UNIVERSITAS BUDENSIS

Óbudai Egyetem
Neumann János Informatikai Kar

Beágyazott és érzékelőalapú rendszerek

(NIXBE1IBNE)

Lovas István
lovas.istvan@nik.uni-obuda.hu
Ládi Alexander
Ladi.alexander@nik.uni-obuda.hu

Óra anyaga:

- Ultrahangos távoltás mérő használata
- Szűrő algoritmusok: átlagoló, medián, exponenciális simítás

Ultrahangos távolságmérő

Az ultrahangos távolságmerő 3 fő egységből áll:

Adó (hangszóró)

Vevő (mikrofon)

Erősítő

A működése, egy bemeneti feszültség impulzus hatására a hangszórón egy ultra hang tartományú (30-40 kHz) hanghullámot enged ki. Ez a hanghullám vissza verődik a mérendő tárgyról és a mikrofonba visszatér, az erősítő áramkör ezt értékeli és vissza alakítja egy feszültség impulzussá amit fel lehet dolgozni egy mikrovezérlővel.

Ultrahangos távolságmérő

A vissza verődött hanghullám sokkal gyengébb mint a kibocsájtott, ezért nagy erősítésre van szükség, ami fokozza a hibalehetőséget. Valamint ha nem közel 90° áll a mérendő tárgy akkor, a visszavert hullám nem a mikrofont találja el emiatt nem érzékeli a rendszer.

A fent említett hiba lehetőségek miatt szükségesek a szűrő algoritmusok amik ezeket a hibákat kiszűrik.

Előtte viszont egy alap kapcsolást nézünk meg amin kipróbáljuk az ultrahangos távolságmérő szenzort.

Ultrahangos távolságmérő

Az álltalunk használt szenzoról a legfontosabb adatok,

- Típusa : HC-SR04
- 2cm és 400 cm között mér
- Maximális mintavételezési frekvencia: 40 Hz
- 4 kivezetése:

- VCC : 5V
- GND : földelés
- Trig : Ezen a kivezetésen lehet impulzust adni a szenzornak, hogy kiküldje a hanghullámot.
- Echo: Ezen a visszavert hullám érzékelése esetén keletkezik impulzus.

Ultrahangos távolságmérő

A kód leírása: (adatlap alapján)

Először generálni kell egy impulzust (Trig), ami 10 ms ideig van magas állapotban utána lemegy alacsonyba. A szenzor a lefutó élre aktív.

Utána meg kell mérni, hogy mennyi ideig van magas állapotban az ECHO láb. Erre egy beépített függvényt használunk. Ez a `pulseIn(<pin>, <state>, <timeout>);`

Pin: arduino port száma.

State: port állapota, amelyiket mérni szerelnénk. (HIGH, LOW)

Timeout : maximális idő, amit megengedünk a mérésre. (jelen esetben a maximális távolság idejével egyezik meg, ha nem mértünk semmi akkor ne várassuk a programunkat.)

