Dokumentacija projekta
Automatizacija pomoću IOT-a i C#-a,
Sistem za praćenje sunca za solarne panele

Autor: Hasić Almedin

Gradačac, maj 2023

Sadržaj

Uvod

U ovom radu će biti predstavljeni različiti načini praćenja položaja sunca i analizirani su njihovi nedostaci i prednosti. Nakon toga, opisat ćemo naš program praćenja položaja sunca koji je razvijen pomoću Arduina i .NET programa. Pri čitanju ovog rada pretpostavlja se da čitalac ima određeni nivo poznavanja programskog jezika C++, uključujući petlje, uslove i funkcije koje su osnovne za ovaj jezik, te ih nećemo detaljno objašnjavati. Definicije, princip rada funkcija i ključne riječi iz biblioteka koje smo koristili, a koje nisu uključene po default-u, će biti objašnjene prilikom njihovog korištenja u programu.

Glavni cilj našeg programa je olakšati praćenje položaja i stanja sistema na koji je program spojen, te pružiti podršku u tranziciji mikrokontrolera i IOT uređaja u industrijsku automatizaciju. Svrha programa je da omogući korisnicima jednostavan uvid u položaj sunca, a također i da pruži korisne informacije koje se mogu koristiti za različite svrhe, poput razvoja solarne energije i slično. Program se sastoji od elektroničkog sklopa i softverskog dijela, koji su zajedno razvijeni da bi pružili što preciznije mjerenje položaja sunca

Solarni sistem za praćenje sunca

Solarni sistem za praćenje je mehanizam koji pomaže solarnim panelima da prate kretanje sunca tokom dana. To je tehnologija koja maksimizira efikasnost solarnih panela tako što ih drži u ravni sa sunčevim zracima. Solarni sistem za praćenje prati sunce dok se kreće po nebu i prilagođava položaj solarnih panela kako bi održao okomit ugao sa sunčevim zrakama.

Postoje dvije glavne vrste solarnih sistema za praćenje: jednoosni i dvoosni. Jednoosni sistem prati samo kretanje Sunca od istoka ka zapadu, dok dvoosni sistem takođe prati kretanje sunca od sjevera ka jugu. Dvoosni sistemi za praćenje su efikasniji jer hvataju više sunčeve svjetlosti tokom dana.

Prednosti solarnih sistema za praćenje su brojne. Oni povećavaju efikasnost solarnih panela do 40%, što znači da se više energije može proizvesti iz iste veličine solarnog panela. Ovo čini solarnu energiju održivijom opcijom za domaćinstva i preduzeća, jer mogu proizvesti više električne energije iz manje površine panela. Dodatno, solarni sistemi za praćenje mogu poboljšati životni vijek solarnih panela smanjujući količinu stresa na panele, jer su oni uvijek usklađeni sa sunčevim zracima.

Solarni sistemi za praćenje koriste se u raznim aplikacijama, od stambenih kuća do velikih solarnih farmi. U stambenim okruženjima, solarni sistemi za praćenje mogu se instalirati na krovovima ili u dvorištima, omogućavajući vlasnicima kuća da generiraju više električne energije iz svojih solarnih panela. U komercijalnim okruženjima, solarni sistemi za praćenje koriste se za napajanje zgrada ili industrijskih objekata, smanjujući potrebu za neobnovljivim izvorima energije.

Postoje i neki nedostaci solarnih sistema za praćenje. Skuplji su za instalaciju i održavanje od stacionarnih solarnih panela, a za rad im je potrebno više prostora. Osim toga, oni su podložniji oštećenjima od jakog vjetra ili jakog vremena, što može dovesti do većih troškova popravke.

Zaključno, solarni sistemi za praćenje su inovativna tehnologija koja povećava efikasnost solarnih panela i čini solarnu energiju dostupnijom za domaćinstva i preduzeća. Iako mogu biti skuplji za instalaciju i održavanje, prednosti solarnih sistema za praćenje nadmašuju nedostatke, posebno za velike solarne farme ili komercijalne aplikacije. Kako potražnja za obnovljivim izvorima energije i dalje raste, solarni sistemi za praćenje će igrati sve važniju ulogu u proizvodnji čiste i održive energije.

