hardware TWI al ATmega328P — singura opțiune pentru I2C hardware.

SCL	PC5	MPU6050	Pin hardware TWI — pereche cu SDA.
SCK	PB5	Display	Pin hardware SPI SCK.
MOSI (SDA)	PB3	Display	Pin hardware SPI MOSI.
CS	PB2	Display	Pinul hardware SS; trebuie configurat ca ieșire, altfel modulul SPI cade în mod slave.
DC	PB1	Display	GPIO liber, adiacent fizic pinilor SPI — cablaj scurt.
RST	PB0	Display	GPIO liber, adiacent pinului DC.
Rosu (R)	PD3	LED RGB	Ieșire OC2B — PWM hardware generat de Timer 2.
Verde (G)	PD5	LED RGB	Ieșire OC0B — PWM hardware generat de Timer 0.
Buton	PD2	Buton	Pin INT0 — întrerupere externă dedicată, cu pull-up intern activat în software.

De ce I2C pe PC4/PC5: magistrala I2C hardware (TWI) a ATmega328P este fixată pe acești doi pini. Nu există alternativă dacă vrem comunicare I2C hardware, fără emulare software.

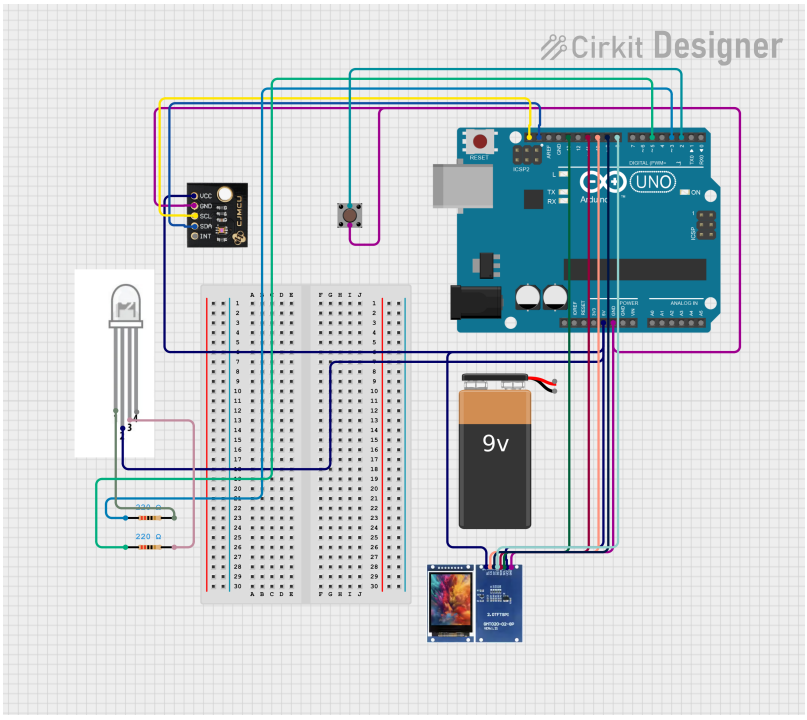
De ce SPI pe portul B: modulul SPI hardware folosește obligatoriu PB3 (MOSI) și PB5 (SCK). Pinii de control specifici ST7735 (CS, DC, RST) pot fi GPIO oarecare; i-am ales pe PB0–PB2 pentru a fi grupați lângă pinii SPI, ceea ce scurtează cablajul și reduce riscul de erori.

De ce PWM pe PD3 și PD5: pe ATmega328P, pinul PD3 este ieșirea comparatorului OC2B (Timer 2), iar PD5 este OC0B (Timer 0). Acestea sunt ieșirile de PWM hardware disponibile, care generează semnalul fără a încărca procesorul.

De ce butonul pe PD2: pinul PD2 corespunde întreruperii externe INT0. Resetarea contorului este un eveniment punctual, ideal de tratat într-o rutină de întrerupere, fără a fi nevoie de verificare repetată (polling) în bucla principală.

Lipsa conflictelor: cele trei magistrale folosesc porturi diferite — I2C pe portul C, SPI pe portul B, PWM-ul LED-ului pe portul D — deci nu există suprapuneri. Singura observație: PB5 este folosit și de LED-ul utilizator de pe placă, care va pâlpâi în timpul transmisiilor SPI; acest efect este pur cosmetic și nu afectează funcționarea.