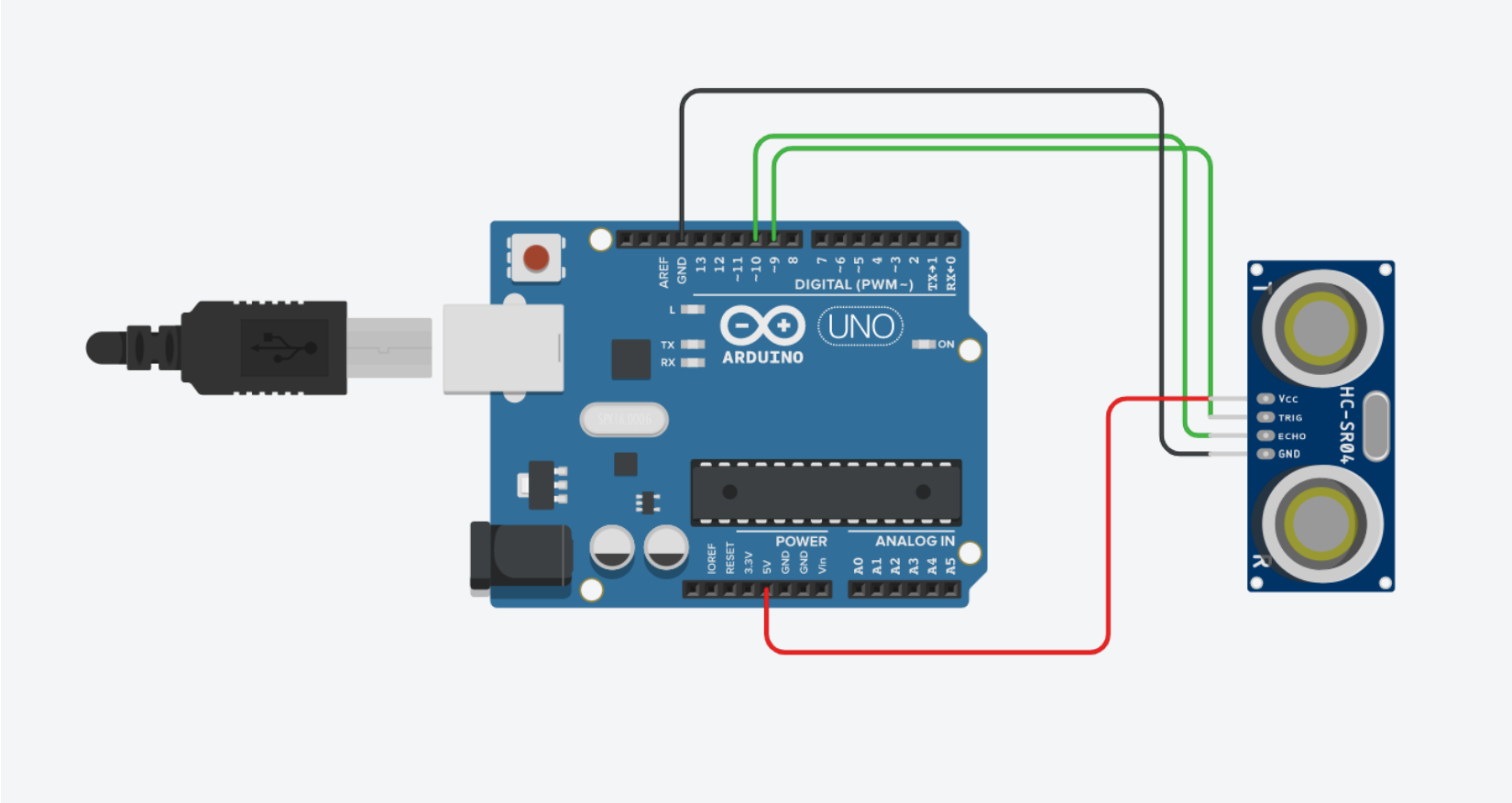
EZ A FÜGGVÉNY MEGSZAÍTÁST HASZNÁL, ÍGY MEGSZAKÍTÁSBÓL KÖZVETLENÜL NEM HASZNÁLHATÓ

Ultrahangos távolságmérő

A függvény visszatérési értéke egy us időtartam, azaz mennyi ideig volt a láb az adott állapotán. Ezt az us értéket meg kell szorozni 0,034 el ami a hangsebességből vezethető vissza, valamit el kell osztani 2 vel mivel a hanghullám ezt az utat kétszer teszi meg. (adatlap)

Ultrahangos távolságmérő

Kapcsolás:



Ultrahangos távolságmérő

Alap kód:

```
1  const int trigPin = 9;
2  const int echoPin = 10;
3
4  long duration;
5  int distance;
6
7  void setup() {
8      pinMode(trigPin, OUTPUT);
9      pinMode(echoPin, INPUT);
10     Serial.begin(9600);
11 }
12
13
14 void loop() {
15     digitalWrite(trigPin, HIGH);
16     delayMicroseconds(10);
17     digitalWrite(trigPin, LOW);
18
19
20     duration = pulseIn(echoPin, HIGH,25000);
21
22     distance= duration*0.034/2;
23
24     Serial.print("Distance: ");
25     Serial.println(distance);
26 }
```

Ultrahangos távolságmérő

Mivel, ez egy szimulált környezet így úgy tűnhet, hogy nincs hiba a mérésekben de a valóságban sajnos igen is előfordulnak a hibák és kezelni is kell őket.

Az algoritmusok:

- Átlagoló szűrő
- Medián szűrő
- Exponenciális simítás

Az algoritmusok tesztelése miatt, bele teszünk hibákat a mért értékekbe annak érdekében, hogy megnézzük az algoritmusok hatékonyságát.

Ultrahangos távolságmérő

Hibák generálása, getDistance() fv.:

```
1  int getDistance()
2  {
3    digitalWrite(trigPin, HIGH);
4    delayMicroseconds(10);
5    digitalWrite(trigPin, LOW);
6
7
8    duration = pulseIn(echoPin, HIGH);
9
10   distance= duration*0.034/2;
11
12   switch (random(0,6))
13   {
14     case 1   : distance = 400; break;
15     case 0   : distance = 0; break;
16
17     default : break;
18   }
19 return distance;
20
21 }
22
```

Átlagoló szűrő

Az átlagoló szűrő működése az, hogy néhány mérést eltárol és ezeket minden kiértékeléskor átlagolja. Előnye, hogy egyszerű. Hátránya költséges el kell végezni egy teljes összegzést és egy osztást. Kevés eltárolt minta esetén érzékeny ha egy szélső érték érkezik be akkor a kimenetet képes nagy mértékben befolyásolni, sok eltárolt minta esetén pedig lassú.

