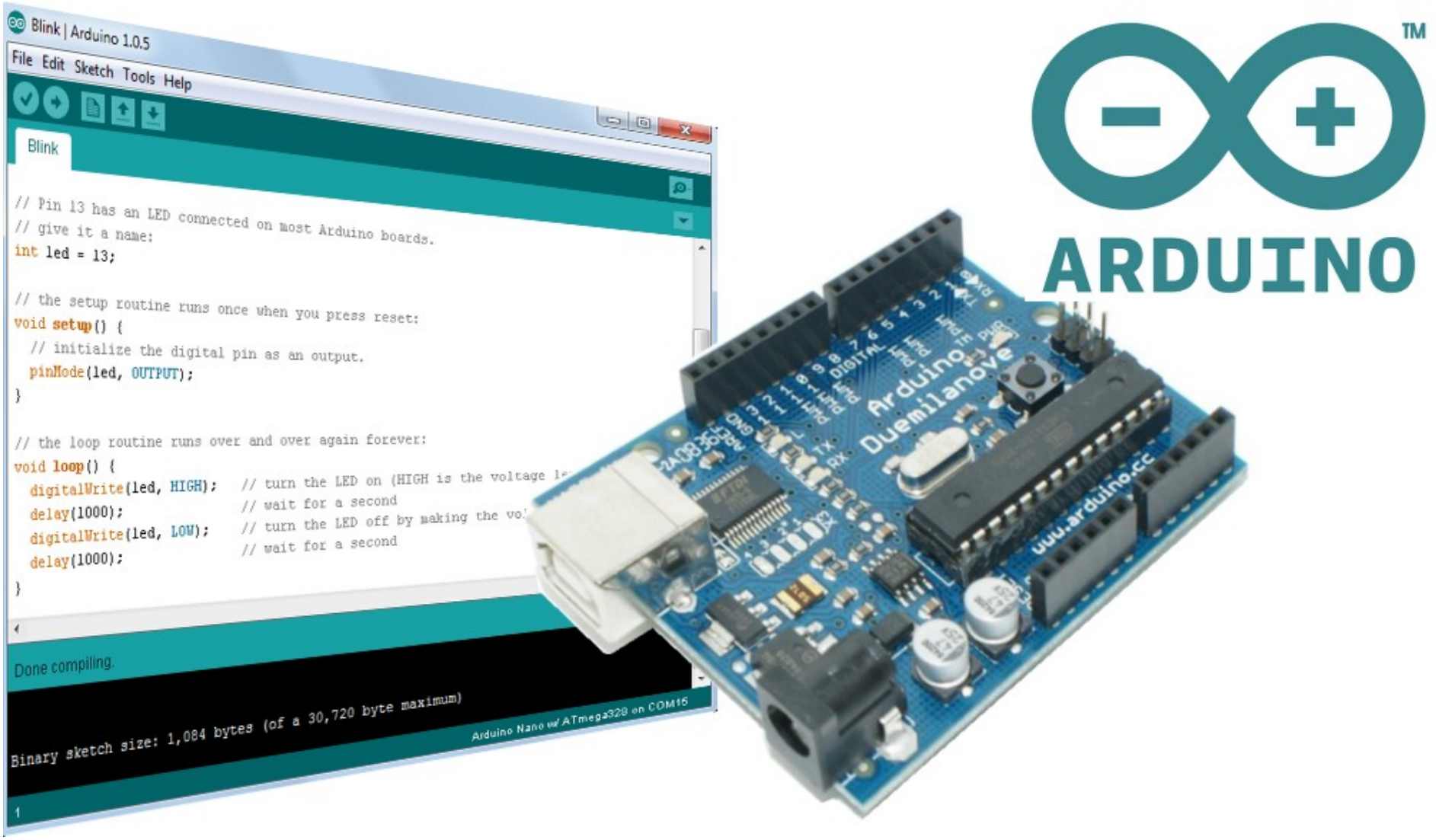
Arduino tanfolyam kezdőknek és haladóknak



12. Szervók, Processing IDE, ultrahangos „radar”

Ajánlott irodalom

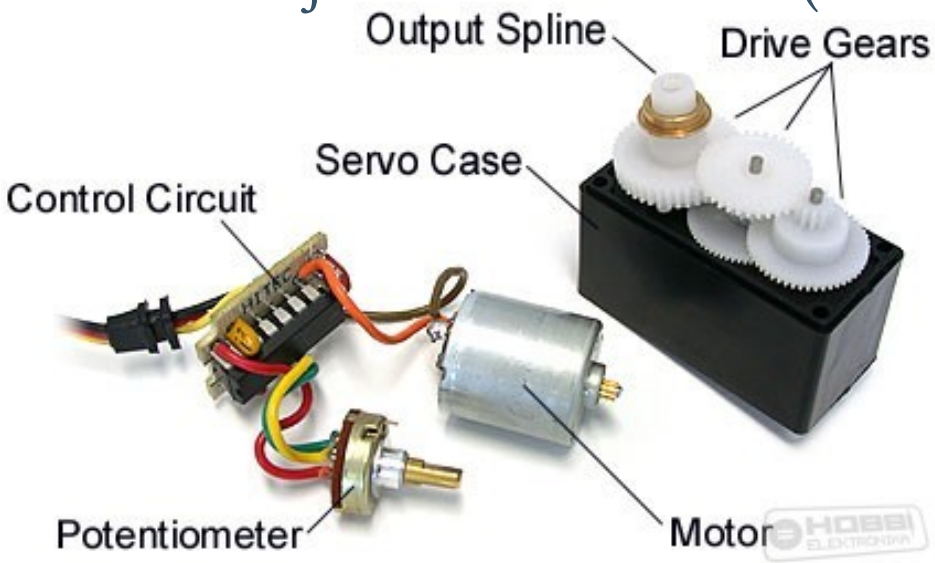
- Aduino LLC.: Arduino Language Reference
- ATMEL: ATmega328p mikrovezérlő adatlapja
- Brian W. Kernighan, Dennis Ritchie: A C programozási nyelv
- Cseh Róbert: Arduino programozási kézikönyv
- Harsányi Réka – Juhász Márton András:
  Fizikai számítástechnika: elektronikai alapok és Arduino programozás
- Ruzsinszki Gábor: Mikrovezérlős rendszerfejlesztés C/C++ nyelven I. – PIC mikrovezérlők
- Ruzsinszki Gábor: Mikrovezérlős rendszerfejlesztés C/C++ nyelven II. – Arduino