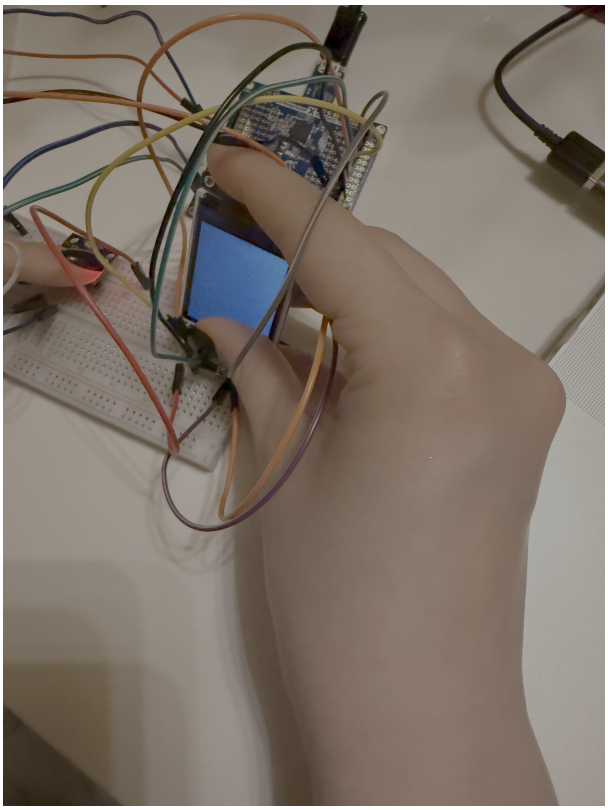
3. Schema electrică



4. Imagini și dovezi de funcționare

Montajul complet pe breadboard

Imaginea arată sistemul asamblat pe breadboard: placa ATmega328P Xplained Mini alimentată prin USB, senzorul MPU6050, display-ul ST7735, LED-ul RGB cu cei doi rezistori de limitare și butonul de reset. Alimentarea este distribuită prin șinele laterale ale breadboard-ului.



Listă de piese

^ # ^ Componentă ^ Cantitate ^ Rol în proiect ^

1	ATmega328P Xplained Mini (placă dezvoltare)	1	coordonator central, programator on-board
2	MPU6050 (modul GY-521) — accelerometru + giroscop 3 axe	1	senzor inerțial pentru detecția mișcării (I2C, adresa 0x68)
3	LED RGB 5 mm, 4 pini, anod comun, opac	5	feedback vizual (PWM, doar canalele R+G); restul ca rezervă
4	Display TFT color 1.8”, controller ST7735, interfață SPI	1	afișare statistici și tip lovitură
5	Kit rezistențe 1/4 W, 30 valori, 600 buc, 10 Ω – 1 MΩ	1	limitare curent LED (2× 220 Ω pe R și G); pull-up I2C dacă e necesar
6	Breadboard MB-102, 830 puncte	1	suport prototipare
7	Fire dupont 20 cm, mamă-mamă, set 40	1	conexiuni breadboard ↔ Xplained Mini ↔ module
8	Fire dupont 20 cm, tată-mamă, set 40	1	conexiuni mixte (pini Xplained ↔ module)
9	Suport baterie 9V cu jack 5.5×2.1 mm	1	alimentare portabilă
10	Buton tactil PCB mini 6×6×5, 2 pini	1	reset contor lovituri

Software Design

Mediu de dezvoltare

Proiectul a fost dezvoltat folosind PlatformIO cu toolchain-ul avr-gcc, integrat în Visual Studio Code. Compilatorul țintă este `avr-gcc` cu flagul `-mmcu=atmega328p` și `-DF_CPU=16000000UL`. Deoarece codul folosește `sqrt()` din `<math.h>`, au fost adăugate flagurile de linker pentru floating point:

```
build_flags = -Wl,-u,vfprintf -lprintf_flt -lm
```

Depanarea s-a realizat prin interfața UART la 9600 baud, cu mesaje trimise către un terminal serial (Serial Monitor din PlatformIO sau PuTTY).

Biblioteci și surse folosite

Modul	Fișier sursă	Origine	Rol
Driver I2C/TWI	`twi.c` / `twi.h`	Preluat din Laboratorul 6, adaptat (F_CPU = 16 MHz, timeout adăugat)	Comunicare I²C cu MPU6050
Driver SPI	`spi.c` / `spi.h`	Preluat din Laboratorul 5	Comunicare SPI cu display-ul ST7735

Driver TFT	`st7735.c` / `st7735.h`	Preluat din Laboratorul 5, adaptat pentru pinii ATmega328P Xplained Mini	Afișare text și grafice pe ecranul 1.8”
Loop principal + logică	`main.c`	Scris integral în cadrul proiectului	Inițializare, FSM detecție swing, clasificare, UART, LED RGB, buton
AVR LibC	`<avr/io.h>`, `<avr/interrupt.h>`, `<util/delay.h>`, `<math.h>`	Standard AVR-LibC	Acces registre, întreruperi, delay, sqrt()

Nu au fost folosite biblioteci externe de tip Arduino sau framework-uri de nivel înalt — întreaga stivă este bare-metal C pe AVR.