Kettő részből áll az algoritmus, egy csúszó ablakból ami a minták tárolásáért felel. Valamit magából az átlagolóból, ami kiszámolja a minták átlagát.

Átlagoló szűrő

Kód :

```
3  #define buffetSize 20
4  int buffer[buffetSize];
5
6  void addSimple(int simpe)
7  {
8    for(byte i = 0; i<buffetSize-1; i++)
9    {
10        buffer[i] = buffer[i+1];
11   }
12   buffer[buffetSize-1] = simpe;
13 }
14
15
16 int makeAvg()
17 {
18   long sum = 0;
19     for(byte i = 0; i<buffetSize; i++)
20   {
21        sum += buffer[i];
22   }
23
24    return sum / buffetSize;
25 }
```

Átlagoló szűrő

Látható, hogy a generált hibák miatt eléggé ugrál a kimenet 20 eltárolt hibánál, érdemes kísérletezni több illetve kevesebb eltárolt minta esetén mi az eredmény.

Medián szűrő

A medián szűrő ahogy a matematikai medián defenziója is megfogalmazza a közép értéket adja vissza.
Előnye szélső értékekre nem érzékeny mivel azok ki mennek a szélekre, nem annyira költséges mind az átlagoló mivel nem feltétlen tartalmaz osztást.

3 részből áll,
- Csúszó ablakból
- Rendező algoritmusból ami egy újonnan létrehozott tömbbe rendez.
- Egy olyan függvényből ami a rendezett algoritmusból kiveszi a közép értéket azaz a középen lévő tömb elem értéket páratlan esetén.

UNIVERSITAS BUDENSIS

Medián szűrő

```
1  int bufferCopy[buffetSize];
2  int tmp;
3  void rendez()
4  {
5    for(int j = 0; j<buffetSize; j++)
6    {
7        bufferCopy[j] = buffer[j];
8    }
9    for(int i = 0; i<buffetSize; i++)
10   {
11       for(int j = i; j<buffetSize; j++)
12     {
13       if(bufferCopy[i] > bufferCopy[j])
14       {
15         tmp = bufferCopy[i];
16           bufferCopy[i] = bufferCopy[j];
17           bufferCopy[j] = tmp;
18       }
19
20     }
21   }
22 }
23
24 int getMedian()
25 {
26   return bufferCopy[buffetSize/2];
27 }
```

Exponenciális simítás

Az exponenciális simitás lényege, hogy az elmúlt pár mintavétel exponenciális függvény szerint súlyozzuk meg és ennek veszül az átalágt ami sulyozott átlag. Az α paraméterrel lehet állítani a simítás pontosságát.

A képelt :

$$\hat{y}_t = \frac{y_t + (1-\alpha)y_{t-1} + (1-\alpha)^2 y_{t-2} + \ldots}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots}$$

Exponenciális simítás

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1}$$

Ahol:

- α, a „smoothing factor”, $0 < \alpha < 1$ (minél kisebb az érték annál nagyobb súllyal veszi figyelembe a $t - 1$ állapotot)
- x_t a t időpontban mért érték,
- s_t súlyozott átlaga az x_t és a $t - 1$ időpillanatban simított s_{t-1} értéknek.

UNIVERSITAS BUDENSIS

Feladatok

8.1.: Az alapkódot és a getDistance() fv.-t felhasználva alkalmazza az átlagoló és medián szűrő algoritmust. Jelenítse meg a szűrt értéket (mind 2) soros monitoron.

8.2.: Az alapkódot és a getDistance() fv.-t felhasználva, valósítsa meg az exponenciális simítást végző algoritmust. Jeleníts meg a szűrt értéket soros monitoron.

8.2.: Alkalmazza mind a három szűrőalgoritmust, majd jeleníts meg soros monitoron! A soros monitoron kapott értékeket egy excel-ben ábrázolja egy grafikonon! A grafikon tartlamzza a nyers értékeket, az avg, median és exp. Szűrt értékeket is!

8.4.: Egy egyszerű tolató radar elkészítése. Timer segítségével csinálj egy folyamatosan mintavételező távolság mérőt, és egy digitális ledszalagra "írasd" ki az eredményt színekkel. Minél közelebb annál több piros led világítson 10 cm és 100 cm közöti ételek esetén legyen működjön.

UNIVERSITAS BUDENSIS

Köszönöm a figyelmet