Arduino19_12 projektek

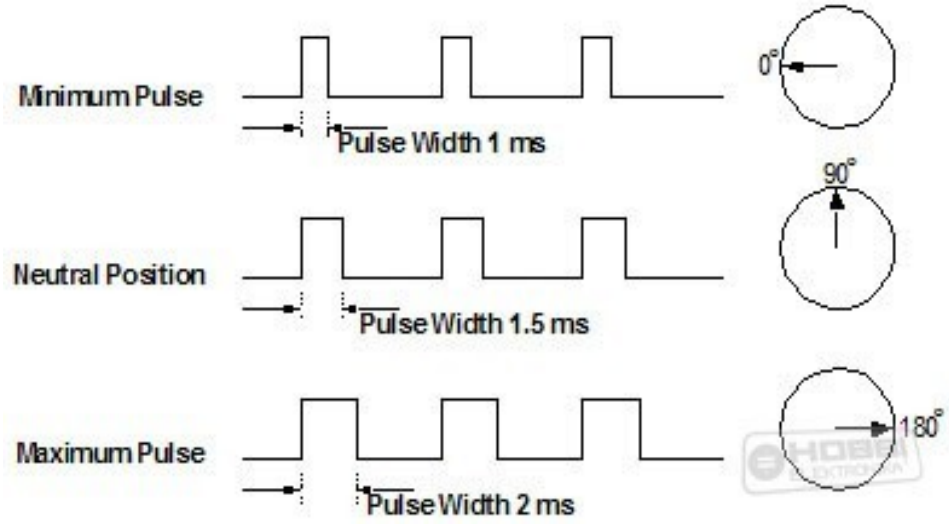
- **Servo_Sweep** – Egyszerű mintaprogram, amely a szervo motorral automatikusan oda-vissza végigpásztázza a 0-180 fokos tartományt
- **Servo_Knob** – „Gyári” mintaprogram, amelyben egy potméterrel vezéreljük a szervót
- **Servo_Set** – Mintaprogram, amelyben a soros portról vezéreljük a szervót
- **Walking_robot** – Egyszerű, két szervóval megvalósított lépegető robot működtető programja
- **Arduino_radar** – ultrahangos távolságmérőre alapozott „radar” program
- **Processing_radar** – Grafikus megjelenítő program a „radar”-hoz (ez a program a PC-n fut, **Processing** környezetben)

Szervo motorok

- A szervo egy pozícionálható motor, amely „ismeri” az aktuális pozícióját, és a cél pozíciót. Feladata, hogy az aktuális pozícióból a kívánt pozícióba álljon
- Felépítése: vezérlő áramkör, mechanikusan összekapcsolt motor és potenciométer (a potméterrel leosztott feszültség jelzi a pozíciót)
- Általában 180 °-os tartományban mozgatható, 1 – 2 ms szélességű, 50 Hz-es jellel vezérelhető (PWM)



Felépítés



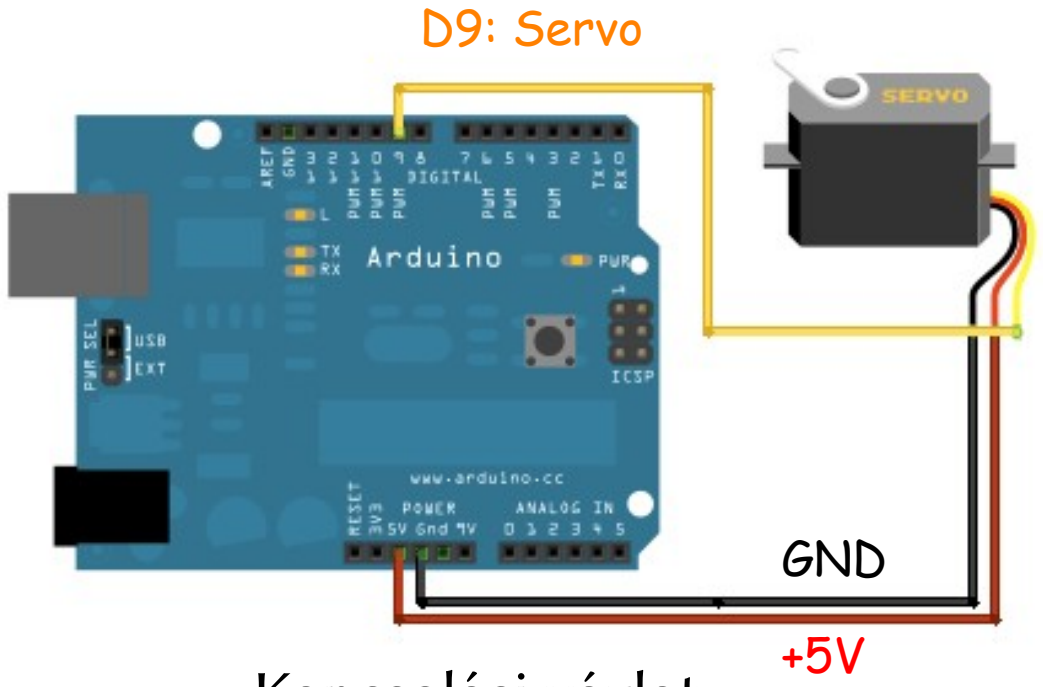
Jelalak

Servo programkönyvtár

- A beépített **Servo** programkönyvtár max. 8 db szervót kezel, s felhasználja a **Timer0** időzítőt, tehát az közben más célra nem használható.
- A programkönyvtár használatához be kell csatolni a **Servo.h** állományt és példányosítani kell a **Servo** osztályt (pl. `Servo myservo;`)
- **Servo.attach(***pin[, minvalue, maxvalue]***)** – hozzárendeli a **Servo** objektumot a megadott kivezetéshez. A mikroszekundumokban mért határértékek (minimális és maximális impulzusszélesség) megadása nem kötelező. Az alapértelmezett értékek: 544 és 2400 µs.
- **Servo.write(***adat***)** – a szervo beállítása (ha az adat < 200, akkor fokokban értendő, s a 0 – 180 tartományban vehet fel értéket, 200-nál nagyobb számok estén mikroszekundumban adható meg az impulzusszélesség)
- **Servo.writeMicroseconds(***adat***)** – a szervo beállítása (az adat itt mikroszekundumokban értendő, az inicializálásnál megadott *minvalue – maxvalue* tartományban)

Servo_Sweep

- Az Arduino IDE gyári mintaprogramja a szervo motorral automatikusan oda-vissza végigpásztázza a 0-180 fokos tartományt
- A **File/Examples/Servo/Sweep** menüben is elérhető programon csak a jobb érthetőség kedvéért változtattunk