Murphyjev/Marfijev zakon

Marfijev zakon je popularna poslovica koja kaže da "sve što može poći naopako, poći će naopako". Ovaj zakon je postao vodeći princip za inženjere i naučnike, koji ga koriste kako bi se podsjetili da uvijek moraju biti spremni na neočekivano. Kada su u pitanju solarni sistemi za praćenje, Murphyjev zakon se primjenjuje na nekoliko načina.

Prvo, solarni sistemi za praćenje su složeni dijelovi opreme koji zahtijevaju precizan inženjering i održavanje. Uprkos najboljim naporima inženjera i tehničara, i dalje se mogu pojaviti nepredviđeni problemi, poput mehaničkih kvarova ili električnih kvarova. Ovdje stupa na scenu Marfijev zakon – ako nešto može poći naopako, vrlo je vjerovatno da će u nekom trenutku i biti.

Drugo, solarni sistemi za praćenje su izloženi elementima, pa su stoga podložni problemima vezanim za vremenske prilike kao što su jaki vjetrovi, grad i jak snijeg. Ovi faktori okoline mogu oštetiti solarne panele i druge komponente sistema, što dovodi do gubitka efikasnosti ili čak potpunog kvara. Još jednom, Murphyjev zakon nas podsjeća da moramo biti spremni za ove neočekivane događaje i poduzeti korake da ublažimo njihove posljedice.

Uprkos potencijalu da Marfijev zakon utiče na solarne sisteme za praćenje, postoje načini da se njegov uticaj minimizira. Na primjer, redovno održavanje i inspekcije mogu pomoći da se identificiraju potencijalni problemi prije nego što postanu ozbiljni problemi. Dodatno, uključivanje redundantnih i rezervnih sistema u dizajn solarnih sistema za praćenje može pomoći da se osigura da sistem nastavi da funkcioniše čak i ako određene komponente pokvare.

U zaključku, Murphyjev zakon služi kao podsjetnik da se neočekivani događaji mogu dogoditi, čak iu najdobro dizajniranim i dobro održavanim solarnim sistemima za praćenje. Međutim, poduzimanjem proaktivnih mjera i uključivanjem redundantnih i rezervnih sistema u dizajn, inženjeri i tehničari mogu minimizirati uticaj Marfijevog zakona na solarne sisteme za praćenje i osigurati da oni nastave da funkcionišu efikasno i pouzdano.

Da bi umanjili Murphyev zakon u idealnom slučaju potrebno je svesti elektroniku na minimum u ovom radu da bi napravili system solarnog praćenja po jednoj osi iskoristili smo sljedeće komponete:

- Arduino UNO R3 (mikrokontroler)
- DHT11 (senzor vlage I temperature)
- Jednokanalni relej
- Real Time Clock, RTC (sat)
- Ventilator
- Stepper motor
- Driver ULN 2005

Arduino

Arduino Uno je ploča mikrokontrolera otvorenog koda zasnovana na Microchip ATmega328P mikrokontroleru i koju je razvio Arduino.cc i prvobitno objavljen 2010. godine. Ploča je opremljena skupovima digitalnih i analognih ulazno/izlaznih (I/O) pinova koji se mogu povezati sa različitim pločama za proširenje (šildovima) i drugim kolima. Ploča ima 14 digitalnih I/O pinova (šest sposobnih za PWM izlaz), 6 analognih I/O pinova i programibilna je pomoću Arduino IDE (Integrirano razvojno okruženje), preko USB kabla tipa B.Može se napajati preko USB kabla ili eksterne baterije od 9 volti, iako prihvata napone između 7 i 20 volti. Sličan je Arduino Nano i Leonardu.

Slika 0.1 Arduino mikrokontroler

Specifikacije:

Mikrokontroler: Microchip ATmega328P[9]
Radni napon: 5 volti
Ulazni napon: 7 do 20 volti
Digitalni I/O pinovi: 14
PWM pinovi: 6 (pin #3, 5, 6, 9, 10 i 11)[11]
UART: 1
I2C: 1
SPI: 1
Pinovi analognog ulaza: 6
DC struja po I/O pinu: 20 mA
DC struja za 3.3V Pin: 50 mA
Flash memorija: 32 KB od čega 0,5 KB koristi bootloader
SRAM: 2 KB

