IOT adatgyűjtés és adatelemzés

Fordulatszám és ingadozás mérése ESP-vel

Kedves Áron Csanád – AV43UW

A feladat

A félév során a témában részvevők egy-egy IOT-s eszköz elkészítésének feladatát kapták. Az eszközökben az a közös, hogy terv szerint felhelyezésre kerülnek majd egy esztergagépre. Az eszközök ennek az esztergagépnek a működését fogják folyamatosan mérni, ezekből adatokat gyűjteni, amelyeket végül összegyűjtünk és elemzünk/megjelenítünk. A feladatban az adatok gyűjtése mögötti lényegi tartalom, hogy majd végül ezekből az adatokból esetlegesen előre tudjuk jelezni egy probléma kialakulását.

Ebben a félévben az összetett elemzésével az adatoknak nem foglalkoztunk. A feladatból az eszközök és az adatgyűjtés megalkotása volt a feladat. Minden résztvevő kapott egy vagy több szenzort és egy ESP-t. Mindenki a szenzor(ok)hoz tartozó feladatot kapott. Végül a mérés és adatgyűjtés megoldása után az adatokat MQTT-n keresztül kellett InfluxDb és Grafana felhasználásával megjeleníteni.

Az én feladatom

A félév elején azt a feladatot kaptam, hogy az esztergagép fordulatszámát mérjem. A feladat megvalósításához kaptam egy ESP-8266-ot, egy fénykaput és egy lyukakkal rendelkező tárcsát. A működés teszteléséhez ezeken kívül kaptam egy kis elektromos motort, hogy meg tudja forgatni a tárcsát. Ez a tárcsa azért állt rendelkezésemre, mert már ezzel történt kísérlet korábban, de erről nem találtam fenn maradt infót és nekem is egyszerűbb volt tiszta lappal indulnom.

A feladat során használt minden technológia ismeretlen volt számomra az elején, ezért először a megismeréssel kezdtem. Szerencsére az ESP programozáshoz nagyon sok segédanyag található, valamint a Schönherz Elektronikai Műhelyben található labor táp használatával, ami a motor szabályozott forgatásához volt szükséges, tesztelni is tudtam a megoldásaim.

A mérendő adatok:

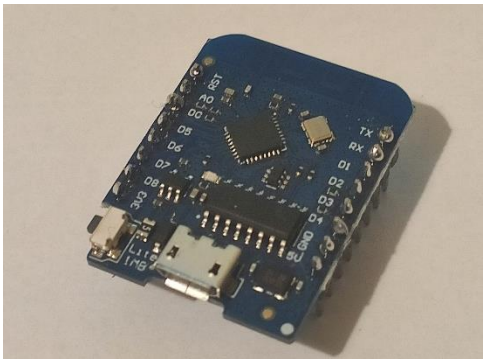
Fordulatszám. Ezt az adatot könnyebb mérni, hiszen a lyukak számolásával és az idő mérésével elég pontos értéket tudok rendelni az elmúlt másodperc átlag fordulatszámához.

Ingadozás. Ez az érték azt jelenti, hogy megfigyeljük, hogy egy-egy lyuk pár között átlagosan mennyi idő telik el. Ezeknek a mérésével és figyelésével remélhetőleg korán észre lehet venni, ha a rendszer valamelyik tengelyre merőleges irányban nem egyenletesen mozog. Ezek a kitérések jelezhetik számunkra, hogy a rendszerben akár káros rezgés van jelen.

A megvalósítás

Eszközök

ESP 8266



Egy ilyen típusú ESP mikrokontrollert kaptam a feladat elvégzéséhez. Erre az eszközre töltöttem fel a kódomat miután találtam egy használható USB kábelt. Ez az eszköz képes WiFi-n keresztül 2.4Ghz-en kommunikációra, ami fontos az MQTT szerverre való publish-oláshoz.

Maga a konkrét ESP-t egy D1 mini Lite board-on kaptam. Ennek a board-nak az adatlapja és tutorialok ezen a linken találhatóak. Ezt az adatlapot felhasználva tudtam kiolvasni a megfelelő lábakat, amelyeket az Arduino fejlesztő környezetben végül használni tudtam.