Kapcsolási vázlat

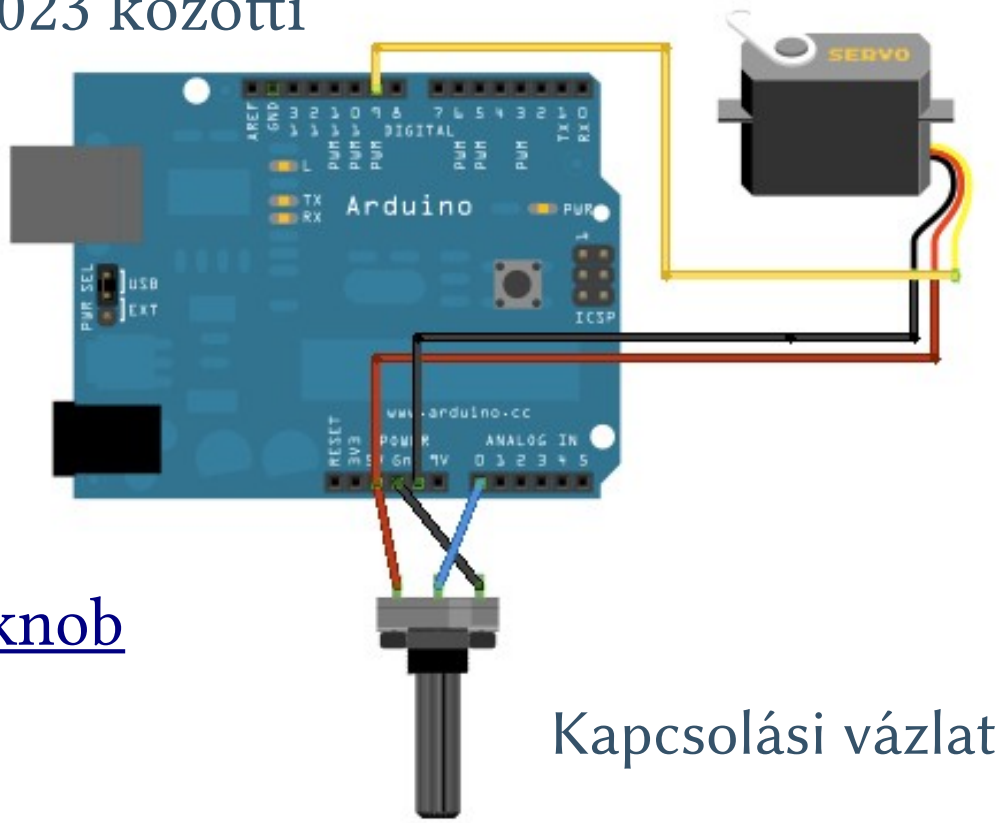
- Forrás: arduino.cc/en/tutorial/sweep

Servo_Sweep.ino

```
#include <Servo.h>
const int servoPin = 9;              // Szervo a D9 lábra
Servo servo;                         // Objektum példányosítása
int angle = 0;                       // Szervo pozíciója fokokban

void setup() {
  servo.attach(servoPin,640,2400); // Inicializálás határérték megadásával
}

void loop() {
  //--- Pásztázás 0-tól 180 fokig ------
  for(angle = 0; angle < 180; angle++) {
    servo.write(angle);
    delay(100);
  }
  //--- Pásztázás 180-tól 0 fokig ------
  for(angle = 180; angle > 0; angle--) {
    servo.write(angle);
    delay(100);
  }
}
```

Servo_Knob

- Ez a beépített mintaprogram (**File/Examples/Servo/Knob**) az **A0** bemenetre kötött potméterrel vezérli a szervót
- A 10 kΩ-os potméter csúszkáját az **A0** analóg bemenetre kötjük, a két végét padig **GND**-re, illetve az 5V-os tápfeszültségre
- Az **ADC** által visszaadott 0 – 1023 közötti számot át kell skálázni 1 – 180 közötti tartományba, ehhez a **map()** függvényt használjuk
- A szervo vezérléséhez a **Servo** könyvtárat használjuk
- Forrás: arduino.cc/en/tutorial/knob



Kapcsolási vázlat

Servo_Knob.ino

```
#include <Servo.h>
Servo myservo;                          // Objektum példányosítása
#define potpin 4                        // A potmétert az A0 lábra kötjük
int val;                                // Változó a feszültség beolvasásához

void setup() {
  myservo.attach(9,640,2400);           // Inicializálás határértékek megadásával
}                                       // Szervó a D9 lábra kötve

void loop() {
  val = analogRead(potpin);             // a potméter állásának beolvasása
  val = map(val, 0, 1023, 0, 179);      // a kapott szám átskálázása (0-180)
  myservo.write(val);                   // a szervo beállítása
  delay(100);                           // várunk, hogy a szervo beálljon
}
```

Servo_Set.ino

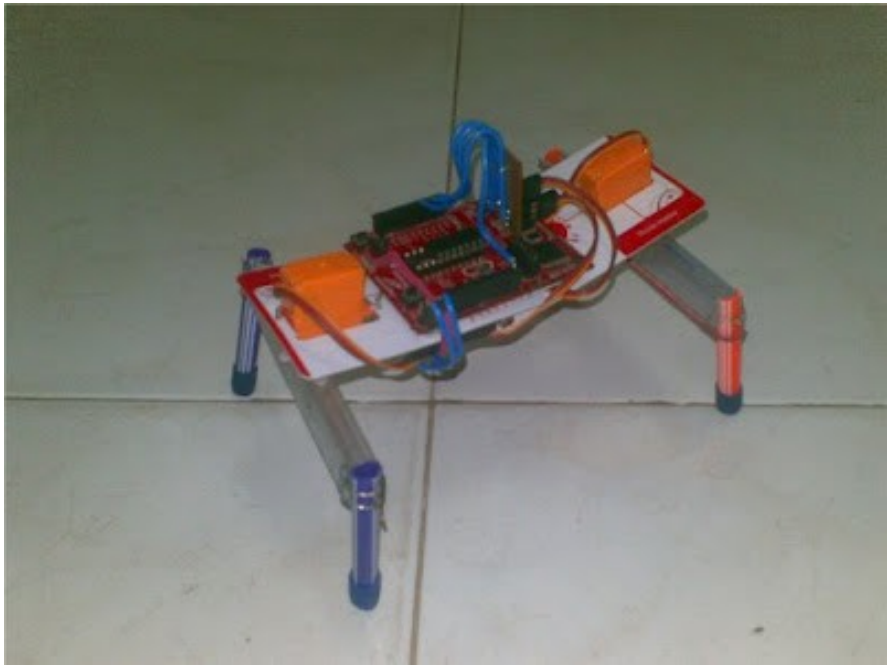
- Szervo vezérlése a soros porton keresztül. A terminálablakban megadhatjuk a kívánt beállási szöget (0 - 179)
- A kapcsolás ugyanaz, mint a **Servo_Sweep** projektnél.

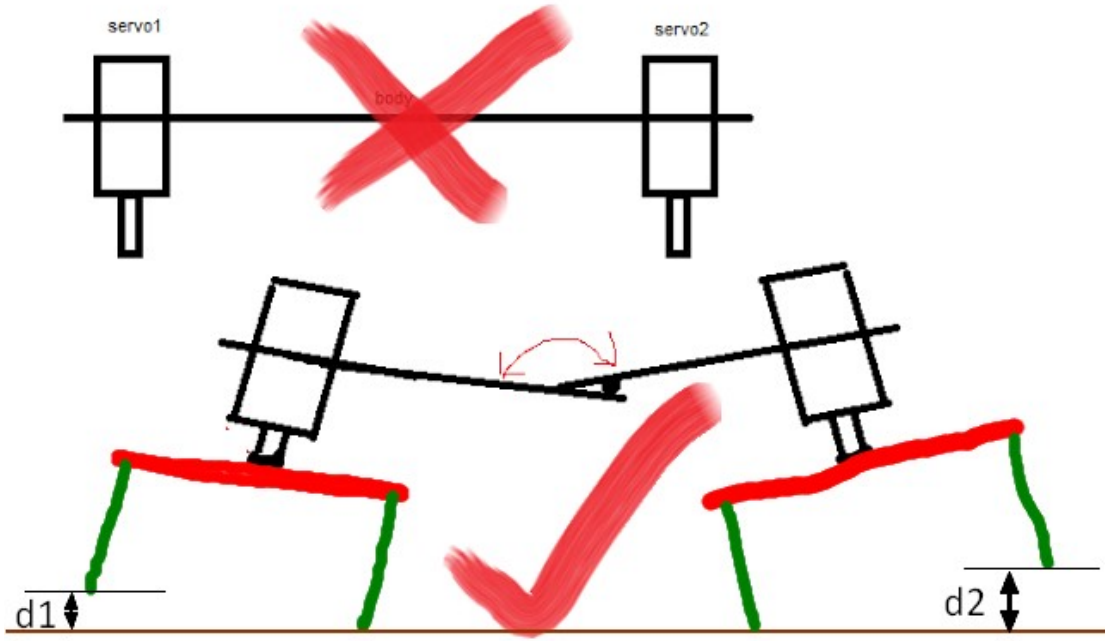
```
#include <Servo.h>
Servo servo;                          //Objektum példányosítása
#define servoPin  9                   //Szervo a D9 lábra kötve

void setup() {
  Serial.begin(9600);                 //A soros port inicializálása
  servo.attach(servoPin,640,2400);    //Inicializálás határértékek megadásával
}

void loop() {
  if(Serial.available() > 0) {
    int angle = Serial.parseInt();    //Pozíció megadása soros porton
    angle = constrain(angle,0,179); //Korlátozzuk 0-179 közé!
    servo.write(angle);               //Szervo beállítása a megadott állásba
    Serial.print("Servo position: ");
    Serial.println(angle);
    delay(100);
  }
}
```

Lépegető robot

Már két szervó is elég lehet ahhoz, hogy egy egyszerű lépegető robotot építsünk. Az ötlet és a zseniálisan egyszerű megvalósítás Vinod Stanur (Kerala, India) nevéhez fűződik.