Komunikacija

Arduino/Genuino Uno ima niz mogućnosti za komunikaciju sa računarom, drugom Arduino/Genuino pločom ili drugim mikrokontrolerima. ATmega328 pruža UART TTL (5V) serijsku komunikaciju, koja je dostupna na digitalnim pinovima 0 (RX) i 1 (TX). ATmega16U2 na ploči kanališe ovu serijsku komunikaciju preko USB-a i pojavljuje se kao virtuelni com port softveru na računaru. Firmver 16U2 koristi standardne USB COM drajvere i nije potreban eksterni drajver. Međutim, na Windows-u je potrebna .inf datoteka. Arduino softver (IDE) uključuje serijski monitor koji omogućava slanje jednostavnih tekstualnih podataka na i sa ploče. Unašem slučaju veoma je bitna serijska komunikacija **važno je napomenuti da kod serijske komunikacije pristup portu koji je otvoren može imati samo jedan program.**

Kada se Arduino poveže sa računarom, možete koristiti softver za serijsku komunikaciju kao što je Arduino IDE, PuTTY ili Serial Monitor za slanje i primanje podataka između Arduina i računara. Možete napisati program na Arduinu koji šalje podatke na PC preko serijskog porta, ili možete napisati program na PC-u koji šalje podatke na Arduino. Za slanje podataka sa Arduina na PC, možete koristiti funkcije Serial.print() ili Serial.write() u Arduino IDE.

Ove funkcije vam omogućavaju slanje podataka u različitim formatima kao što su cijeli brojevi, plutajući brojevi, znakovi i nizovi.

Za primanje podataka na PC, možete koristiti softver za serijsku komunikaciju kao što je PuTTY ili Serial Monitor. Ovi programi vam omogućavaju da vidite podatke koje Arduino šalje preko serijskog porta. Takođe možete napisati program na računaru koji čita podatke sa serijskog porta i obrađuje ih.

Sve u svemu, komunikacija sa Arduinom i PC-om putem serijske veze je jednostavan i efikasan način za razmjenu podataka između dva uređaja, a može se koristiti u raznim aplikacijama kao što su evidentiranje podataka, kontrolni sistemi i robotika.

Arduin IDE

Arduino IDE (Integrated Development Environment) je softverska aplikacija koja se koristi za pisanje, kompajliranje i učitavanje koda na Arduino ploču. Pruža grafičko sučelje lako za korištenje koje pojednostavljuje proces programiranja Arduino ploče, čak i za one koji nisu upoznati s programiranjem.

Arduino IDE je softver otvorenog koda koji je dostupan za besplatno preuzimanje sa Arduino web stranice. Kompatibilan je sa Windows, Mac OS X i Linux operativnim sistemima. IDE obezbeđuje uređivač teksta za pisanje koda, kompajler za pretvaranje koda u mašinski čitljive instrukcije i programator za učitavanje koda na Arduino ploču.

IDE je dizajniran da podržava različite programske jezike, uključujući C i C++, koji se obično koriste za programiranje Arduino ploča. Takođe uključuje biblioteku unapred napisanog koda, koji se naziva skice, koji se može koristiti kao predlošci ili početne tačke za nove projekte.

Slika 0.2 Arduino IDE

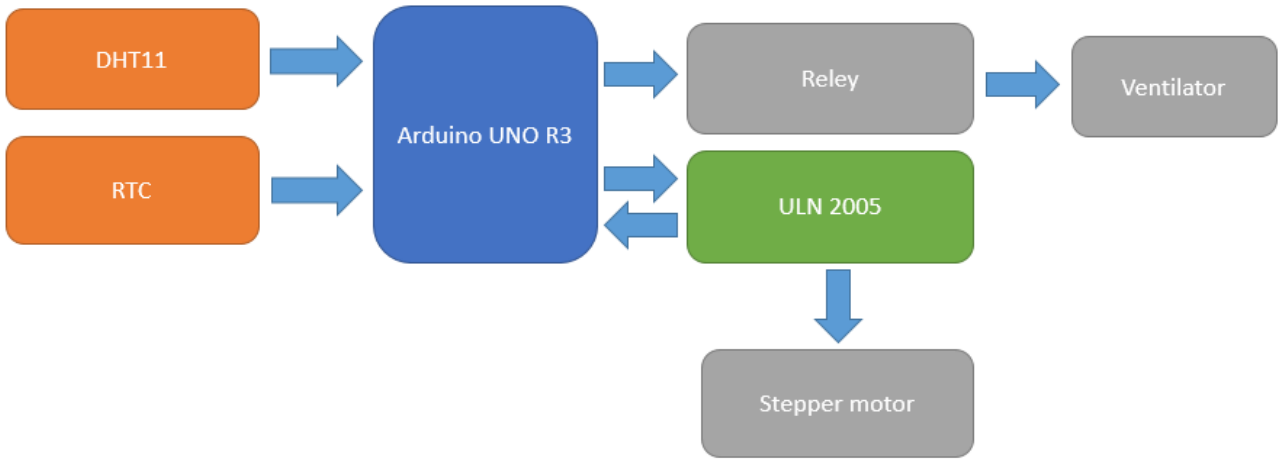
Jedna od ključnih karakteristika Arduino IDE je njegova sposobnost da pruži povratne informacije i informacije o otklanjanju grešaka. Uključuje serijski monitor koji omogućava korisnicima da vide izlaz svog programa u realnom vremenu. Ovo je posebno korisno za otklanjanje grešaka i otklanjanje grešaka u kodu.