Algoritmi și structuri implementate

1. Citirea senzorului MPU6050 prin I²C (burst read)

Funcția `mpu_read_gyro()` realizează o citire în rafală (burst read) a 6 octeți consecutivi începând de la registrul `0x43` (GYRO_XOUT_H). Secvența respectă protocolul I²C descris în Laboratorul 6:

```
twi_start(SLA_W) → twi_write(REG_GYRO_X) →
twi_start(SLA_R) → twi_read(ACK) x5 → twi_read(NACK) → twi_stop()
```

Cei 6 octeți sunt recombinați în trei valori `int16_t` (gx, gy, gz) prin shift și OR pe biți:

```
*gx = (int16_t)((b[0] << 8) | b[1]);
```

2. Calculul magnitudinii vectorului giroscopic

Pentru a obține un scalar independent de orientarea rachetei, se calculează norma euclidiană a vectorului angular:

```
mag = sqrt(gx² + gy² + gz²)
```

Aceasta este implementată în `gyro_magnitude()` folosind `sqrt()` din `<math.h>` aplicat pe `double`. Valorile intermediare sunt calculate pe `int32_t` pentru a evita overflow-ul pe 16 biți:

```
int32_t sx = (int32_t)gx * gx;
```

Avantajul magnitudinii față de o singură axă este robustețea la montaj: indiferent cum este fixat senzorul pe rachetă, o mișcare rapidă va produce o valoare mare.

3. Mașina de stări pentru detecția swing-ului (histerezis)

Detecția loviturii folosește un filtru cu histerezis pe două praguri, pentru a evita declanșările false datorită zgomotului sau vibrațiilor mici:

Starea IDLE: dacă `mag > GYRO_THR_START` (5000), se intră în starea `IN_HIT` și se înregistrează primul vârf. Starea IN_HIT: se urmărește valoarea maximă (`peak_gyro`). Dacă `mag < GYRO_THR_END` (1500), lovitura s-a terminat → clasificare și reset. Cooldown: după fiecare lovitură detectată, un contor de 1500 ms blochează detecțiile noi, pentru a lăsa timp rachetei să revină la poziția inițială fără a genera lovituri fantomă.

Separarea pragului de intrare (5000) față de cel de ieșire (1500) formează un gap de histerezis de ~3500 unități, eliminând oscilațiile în jurul pragului.

4. Clasificarea loviturii

La finalul unui swing, valoarea `peak_gyro` este comparată cu pragul `GYRO_GOOD_THR` (20000):

```
if (peak_gyro >= GYRO_GOOD_THR) → BUNA (strong_hits++)
else                            → SLABA (weak_hits++)
```

5. Feedback vizual prin LED RGB — degradeu roșu→verde

Funcția `led_from_peak()` mapează `peak_gyro` din intervalul [GYRO_MIN, GYRO_MAX] = [5000, 30000] pe un factor `f ∈ [0, 255]`:

```
f = (peak - GYRO_MIN) * 255 / (GYRO_MAX - GYRO_MIN)
green = f;   red = 255 - f;
```

Valorile sunt scalate la 80% din maxim (`* 4 / 5`) pentru a nu orbi utilizatorul. LED-ul are anod comun, deci duty cycle-ul este inversat: `OCRxB = 255 - valoare`, conform macrourilor `LED_SET_R()` și `LED_SET_G()`. Semnalele PWM hardware sunt generate de Timer 0 (OC0B → verde, PD5) și Timer 2 (OC2B → roșu, PD3), fără intervenție software în buclă — exact modul Fast PWM studiat în Laboratorul 3.

6. Debouncing software pentru buton (INT0)

Butonul pe PD2 este tratat în întreruperea externă `INT0_vect`. Debouncing-ul se realizează prin compararea unui contor global de tick-uri (`global_tick`, incrementat la fiecare 20 ms în bucla principală) cu momentul ultimei apăsări valide:

```
if ((uint16_t)(global_tick - last_press_tick) > 3)  // > 3 × 20ms = 60ms
```

Această abordare non-blocantă este echivalentă cu fereastra de 30–50 ms recomandată în Laboratorul 2, fără a bloca execuția din `main`.