Link: http://blog.vinu.co.in/2012/06/two-servo-walking-robot-using-ti.html





A működés feltétele, hogy a két motor tengelye ne legyen párhuzamos!

Lépegető robot: walking_robot.ino

Az eredetileg MSP430 assembly nyelven írt program Arduino átirata:

```
#include <Servo.h>
Servo servoA, servoB;          // Objektum példányosítás
void setup() {
  servoA.attach(5,640,2400); // Inicializálás határértékek megadásával
  servoB.attach(6,640,2400); // Inicializálás határértékek megadásával
}

void loop() {
  walk_forward(5);             // Öt lépés előre
  stop_idle();                 // Lábak alaphelyzetbe
  delay(2000);
  walk_reverse(5);             // Öt lépés hátra
  stop_idle();                 // Lábak alaphelyzetbe
  delay(2000);
}

void walk_forward(int n) {
  for(int i=0; i<n; i++) {
    servoA.writeMicroseconds(1000);    delay(200);
    servoB.writeMicroseconds(1000);    delay(200);
    servoA.writeMicroseconds(2000);    delay(200);
    servoB.writeMicroseconds(2000);    delay(200);
  }
}
```

Bekötés:
servoA - D5
servoB - D6

Folytatás a következő oldalon…

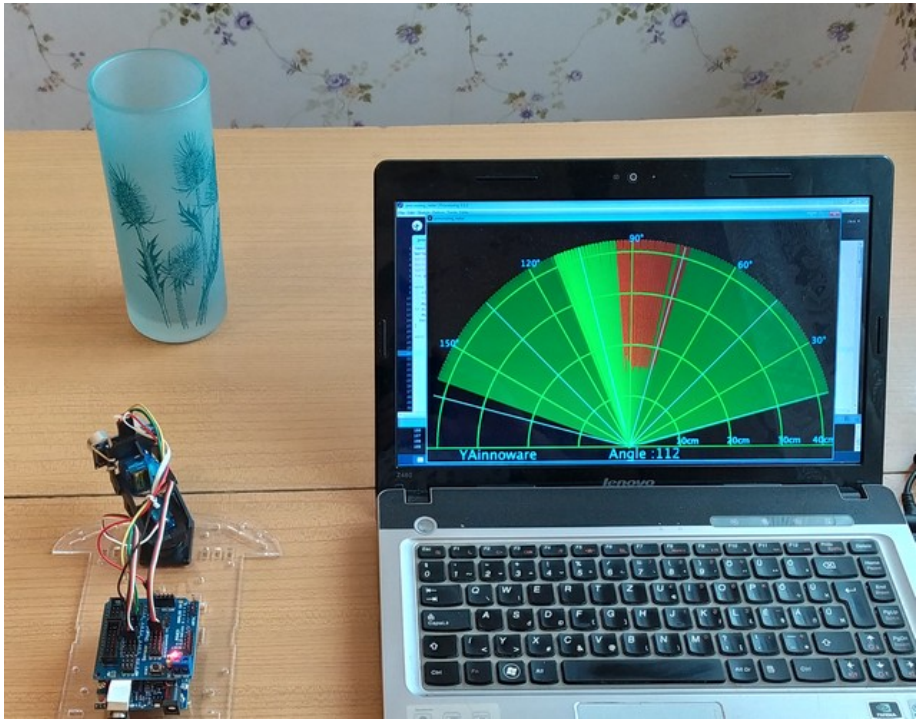
Lépegető robot: walking_robot.ino

… folytatás az előző oldalról

```
void walk_reverse(int n) {
  for(int i=0; i<n; i++) {
    servoB.writeMicroseconds(1000);    delay(200);
    servoA.writeMicroseconds(1000);    delay(200);
    servoB.writeMicroseconds(2000);    delay(200);
    servoA.writeMicroseconds(2000);    delay(200);
  }
}

void stop_idle(void) {
  servoB.writeMicroseconds(1500);    delay(200);
  servoA.writeMicroseconds(1500);    delay(200);
}
```

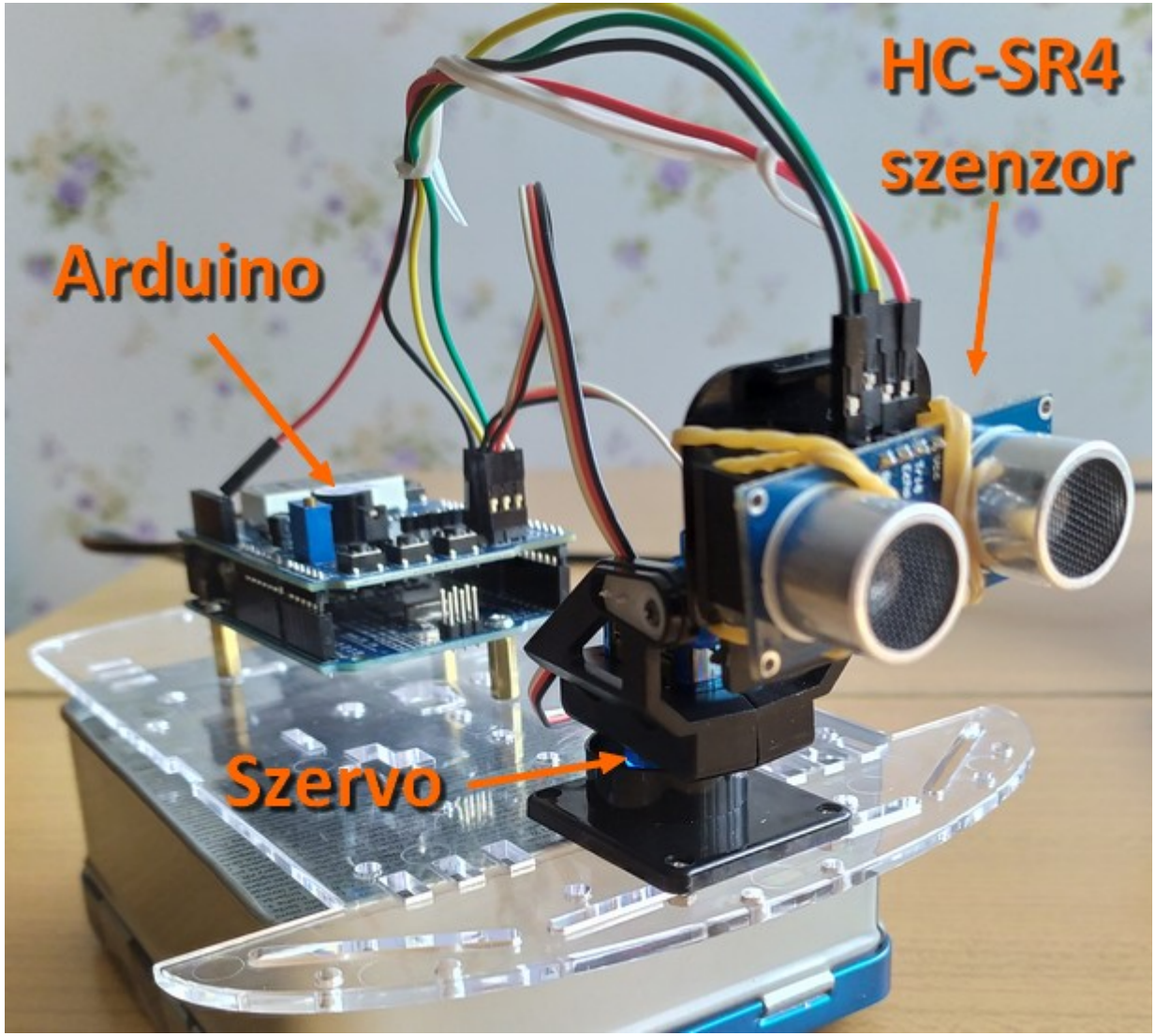
Arduino ultrahangos „radar”