IDE takođe uključuje niz alata za upravljanje bibliotekama i paketima podrške za ploče. Ovi alati olakšavaju dodavanje nove funkcionalnosti Arduino ploči i održavanje firmvera ploče ažurnim.

Sve u svemu, Arduino IDE pruža korisničko okruženje za programiranje i učitavanje koda na Arduino ploču. Njegova jednostavnost i lakoća korištenja čine ga popularnim izborom za hobiste, studente i profesionalce koji žele eksperimentirati s elektronikom i mikrokontrolerima.

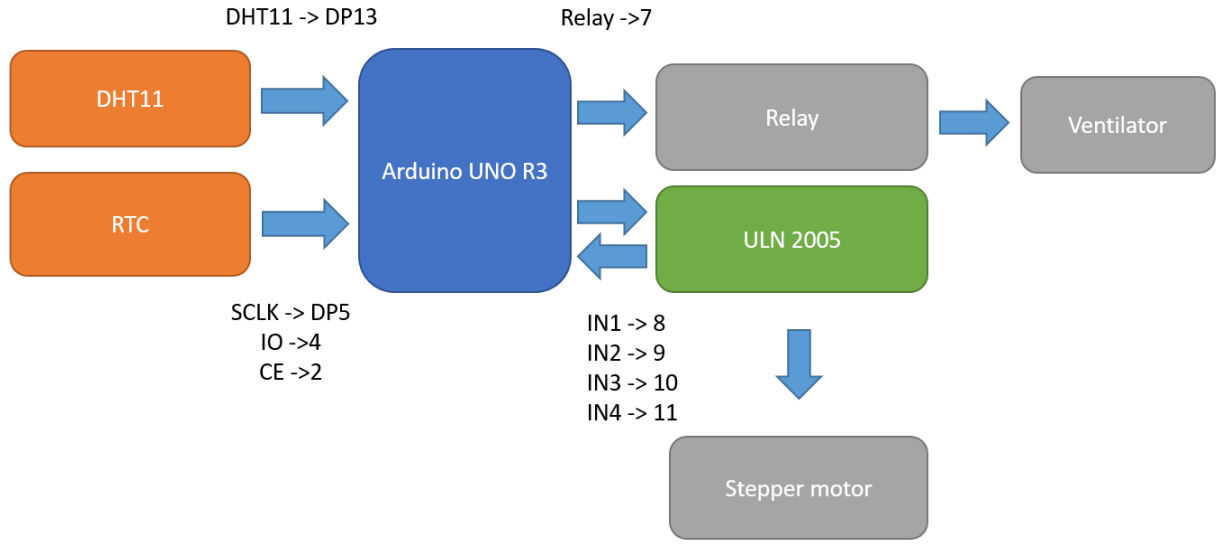
Dijagram elektronike

Prvo je bilo potrebno skicirati šemu a to je prikazano na sljedećem dijagramu



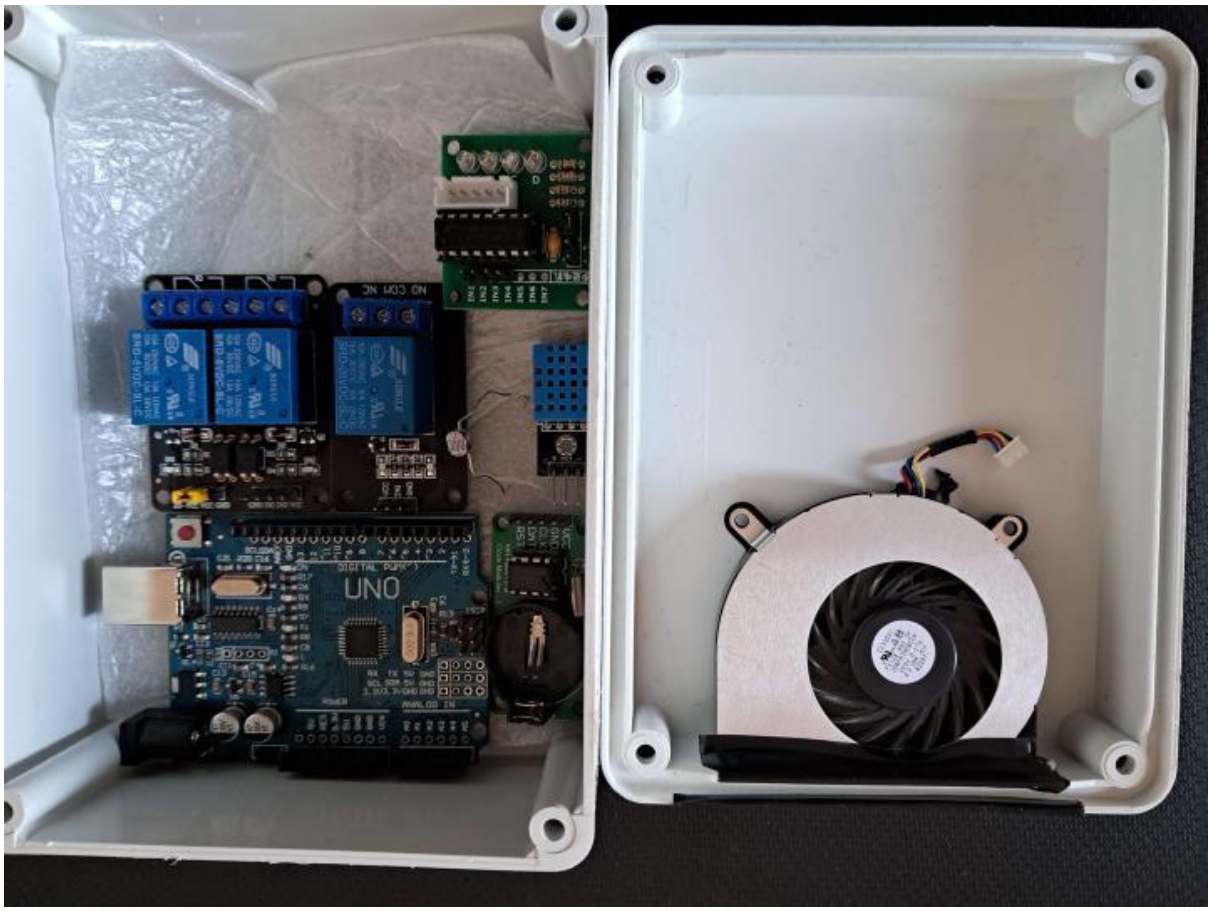
Slika 1.0 dijagram komunikacije elektronike

Sljedeće što je bilo potrebno uraditi je označiti spajanje tako da znamo koji pinovi u programu su zauzeti od koje komponente.



Slika 1.1 Dijagram sa pinovima

Nakon što smo napravili dijagram mozemo postaviti komponente. U ovom slučaju komponente su stavljene u plastičnu razvodnu kutiju na kojoj je napravljen prorez za ventilaciju i kablove.



Slika 1.3 komponente koje su korištene

Optimizacija

Da bi uporedili dobivene vrijednosti koristimo tabelu za lokalni položaj sunca u Slavonskom brodu.

Općenito, sistem solarnih panela s instaliranim solarnim tragačem s jednom osom vidi povećanje performansi između 10 i 30 posto, a u pravoj situaciji, dvoosni tracker može proizvesti do 40% više od statičkog niza.