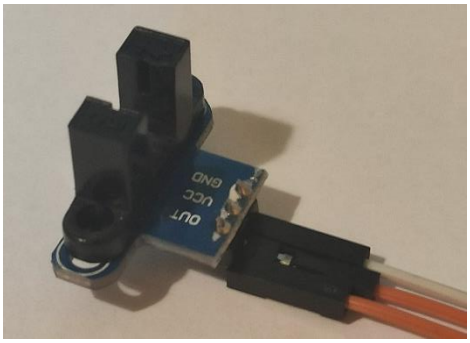
A kódom és a szenzor teszteléséhez általában soros kommunikációt használtam, amire a Serial segítségével írtam ki a számomra fontos információkat.

A felhasznált lábaim:

- GND – föld, erre kötöttem a szenzor föld lábát
- 3.3V – 3.3V-t kaptam erről a lábról, ennek a lábnak a segítségével láttam el árammal a szenzort
- D2 – erre a lábra kötöttem a szenzor kimenetét

Probléma a megoldás során:
Hosszú ideig nem tudtam miért, de ha a szenzor ágai között volt a tárcsa, akkor az ESP nem volt programozató vagy indítás után memória szeméttel öntötte el a soros kimenetet. Az adatlap alapján kiderül, hogy az ekkor használt D1 lábam más, a feltöltésé szabályozásával kapcsolatos funkciót is ellátott, ezért mivel magas volt amikor feltölteni próbáltam sikertelen volt a kommunikáció. Ezt a problémát egy másik input láb választása egyszerűen megoldotta.

Fénykapu (M0C70T3)



A tárcsa forgását a tárcsán található lyukak segítségével lehet érzékelni. Ezeken a lyukakon át tud világítani a fény és ezen villanások sűrűsége alapján tudunk következtetni a tárcsa fordulatszámára.

Maga a kapu lényegében egy dióda, amely egy apró nyomtatott áramkörre van helyezve. A dióda segítségével tudjuk megállapítani, mikor van lyuk a tárcsán.

A nyomtatott áramkörnek három lába van:

- GND – föld
- VCC – az áramellátás, itt adtam rá a 3.3V-ot
- OUT – kimenet, ha ez magasra lépett, akkor lehetett tudni, hogy lyuk került az érzékelő ágai közé

Probléma a dióda használatával:

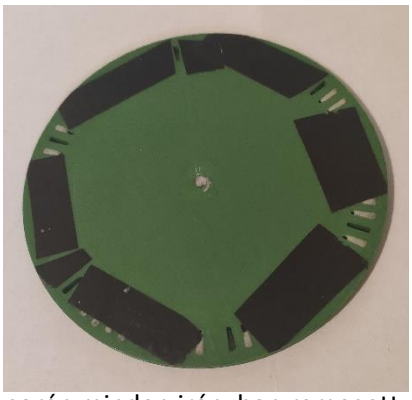
Amikor megkaptam a tárcsát, már akkor le voltak fedve lyukak. Eredetileg 72 lyuk volt a tárcsa peremén, amelyek közül csak 12 nem volt betömve. A sok lyukkal az a probléma, hogy minél több lyuk van annál kisebb fordulatszámot vagyunk képesek mérni.

Ennek az oka az, hogy ha túl gyakran kerül lyuk a diódába akkor az egyezrűen nem képes visszakerülni elég gyorsan alacsony állapotra. Tehát minél több lyuk van, annál kisebb az a fordulatszám, amelynél a dióda már folyamatosan magas értéken marad, ezzel ellehetetlenítve a fordulatszám mérését. Ezt volt szerencsém néhány segítőkész SEM-es segítségével Oszcilloszkóppal is megfigyelnem,

Néhány próbálkozás során kiderült, hogy 12 lyukkal nagyjából 2000-3000-es fordulatszámnál a rendszer mérés képtelenné vált. A végül használt 6 lyuk esetében akár 8000-es fordulatszámig is el tudtam jutni. Ez alapján a később nyomtatott új tárcsa már 6 lyukkal rendelkezik.

Ez a maximálisan mérhető fordulatszám megfelel a feladatra ugyanis végül mérendő esztergagép nagyjából 3000-es fordulatszámmal szokott forogni, amit biztosan tud mérni és kellően nagy ráhagyás van arra az esetre, ha ennél gyorsabb lenne.