- Egy érdekes Arduino/Processing projekt található az alábbi linken: create.arduino.cc/projecthub/Yug_Ajmera/radar-sonar-using-processing-3-7302c6
- A projekt Arduino-n futó része egy szervo segítségével
  - (Fél)körbeforgat egy **HC-SR04** ultrahangos távolságmérő modult
  - Minden szögnél megméri az útjába eső akadály távolságát
  - Az eredményt (szög, távolság számpár) kiküldi a soros porton
- A Processing alatt futó rész:
  - Fogadja az adatokat
  - Egy radarernyőre emlékeztető grafikonon kijelzi a mért adatokat



A hardver

- A szervo és a **HC-SR04** szenzor mechanikai összeépítése után csatlakoztassuk a szervót az Arduino D9 kivezetésére!
- Csatlakoztassuk a szenzor **Trigger** bemenetét az Arduino **D5** kimenetére!
- Csatlakoztassuk a szenzor **Echo** kimenetét az Arduino **D6** bemenetére!
- A **GND,** illetve a **VCC** kivezetések bekötéséről sem feledkezzünk meg!



arduino_radar.ino

- Hardver bekötés: **HC-SR04** Trigger – **D5**, Echo – **D6**
  Szervo vezérlő bemenet – **D9**

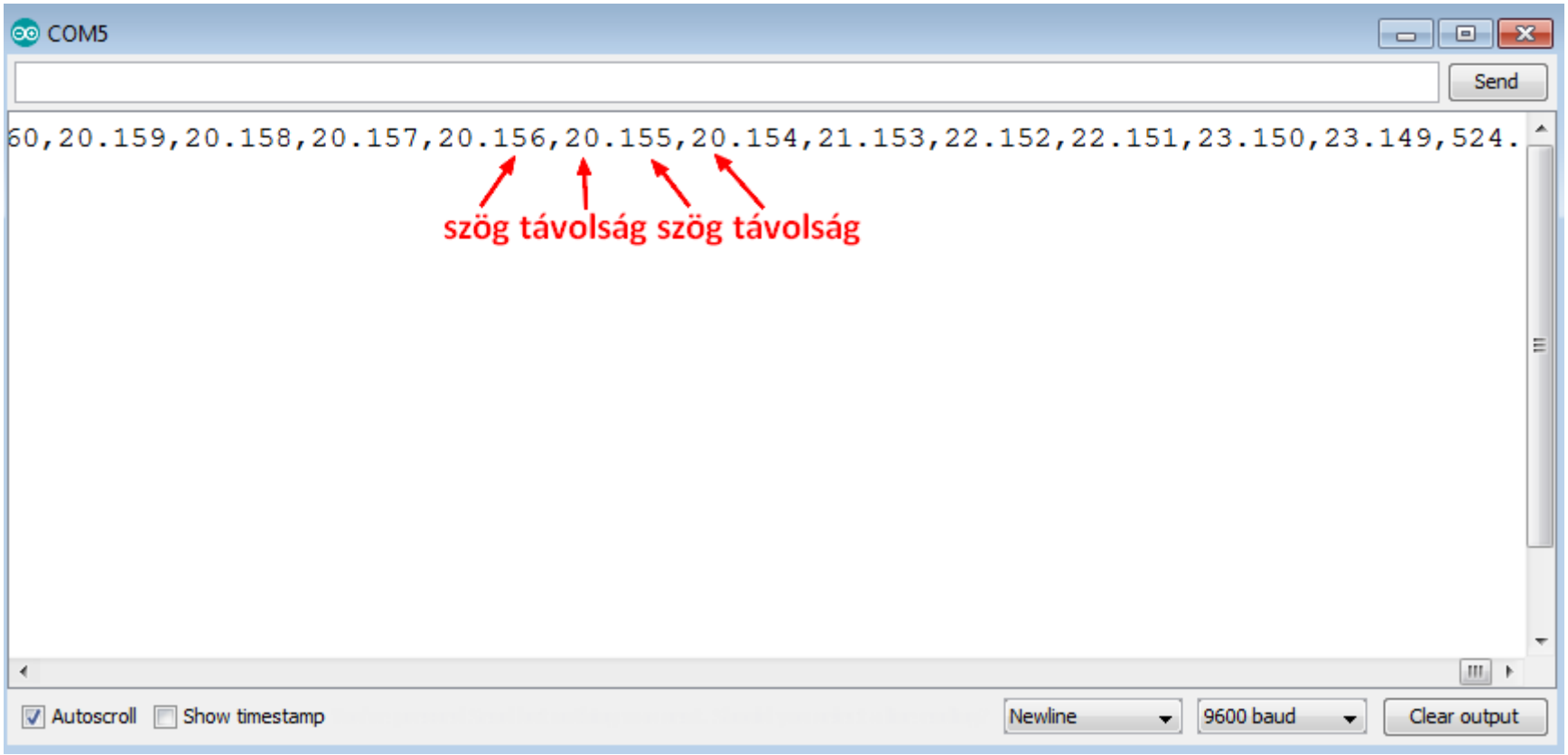
```
#include <Servo.h>
#define trigPin 5                       // Trigger a D5-re csatlakozik
#define echoPin 6                       // Echo a D6-ra csatlakozik

Servo s1;                               // Forgató motor

int angle= 90;                          // Kezdő irány (fok)
int angleStep = 1;                      // Kezdő szögléptetés (fok)
int calDist(void);                      // Függvény prototípus

void setup() {
  Serial.begin(9600);                   // Soros port inicializálás
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  s1.attach(9);                         // A D9 lábra csatlakozik a szervo
}
```

arduino_radar.ino (folytatás)

```
void loop() {
  s1.write(angle);                    // Irány beállítás
  delay(50);                          // Várunk egy picit
  int distance = calDist();           // Távolságmérés
  Serial.print(180-angle);            // Szög kiírása (fordítással)
  Serial.print(",");                  // Elválasztó karakter
  Serial.print(distance);             // Távolság cm-ben
  Serial.print(".");                  // szinkronizáló karakter
  if(angle==20 || angle==165)         // Határ elérésekor
    angleStep = - angleStep;          // megfordítjuk az irányt
  angle = angle + angleStep;          // Szög léptetése
}

int calDist() {
  int duration;
  digitalWrite(trigPin, HIGH);        // Start Trigger
  delayMicroseconds(10);              // 10 us késleltetés
  digitalWrite(trigPin, LOW);         // Stop trigger
  duration = pulseIn(echoPin,HIGH);   // Impulzus-szélesség mérés
  int distance = duration/58.82;      // Kiszámoljuk a távolságot
  return distance;                    // A cm-ben mért távolság
}
```