Mjesec	Vrijeme izlaska (h)	Vrijeme zalaska (h)	Mjesečni optimalni kut
Siječanj	7:30:00	17:00:00	58°
Veljača	12:00:00	17:00:00	50°
Ožujak	6:30:00	18:00:00	39°
Travanj	5:30:00	19:00:00	22°
Svibanj	4:30:00	20:00:00	8°
Lipanj	4:00:00	21:00:00	1°
Srpanj	4:00:00	21:00:00	4°
Kolovoz	4:30:00	20:00:00	17°
Rujan	5:30:00	19:00:00	36°
Listopad	6:00:00	17:00:00	49°
Studeni	6:30:00	17:00:00	57°
Prosinac	7:00:00	16:00:00	57°

Slika 1.4 Parametni uglovi položaja za sunce (slavonski brod)

Arduino kod

Nakon što smo ostvarili sve konekcije i spojili sve komponente sa Arduinom možemo početi pisati kod. Program koji slijedi zahtjeva osnovno znanje iz programskog jezika C++ .

Biblioteke koje su potrebne su sljedeće :

```
#include <math.h> // za matematičke proračune
#include <stdio.h> // za ispis, upis i generalne funkcije
#include <ThreeWire.h> // za prosljeđivanje vrijednosti to RTC
#include <RtcDS1302.h> // bliblioteka za proracun vremena
#include<DHT.h> // biblioteka za senzor vlage i temperature
#include <AccelStepper.h> // biblioteka za stepper motor
```

Nakon što smo uključili sve biblioteke koje su nam potrebne deklarišemo varijable koje će se koristiti. U Lat i Lon postavljamo kordinate mjesta na kojem se postavlja system za praćenje ostale vrijednosti bitne za system se same popunjavaju. Također još jedna jako bitna varijabla je key koja dobija vrijednosti od C# programa.

```
ThreeWire myWire(4,5,2); // IO, SCLK, CE
RtcDS1302<ThreeWire> Rtc(myWire);

//deklaracija varijabli
float UTHours=0, UTMinutes=0, UTSeconds=30;
float Lat=44.8785, Lon=18.4276, TZ = 0;
int year=0,month=0,day=0;
int num=1;
int fan_status=0;
int received;
char key[2]={fma};
int value;
int fan;
```

Slika2.1 deklaracija varijabli

Zatim koristimo #define da definišemo pinove na kojima su komponente spojene. Postavljamo vrijednosti tacno kao što smo napisali na dijagramu.(slika 1.1)

DHT dht kreira instance koja prosljeđuje vrijednost koju dobija od senzora u biblioteku da bi dobili precizne vrijednosti

```
//#define dhtpin A0
#define motorPin1  8    // IN1 on the ULN2003 driver
#define motorPin2  9    // IN2 on the ULN2003 driver
#define motorPin3  10   // IN3 on the ULN2003 driver
#define motorPin4  11   // IN4 on the ULN2003 driver
#define STEPS_PER_REV 2048
AccelStepper stepper(AccelStepper::FULL4WIRE, motorPin1, motorPin3, motorPin2, motorPin4);
#define DHTPIN 13     // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);
#define relej 7
```

Slika2.2 definisanje pinova

U sljedećem koraku u setupu postavljamo serijski monitor, brzinu motora, stanje releja, pokrećemo senzor za temperature I RTC komponentu.

Serial.begin(9600,SERIAL_8N1)

Pokreće serijski port sa sljedećim konfiguracijama

- Baud rate: 9600
- Parity rate: NONE
- Bits:8
- Stop BIT:1

Ove konfiguracije sun am jako bitne jer pomoću njih ostvarujemo vezu sa Arduinom isto tako ćemo ovo postaviti I u C# kodu.

```
void setup ()
{
    Serial.begin(9600,SERIAL_8N1);
    stepper.setMaxSpeed(2000.0);
    stepper.setAcceleration(1000.0);
    pinMode(relej,OUTPUT);
    digitalWrite(relej,LOW);
    dht.begin();

    Rtc.Begin();
    RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
    printDateTime(compiled);
}
```

Slika 2.3 setuo

Uvrštavanje vrijednosti sa RTC-a u varijable vremena.

```
snprintf_P(datestring,
           countof(datestring),
           PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
           dt.Month(),
           dt.Day(),
           dt.Year(),
           dt.Hour(),
           dt.Minute(),
           dt.Second() );
Serial.print(datestring);
UTHours=dt.Hour();
UTMinutes=dt.Minute();
year=dt.Year();
month=dt.Month();
day=dt.Day();
```