7. Afișare pe TFT ST7735 prin SPI

Driver-ul `st7735.c` folosește interfața SPI hardware (PB5=SCK, PB3=MOSI, PB2=CS) cu pini de control GPIO (PB0=DC, PB1=RST). Fontul 5×7 este stocat în memoria Flash (`PROGMEM`) și citit cu `pgm_read_byte()`, economisind RAM-ul limitat al ATmega328P. Actualizarea ecranului se face selectiv — doar zona numerelor (coordonate fixe) — pentru a minimiza traficul SPI și latența vizibilă.

Rezultate Obținute

Stadiul actual al implementării software

Implementarea software este completă și funcțională. Toate modulele sunt integrate și validate pe hardware real:

Inițializare și citire continuă MPU6050 prin I²C la 100 kHz Detecție automată lovituri cu histerezis și cooldown Clasificare binară slab/bun bazată pe peak-ul giroscopului Afișare live pe TFT ST7735 (total, slabe, bune) Feedback LED RGB cu degradeu roșu→verde proporțional cu intensitatea Reset prin buton fizic cu debouncing software Logging serial UART la 9600 baud pentru depanare și calibrare

Motivația alegerii bibliotecilor

S-a optat deliberat pentru o arhitectură bare-metal, fără Arduino sau alte framework-uri de nivel înalt, din următoarele motive:

Control precis al resurselor hardware: ATmega328P are 2 KB RAM și 32 KB Flash. Un framework de tip Arduino adaugă sute de octeți overhead pentru funcționalități nefolosite. Familiaritate cu laboratoarele cursului: driver-ele `twi.c`, `spi.c` și `st7735.c` sunt extensii directe ale scheletelor din Laboratoarele 5 și 6, asigurând o înțelegere profundă a protocolelor. Latență determinabilă: în sisteme de detecție în timp real, buclele de polling cu `_delay_ms()` și întreruperile hardware oferă timinguri predictibile, spre deosebire de un RTOS sau scheduler complex.

Singura funcție din bibliotecă externă (AVR-LibC) cu potențial overhead este `sqrt()` pe `double` — acceptabil deoarece este apelată o singură dată per eșantion (la 20 ms), deci ~50 apeluri/secundă, neglijabil față de ciclurile CPU disponibile la 16 MHz.

Elementul de noutate

Față de implementările academice tipice care testează un singur senzor izolat, acest proiect integrează simultan patru periferice hardware diferite pe același microcontroller (I²C + SPI + PWM × 2 + UART + GPIO interrupt), cu o logică de procesare a semnalului în timp real care produce output multimodal (vizual + serial) în același ciclu de 20 ms. Degradeul de culoare continuu al LED-ului (nu simplu roșu/verde binar) și algoritmul de histerezis cu cooldown adaptiv sunt elemente care aduc valoare practică față de o implementare banală cu prag simplu.

Justificarea utilizării funcționalităților din laborator

Funcționalitate	Laborator sursă	Utilizare în proiect
UART / printf redirecționat	Laboratorul 1	Logging serial: magnitudine giroscop, tip lovitură, contor — esențial pentru calibrarea pragurilor
Întrerupere externă INT0 + debouncing	Laboratorul 2	Reset contor prin buton fizic, non-blocant
PWM hardware Fast PWM, Timer 0 + Timer 2	Laboratorul 3	Control intensitate LED RGB cu anod comun pe două canale independente
SPI hardware + driver ST7735	Laboratorul 5	Afișare statistici live pe TFT 1.8” color
I²C / TWI + citire registre senzor	Laboratorul 6	Citire burst gyroscop MPU6050, același pattern ca MPL3115A2 din laborator

Scheletul proiectului și interacțiunea dintre funcționalități

Bucla principală din `main()` rulează la o perioadă fixă de SAMPLE_MS = 20 ms și urmează un pipeline liniar:

```
[Buton ISR] ──────────────────────────────────► reset_flag
                                                    │
main loop (20ms tick):                              ▼
  1. Verifică reset_flag → resetează contoare și ecran
  2. mpu_read_gyro() ──I²C──► MPU6050 ──► gx, gy, gz
  3. gyro_magnitude() ──────────────────► mag (int32_t)
  4. FSM histerezis:
       IDLE ──mag > 5000──► IN_HIT (track peak)
       IN_HIT ──mag < 1500──► clasificare + LED + ecran update
  5. led_from_peak() ──PWM──► Timer0/Timer2 ──► LED RGB
  6. screen_update_counts() ──SPI──► ST7735
  7. Debug UART la fiecare 25 iterații (~500 ms)
  8. _delay_ms(20) + global_tick++
```