A program kimenete

- A programot futtatva láthatjuk, hogy a szög és a centiméterben megadott távolság vesszővel elválasztva jelenik meg
- A távolságot mindig egy pont követi, amelyet az adatok szinkronizálására használunk



A Processing fejlesztői környezet

- Az Arduino IDE elődjének (Wiring) az elődje a Casey Reas és Ben Fry által megalkotott Processing Development Envireonment (PDE)
- A Java alapú **Processing IDE** arra készült hogy megkönnyítsa a vizuális PC alkalmazások fejlesztését különös tekintettel az animációra, és hogy interakció révén azonnali visszacsatolást biztosítson
- Az ingyenes és nyílt forrású Processing IDE telepítője innen tölthető le: https://processing.org/download/
- Windows esetén egyszerűen ki kell bontani a ***ZIP*** állományt, és a használathoz el kell indítani a ***processing.exe*** futtatható állományt
- A Processing honlapján számos oktatóanyag érhető el:
  https://processing.org/tutorials/
  és számos mintapélda is található:
  https://processing.org/examples/

Processing – az első lépések

- Az alábbi programban fontot definiálunk és kiírunk vele egy szöveget, majd megrajzolunk néhány egyenest

getting_started | Processing 3.5.3

File Edit Sketch Debug Tools Help

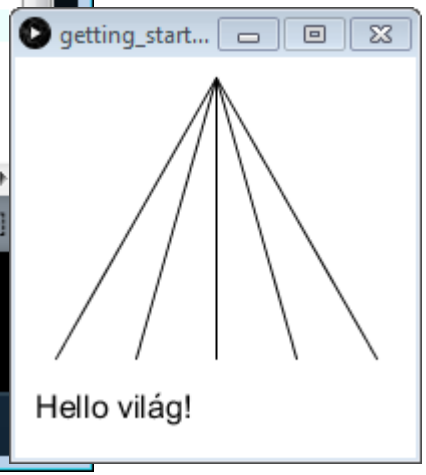
Java

getting_started

```
1  PFont f;                          // STEP 1 Declare PFont variable
2
3  void setup() {
4    size(200, 200);
5    f = createFont("Arial", 16, true); // STEP 2 Create Font
6  }
7
8  void draw() {
9    background(255);
10   textFont(f, 16);                 // STEP 3 Specify font to be used
11   fill(0);                         // STEP 4 Specify font color
12   text("Hello világ!", 10, 180);   // STEP 5 Display Text
13   for (int i=0; i<5; i++) {        // STEP 6 Draw few lines
14     line(100, 10, i*40+20, 150);
15   }
16 }
```

Console Errors

Minden programnak tartalmaznia kell egy **setup** és egy **draw** függvényt!



Processing: getting_started.pde

- Az előző oldalon bemutatott program listája, amelyben egy fontot definiálunk és kiírunk vele egy szöveget, majd megrajzolunk néhány egyenest

```
PFont f;                                  // STEP 1 Declare PFont variable

void setup() {
  size(200, 200);
  f = createFont("Arial", 16, true); // STEP 2 Create Font
}

void draw() {
  background(255);
  textFont(f, 16);                        // STEP 3 Specify font to be used
  fill(0);                                // STEP 4 Specify font color
  text("Hello világ!", 10, 180);          // STEP 5 Display Text
  for (int i=0; i<5; i++) {               // STEP 6 Draw few lines
    line(100, 10, i*40+20, 150);
  }
}
```

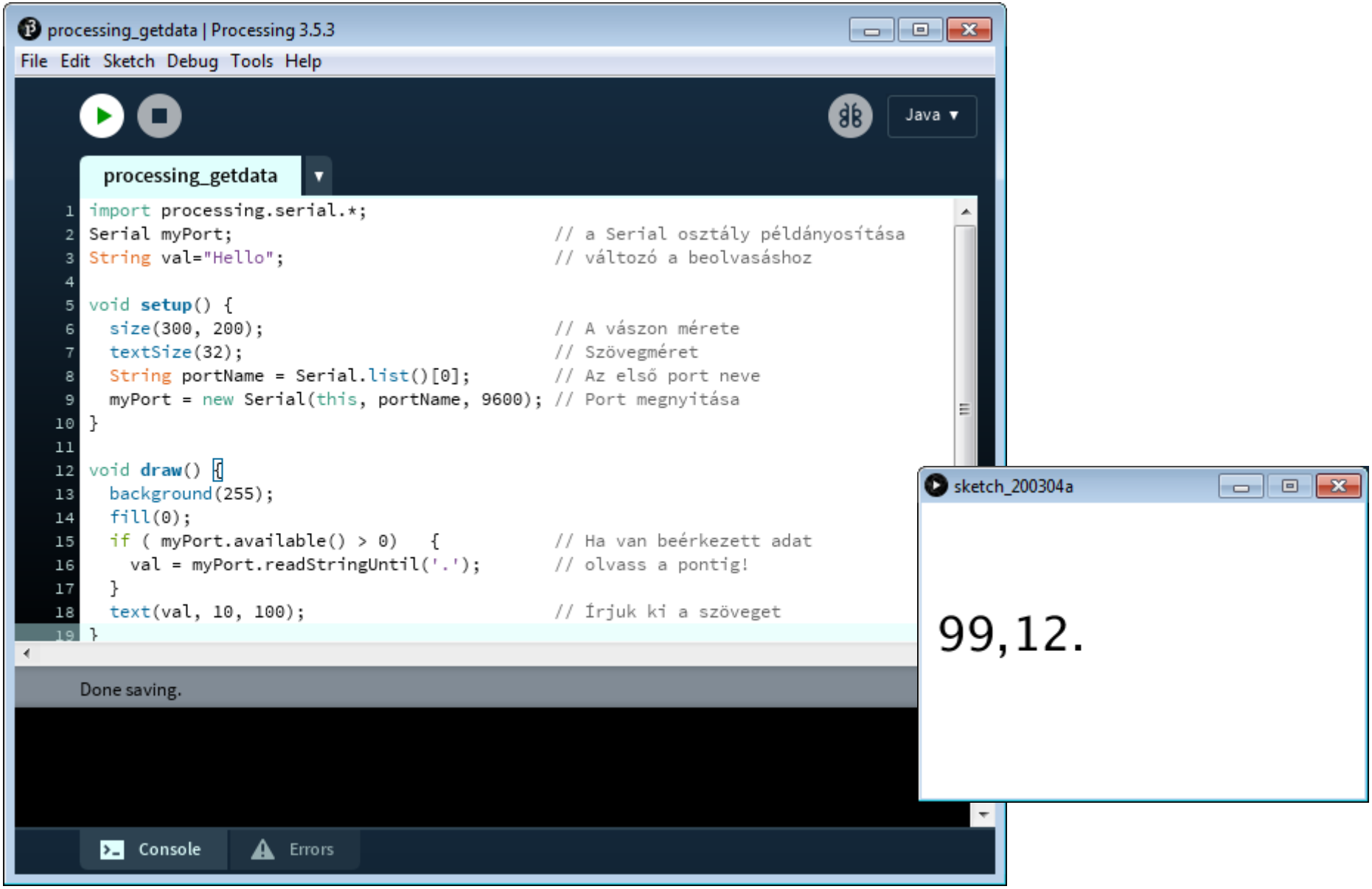
Arduino ⟶ Processing kommunikáció

- Ez a kis program megnyitja az első soros portot, a következő pontig beolvassa a beérkező karaktereket, majd kiírja az adatokat