Tárcsa

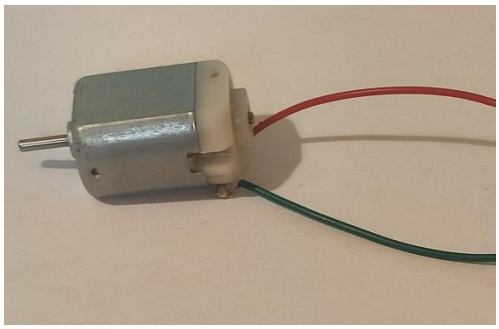


A tárcsa egy egyszerű 3D nyomtatás segítségével készült műanyag korong, aminek a peremén hosszúkás lyukak találhatóak. Eredetileg 72 lyukkal rendelkezik az, amit én használtam, de ezek közül én mindössze 6-ot hagytam meg a rendszer működése érdekében.

Az ingadozás méréséhez fontos tudni, hogy hány lyukat használunk mert minden egymást követő lyukpárhoz rendelek egy-egy átlagos értéket az alapján, hogy az elmúlt másodpercben mennyi idő volt az egyiktől a másikig eljutni.

Sajnos ennek a tesztelése nem nagyon volt lehetséges mivel a nálam lévő tárcsa súlyeloszlása nagyon egyenetlen, ezért a rendszer tesztelés során minden irányban remegett. A végső tárcsa esetében ez már nem lesz gond. A mért adatok ennek ellenére szerintem jónak tűntek.

Motor



A motor egy nagyon egyszerű elektromos motor. Azért kaptam, hogy tudjam tesztelni a rendszer működését. A tesztelés során apróbb nehézség volt, hogy nehéz a tárcsát a motor tengelyéhez rögzíteni így az a rezgés miatt néha gyorsan pörögve leesett.

Fordulat számolás

A fordulatszámolást viszonylag egyszerűen sikerült megoldanom. Tudtam, hogy a bekötött szenzor akkor kerül magas állapotba, amikor lyukhoz érek a tárcsán.

A szenzor OUT lábát fizikailag a nyomtatott lap D2 lábához kötöttem. A D2 fizikailag tartozó 4-es Pinhez pedig egy interruptot kötöttem. Ezt az interruptot a felmenő élhez párosítottam a RISING jelölés segítségével és megadtam neki egy „add” függvényt, mint végrehajtandó függvényt.

Az add függvény minden egyes interrupt esetén lefut. A fordulat számolás szempontjából csak annyit csinál, hogy minden interruptnál hozzáad egyet egy „cnt” nevű számlálóhoz. Mivel interrupt függvényről van szó, ezért fontos, hogy hosszú feladatokat például soros- és WiFi-kommunikációt nem szabad belerakni.

A loop részben 10 milliszekundumos várakozásokkal kiolvasom a számláló értéket, majd ezt egy ilyen számlálásokat tartalmazó tömbbe helyezem. Ez a tömb 100 ilyen cnt értéket képes tárolni. Ebben a tömbben egy forgó index segítségével mindig a legrégebbi értéket írom felül.

Ezt a megoldást, azért választottam az egyszerű számolás helyett, mert így ha olyan igény van nem csak másodpercenként lehet az elmúlt másodperc átlagát elküldeni, de félmásodpercenként is el lehet küldeni a teljes elmúlt másodperc átlagát.

Jelenleg miután lefut ez a rész 100szor, azaz másodpercenként, összeadom a tömbben tárolt cnt értékeket. Az összeadott értéket ezután elosztom a lyukak számával, hiszen egy fordulat során minden lyuk egyszer meg lesz figyelve. Az így kapott számot végül felszorzom 60-nal, hogy a másodpercekénti fordulatszámból a gyakran használt (RPM = round per minute) értéket kapjam.

Végül az így megkapott értéket elküldöm és kezdődik előröl a ciklus.

Ingadozás kiszámítása

Első megoldás

Amikor először elkezdtem foglalkozni az ingadozással úgy gondoltam, hogy az ESP kap egy küldső parancsot, amely hatására foglalkozik az ingadozással. Ez a megoldás lényegében úgy működött, hogy egy gomb lenyomás hatására, amit később MQTT feliratkozásban kapott parancsra cseréltem volna le, elkezdni eltárolni a timestamp értékeket egy tömbben.