Validarea funcționării s-a realizat în două etape:

Validare prin UART: cu senzorul în repaus, mesajele `gyro_mag=XXX` confirmau valori < 500. Scuturând senzorul brusc, valorile depășeau 10000–30000, confirmând că pragurile GYRO_THR_START = 5000 separă corect zgomotul de fundal de un swing real. Validare vizuală: LED-ul schimba culoarea consistent cu intensitatea mișcării (roșu pentru mișcări ușoare, verde/galben pentru swing-uri puternice), iar ecranul TFT incrementa contorul corect la fiecare lovitură detectată.

Calibrarea senzorului

MPU6050 a fost folosit în configurația implicită (fără calibrare offset a giroscopului), deoarece:

Detecția se bazează pe magnitudini relative (este swing vs. nu este swing), nu pe valori absolute. Driftul de offset al giroscopului (tipic 1–3 LSB/°C) este neglijabil față de valorile de 5000–30000 atinse în swing. Pragurile au fost determinate empiric prin UART logging: sistemul a fost conectat la un terminal serial și senzorul a fost mișcat cu intensități variate. S-au observat valorile tipice:

Repaus / vibrații mici: mag ≈ 100–400 Mișcare lentă a mâinii: mag ≈ 800–2000 Swing moderat: mag ≈ 5000–12000 Swing puternic: mag ≈ 15000–28000

Pe baza acestor observații s-au stabilit: GYRO_THR_START = 5000, GYRO_THR_END = 1500, GYRO_GOOD_THR = 20000.

Optimizări realizate

Optimizare	Unde	Motivație
Font stocat în Flash (PROGMEM)	`st7735.c`	Fontul 5×7 ASCII ocupă 475 octeți; dacă ar fi în RAM, ar consuma ~23% din cei 2KB disponibili
Actualizare selectivă a ecranului	`screen_update_counts()`	Se rescrie doar zona cu numere (3 stringuri de 5 caractere), nu întregul ecran; reduce traficul SPI cu ~95% față de `st7735_fill()` complet
Padding cu spații în `sprintf`	`main.c`	`sprintf(buf, "%-5u", val)` suprascrie cifrele vechi fără a șterge și redesena fundalul — evită flickering
Histerezis cu cooldown	`main.c`	Fără cooldown, un singur swing ar genera 5–10 detecții false; cooldown-ul de 1500 ms garantează maximum 1 lovitură per swing
Calcul magnitudine pe `int32_t`	`gyro_magnitude()`	Produsele `gx*gx` pe `int16_t` ar produce overflow (max $32767^2 = 1.07\times10^9 > 32767$); cast la `int32_t` înainte de înmulțire previne silently wrong results
Debouncing non-blocant	`ISR(INT0_vect)`	Compararea tick-urilor nu blochează ISR-ul și nici bucla principală, față de un `_delay_ms()` în ISR care ar dezactiva alte întreruperi

Script demo video

demo_racheta.mp4.zip

Concluzii

Download

smart_tennis_racket.zip

Bibliografie/Resurse

Resurse hardware

- Datasheet ATmega328P — Microchip
- MPU-6000/MPU-6050 Register Map and Descriptions — InvenSense
- MPU-6000/MPU-6050 Product Specification — InvenSense
- ST7735 datasheet — Sitronix
- ATmega328P Xplained Mini User Guide — Microchip

Resurse software

- Laborator 1 (USART) — https://ocw.cs.pub.ro/courses/pm/lab/lab1-2023-2024
- Laborator 2 (Întreruperi & Timere) — https://ocw.cs.pub.ro/courses/pm/lab/lab2-2023-2024
- Laborator 3 (Timere & PWM) — https://ocw.cs.pub.ro/courses/pm/lab/lab3-2023-2024
- Laborator 5 (SPI & ST7735) — https://ocw.cs.pub.ro/courses/pm/lab/lab5-2023-2024
- Laborator 6 (I2C) — https://ocw.cs.pub.ro/courses/pm/lab/lab6-2023-2024

Lucrări conexe

- jrowberg / i2cdevlib — bibliotecă C++ MPU6050 (referință pentru registre): https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050
- “Tennis Stroke Recognition Using Inertial Data” — articole academice pe IEEE Xplore.

Export to PDF