processing_getdata.pde listája

```
import processing.serial.*;
Serial myPort;
String val="Hello";

void setup() {
  size(300, 200);                                  // A vászon mérete
  textSize(32);                                    // Szövegméret
  String portName = Serial.list()[0];              // Az első port neve
  myPort = new Serial(this, portName, 9600); // Port megnyitása
}

void draw() {
  background(255);
  fill(0);
  if ( myPort.available() > 0)   {                 // Ha van beérkezett adat
    val = myPort.readStringUntil('.');             // olvass a pontig!
  }
  text(val, 10, 100);                              // Írjuk ki a szöveget
}
```

processing_getdata.pde futási eredmény



processing_radar.pde 1/5

- Lássuk a medvét! Ez a Processing program jeleníti meg grafikusan az akusztikus radarunk adatait. Forrás:
  create.arduino.cc/projecthub/Yug_Ajmera/radar-sonar-using-processing-3-7302c6

```
import processing.serial.*;   // imports library for serial communication
Serial myPort;                // defines Object for Serial
String ang="";
String distance="";
String data="";
int angle, dist;

void setup() {
  size (1200, 700);           // Canvas size
  String portName = Serial.list()[0];
  myPort = new Serial(this, portName, 9600);  // Open first serial port
  // myPort = new Serial(this,"COM5", 9600);  // Open a given port ...
  myPort.bufferUntil('.');
  background(0);                              // Black background
}
```

Itt adhatjuk meg, hogy melyik porton várja az adatot a program, vagy vehetjük automatikusan az első soros portot

processing_radar.pde 2/5

```
void draw() {
  //for the blur effect use opacity between 1 - 5
  fill(0, 5);   //colour,opacity
  noStroke();
  rect(0, 0, width, height*0.93);
  noStroke();
  fill(0, 255);
  rect(0, height*0.93, width, height);
  drawRadar();     // Radar screen
  drawLine();      // Green line
  drawObject();    // Red line
  drawText();      // Bottom textline
}

void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(0, 255, 0);
  translate(width/2, height-height*0.06);
  line(0,0,(width/2)*cos(radians(angle)),(-width/2)*sin(radians(angle)));
  popMatrix();
}
```

Itt változtathatjuk az „utánvilágítási" időt
1 - 5 közötti érték javasolt

processing_radar.pde 3/5

```
void serialEvent (Serial myPort) {
  data = myPort.readStringUntil('.');
  data = data.substring(0, data.length()-1);
  int index1 = data.indexOf(",");
  ang= data.substring(0, index1);
  distance= data.substring(index1+1, data.length());
  angle = int(ang);
  dist = int(distance);
  System.out.println(angle);
}

void drawObject() {
  pushMatrix();
  strokeWeight(9);  stroke(255, 0, 0);
  translate(width/2, height-height*0.06);
  float pixleDist = (dist/40.0)*(width/2.0);
  float pd=(width/2)-pixleDist;
  float x=-pixleDist*cos(radians(angle));
  float y=-pixleDist*sin(radians(angle));
  if (dist<=40) {                  // limiting the range to 40 cms
    line(-x, y, -x+(pd*cos(radians(angle))), y-(pd*sin(radians(angle))));
  }
  popMatrix();
}
```

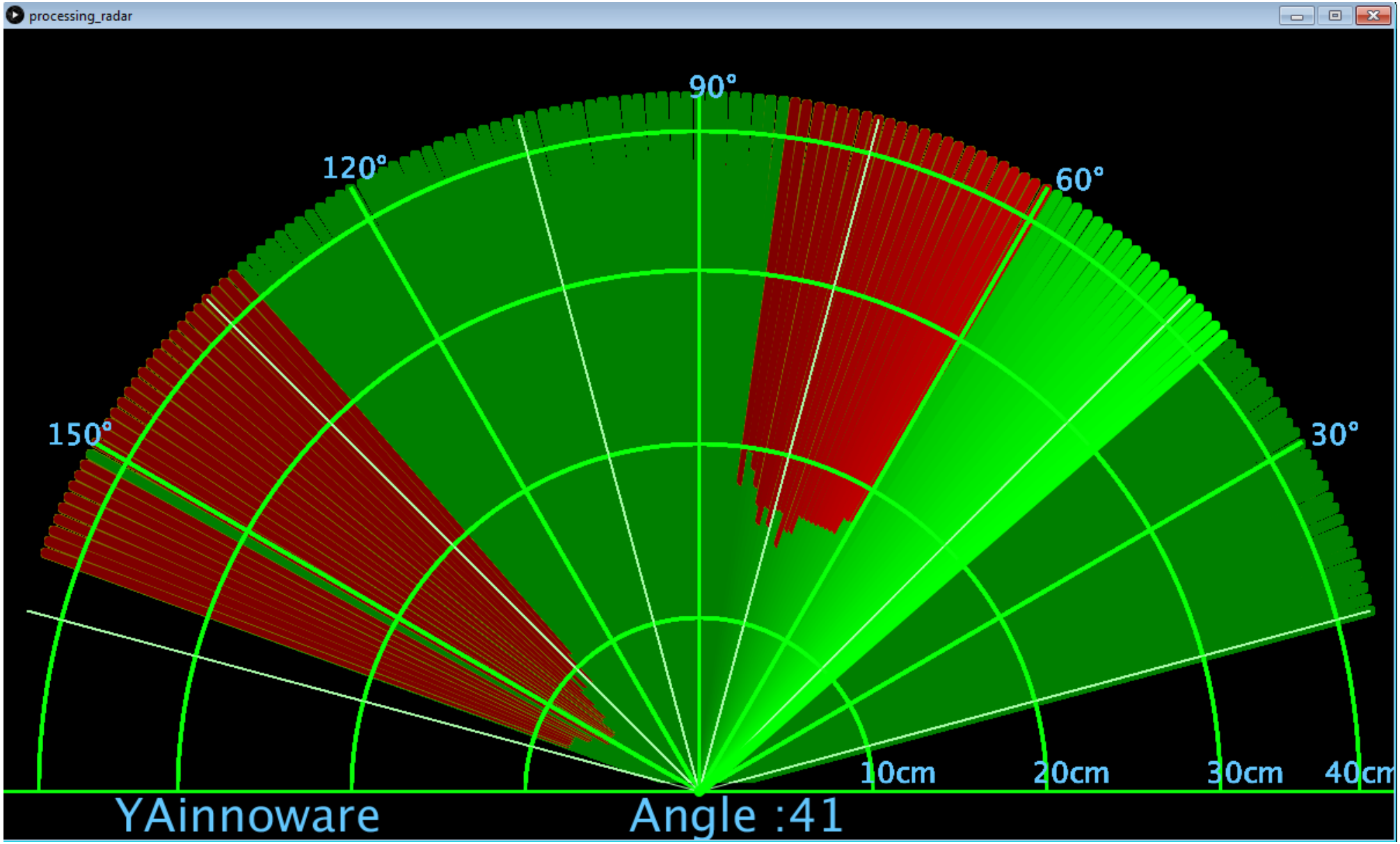
processing_radar.pde 4/5

```
void drawRadar() {
  pushMatrix();  noFill();
  strokeWeight(3);   stroke(10, 255, 10);         //green
  translate(width/2, height-height*0.06);
  line(-width/2, 0, width/2, 0);
  arc(0, 0, (width*0.5), (width*0.5), PI, TWO_PI);
  arc(0, 0, (width*0.25), (width*0.25), PI, TWO_PI);
  arc(0, 0, (width*0.75), (width*0.75), PI, TWO_PI);
  arc(0, 0, (width*0.95), (width*0.95), PI, TWO_PI);
  line(0, 0, (-width/2)*cos(radians(30)), (-width/2)*sin(radians(30)));
  line(0, 0, (-width/2)*cos(radians(60)), (-width/2)*sin(radians(60)));
  line(0, 0, (-width/2)*cos(radians(90)), (-width/2)*sin(radians(90)));
  line(0, 0, (-width/2)*cos(radians(120)), (-width/2)*sin(radians(120)));
  line(0, 0, (-width/2)*cos(radians(150)), (-width/2)*sin(radians(150)));
  stroke(175, 255, 175);
  strokeWeight(1);
  line(0, 0, (-width/2)*cos(radians(15)), (-width/2)*sin(radians(15)));
  line(0, 0, (-width/2)*cos(radians(45)), (-width/2)*sin(radians(45)));
  line(0, 0, (-width/2)*cos(radians(75)), (-width/2)*sin(radians(75)));
  line(0, 0, (-width/2)*cos(radians(105)), (-width/2)*sin(radians(105)));
  line(0, 0, (-width/2)*cos(radians(135)), (-width/2)*sin(radians(135)));
  line(0, 0, (-width/2)*cos(radians(165)), (-width/2)*sin(radians(165)));
  popMatrix();
}
```