Ebben a megoldásban a fordulatszám számlálást abbahagyva egy csak az ingadozással foglalkozó módban a kód megvárta amíg 600 ilyen timestamp értéket össze nem szed, majd ezeket egyszerre egy hosszú MQTT kommunikáció segítségével elküldi a z MQTT szervernek.

Ezzel a megoldással több probléma is volt:

- A működési módváltás, miatt hosszabb-rövidebb ideig nem kaptunk volt volna adatokat a fordulatszámról, ami egy értékes adat lehet.
- A 600 timestamp érték begyűjtése abban az esetben viszonylag rövid idő, ha a tárcsa nagysebességgel forog. Alacsony fordulatszám esetén ez akár több másodperc is lehet.
- Ha a mérés közben változik a rendszer fordulatszáma vagy esetleg egy pillanatra megáll a timestamp-ekben található eltérések teljesen értelmetlenek és nem hordoznak információt.
- A végül ki gyűjtött adatokat hosszú idő 600 MQTT üzenet formájában elküldeni.
- Az elküldött adatokat utána még fel kell dolgozni, mert feldolgozatlan formában a megjelenítésül lényegében értelmetlen.

Második megoldás

Második nekifutásra igyekeztem a korábbi problémáktól megszabadulni és tanulni az elkövetett hibáimból.

Ebben a megoldásban minden lyukhoz tartozik egy tömb-elem. Ezekben a tömb elemekben az eltelt idők átlagát tárolom. Ezeken kívül tárolom még a legutóbb történt interrupt időpontját is a „timestamp” változóban.

A korábban már említett add interrupt függvényben végzem ugyan úgy az ehhez a részhez tartozó műveleteket is. Minden interruptnál van egy éppen vizsgált lyukpár, ezeken interruptonként egy forgó indexszel haladok végig. Minden interrupt során megkérdezem az időt milliszekundumokban. A korábbi interrupt során elmentett idővel veszem az új időnek a különbségét, majd eltárolom az új időt, mint a legutóbbi interrupt időpontját. A különbséget és az éppen aktív lyukpár átlag időit átlagolom, majd visszaírom a tömbbe. Ezután csak tovább kell léptetnem a forgó indexet és jöhet a következő interrupt.

Végül a küldést minden fordulatszám küldésnél végrehajtom mindössze 6 MQTT üzenet segítségével.

Ennek a megoldásnak az előnyei:

- Nincs külön „mód” az ingadozás és a fordulatszám mérésére, így ezek egyszerre futnak.
- Nincs szükség 600 elem eltárolására, mindössze 6-ra. Ez hasznos lehet, ha másik feladatot ellátó kódokkal kerül egy ESP-re az enyém.
- Nincs szükség összevárni 600 interruptot, minden interruptnál frissülnek az adatok.
- A tömbben tárolt elemek összefüggésben vannak a fordulatszámmal, mivel minden lyukhoz tartozó átlag a fordulatszám alakulásával változik.
- Az így összegyűjtött adatokat nem szükséges rengeteg MQTT üzenetben elküldeni, elég csak hat üzenet.
- Az elküldött adatok sokkal kezelhetőbben és akár az egyszerű grafikus megjelenítésük is hordozhat magában információt különösebb feldolgozás nélkül.

Kapcsolat megvalósítása

Az ESP – MQTT – Telegraf – InfluxDb – Grafana kapcsolat magvalósítása egy nagy része volt a feladatnak. Ezt a feladatrészt Kurdi Barnabás hallgató társammal közösen igyekeztünk megvalósítani. Erről a közös munkánkról is írtam egy összefoglalót IOT adatgyűjtés és adatelemzés 2020 Kapcsolat létrehozás címen.

Eredmény

Összefoglalás

A sajnálatos körülmények ellenére szerintem elég jól sikerült megoldani a feladatot. A kód működését amennyire tudtam az itthoni környezetben tesztelni teszteltem.

A kódom képes fordulatszámlálásra, amely szerintem elég pontos, és képes az ingadozás értékek mérésére és átlagolására. Az ESP-ről sikeresen tudok MQTT szerverre üzeneteket küldeni. A feladattal való foglalkozás során sok érdekes és számomra ismeretlen technológiával tudtam megismerkedni. A járvány miatt elköltözésem miatt sajnos nem tudtam annyira tesztelni, amennyire szerettem volna, de szerintem jól mentek volna.

Kód

```
#include "ESP8266WiFi.h"
#include "PubSubClient.h"

const char* ssid = "i40tk"; //Enter SSID
const char* password = "PbKbTtKa5"; //Enter Password

const char *mqtt_broker = "192.168.33.211"; //MQTT
const int mqtt_port = 1883; //MQTT port
const char* user = "";
const char* pass = "";

WiFiClient espClient;
PubSubClient client(espClient);

const byte interruptPin = 4;
volatile float cnt = 0;

volatile float fluctuationAverage[6];
volatile double timestamp = 0;
volatile int numberOfHoles = 6;
volatile int holeIndex = 0;

volatile int Counts[100];
volatile int  indexOfCounts = 0;

volatile bool Writing = false;
```

```
void ICACHE_RAM_ATTR add() {
 cnt++;

 float current = millis();
 float timeSpent = current - timestamp;
 timestamp = current;

 fluctuationAverage[holeIndex] = (fluctuationAverage[holeIndex] + timeSpent) / 2;
 if(holeIndex == 5 ){
  holeIndex = 0;
 }
 else{
  holeIndex++;
 }
}

void setup() {
 Serial.begin(115200);
 // Connect to WiFi
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED)
 {
   delay(500);
   Serial.print("*");
 }

 client.setServer(mqtt_broker, mqtt_port);
 client.setCallback(callback);

 //MQTT CONNECT
 while (!client.connected()) {
  Serial.println("Connecting to public emqx mqtt broker.....");
  if (client.connect("esp8266-client",user,pass)) {
     Serial.println("Public emqx mqtt broker connected");
  } else {
     Serial.print("failed with state ");
     Serial.print(client.state());
     delay(2000);
  }
 }

 pinMode(interruptPin, INPUT);
 attachInterrupt(interruptPin, add, RISING);
}
```

```
void callback(char *topic, byte *payload, unsigned int length) {
   Serial.print("Message arrived in topic: ");
   Serial.println(topic);
   Serial.print("Message:");
   for (int i = 0; i < length; i++) {
      Serial.print((char) payload[i]);
   }
   Serial.println();
   Serial.println("-----------------------");
}

void loop() {
  client.loop();

  delay(10);
  Counts[indexOfCounts] = cnt;
  cnt = 0;

  if(indexOfCounts >= 99){
    Writing = true;
    indexOfCounts = 0;
  }
  else{
    indexOfCounts++;
  }

  Serial.print(holeIndex);
  Serial.print("   ");
  Serial.println(fluctuationAverage[holeIndex]);
```

```
 //Az RPM kiszamolasa es kiirasa
 if(Writing){
  float sumOfCounts = 0;
  for(int i = 0; i < 100 ; i++){
    sumOfCounts += Counts[i];
  }
  float numberOfRounds = sumOfCounts / numberOfHoles;
  float rpm  = numberOfRounds * 60;

  char message[100];
  sprintf(message,"eszterga, rpm=%f", rpm);
  client.publish("esp8266/rpm", message);

  sprintf(message,"eszterga, hole1=%f", fluctuationAverage[0]);
  client.publish("esp8266/hole1", message);

  sprintf(message,"eszterga, hole2=%f", fluctuationAverage[1]);
  client.publish("esp8266/hole2", message);

  sprintf(message,"eszterga, hole3=%f", fluctuationAverage[2]);
  client.publish("esp8266/hole3", message);

  sprintf(message,"eszterga, hole4=%f", fluctuationAverage[3]);
  client.publish("esp8266/hole4", message);

  sprintf(message,"eszterga, hole5=%f", fluctuationAverage[4]);
  client.publish("esp8266/hole5", message);

  sprintf(message,"eszterga, hole6=%f", fluctuationAverage[5]);
  client.publish("esp8266/hole6", message);

  Writing = false;
 }
}
```