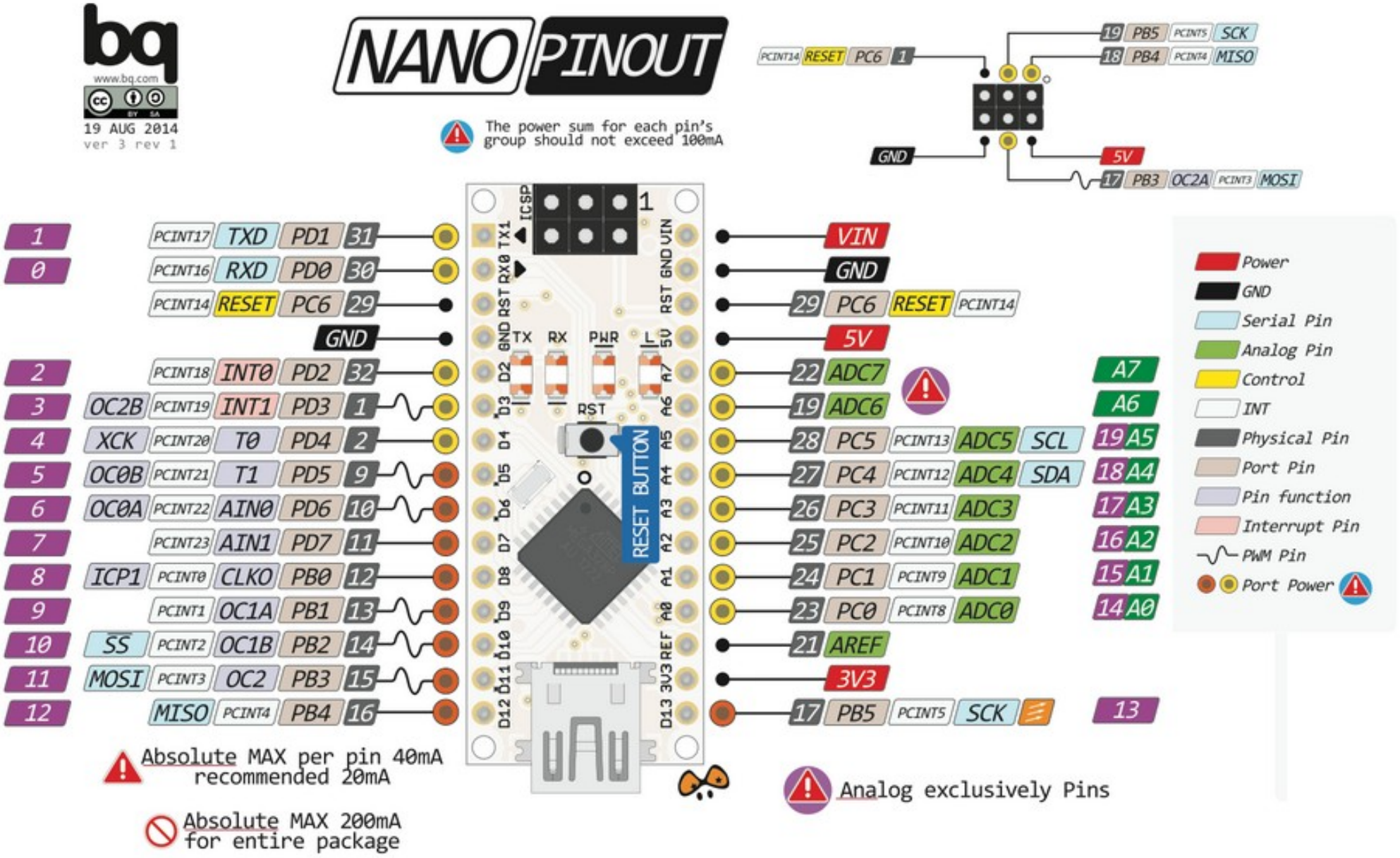
processing_radar.pde 5/5

```
void drawText() {
  pushMatrix();
  fill(100, 200, 255);
  textSize(25);
  text("10cm", (width/2)+(width*0.115), height*0.93);
  text("20cm", (width/2)+(width*0.24), height*0.93);
  text("30cm", (width/2)+(width*0.365), height*0.93);
  text("40cm", (width/2)+(width*0.45), height*0.93);
  textSize(40);
  text("YAinnoware", width*0.08, height*0.99);
  text("Angle :"+angle, width*0.45, height*0.99);
  if (dist<=40) { text("Distance :"+dist, width*0.7, height*0.99); }
    translate(width/2, height-height*0.06);
  textSize(25);
  text(" 30°", (width/2)*cos(radians(30)), (-width/2)*sin(radians(30)));
  text(" 60°", (width/2)*cos(radians(60)), (-width/2)*sin(radians(60)));
  text("90°", (width/2)*cos(radians(91)), (-width/2)*sin(radians(90)));
  text("120°", (width/2)*cos(radians(123)), (-width/2)*sin(radians(118)));
  text("150°", (width/2)*cos(radians(160)), (-width/2)*sin(radians(150)));
  popMatrix();
}
```

processing_radar futási eredmény



Az Arduino nano kártya kivezetései



Ellenállás színkódok