Slika 2.4 dodjeljivanje vrijednosti

Get_sun_pos klasa vrši matematički obračun za pronalaženje azimuth-a i elavacije sunca na osnovu dobivenih vrijednosti položaja Sistema i trenutnog vremena.

```
void get_sun_pos(float latitude, float longitude, float *altitude, float *azimuth){

 float zenith;
 float pi =3.14159265358979323;
 float twopi=(2*pi);
 float rad=(pi/180);
 float EarthMeanRadius=6371.01;  // In km
 float AstronomicalUnit=149597890.;  // In km
 float DecimalHours = UTHours + (UTMinutes + UTSeconds / 60.0 ) / 60.0;
 long liAux1 =(month-14)/12;
 long liAux2=(1461*(year + 4800 + liAux1))/4 + (367*(month - 2-12*liAux1))/12- (3*(year + 4900 + liAux1)/100)/4+day-32075;
 float JulianDate=(liAux2)-0.5+ DecimalHours/24.0;
 float ElapsedJulianDays = JulianDate-2451545.0;
 float Omega=2.1429-0.0010394594*ElapsedJulianDays;
 float MeanLongitude = 4.8950630+ 0.017202791698*ElapsedJulianDays; // Radians
 float MeanAnomaly = 6.2400600+ 0.0172019699*ElapsedJulianDays;
 float EclipticLongitude = MeanLongitude + 0.03341607*sin( MeanAnomaly ) + 0.00034894*sin( 2*MeanAnomaly )-0.0001134 -0.0000203*sin(Omega);
 float EclipticObliquity = 0.4090928 - 6.2140e-9*ElapsedJulianDays +0.0000396*cos(Omega);
 float Sin_EclipticLongitude= sin( EclipticLongitude );
 float Y = cos( EclipticObliquity ) * Sin_EclipticLongitude;
 float X = cos( EclipticLongitude );
 float RightAscension = atan2( Y, X );
 if ( RightAscension < 0.0 ) RightAscension = RightAscension + twopi;
 float Declination = asin( sin( EclipticObliquity )* Sin_EclipticLongitude );
 float GreenwichMeanSiderealTime = 6.6974243242 + 0.0657098283*ElapsedJulianDays + DecimalHours;
 float LocalMeanSiderealTime = (GreenwichMeanSiderealTime*15 + longitude)*rad;
 float HourAngle = LocalMeanSiderealTime - RightAscension;
 float LatitudeInRadians = latitude*rad;
 float Cos_Latitude = cos( LatitudeInRadians );
 float Sin_Latitude = sin( LatitudeInRadians );
 float Cos_HourAngle= cos( HourAngle );
 float UTSunCoordinatesZenithAngle = (acos( Cos_Latitude*Cos_HourAngle*cos(Declination) + sin( Declination )*Sin_Latitude));
Y = -sin( HourAngle );
X = tan( Declination )*Cos_Latitude - Sin_Latitude*Cos_HourAngle;
 float UTSunCoordinatesAzimuth = atan2( Y, X );
 if ( UTSunCoordinatesAzimuth < 0.0 )
   UTSunCoordinatesAzimuth = UTSunCoordinatesAzimuth + twopi;
UTSunCoordinatesAzimuth = UTSunCoordinatesAzimuth/rad;
 float Parallax=(EarthMeanRadius/AstronomicalUnit)*sin(UTSunCoordinatesZenithAngle);
UTSunCoordinatesZenithAngle=(UTSunCoordinatesZenithAngle + Parallax)/rad;
*azimuth=UTSunCoordinatesAzimuth;
zenith=UTSunCoordinatesZenithAngle;
*altitude=90.-UTSunCoordinatesZenithAngle;
```

Slika 2.5 proracun u get_sun_pos

Zatim smo ispisali dio koda za temperature. Ovaj dio je jako bitan jer ako unutar kutije postane previse toplo i vlažno komponente bi mogle stradati zato smo napravili ventilaciju s obzirom da će biti u raznim vremenski prilikama veoma je bitno da ima dobro hlađenje.

```
// temperature
 float dummy=dht.readTemperature();
float temperature = dht.readTemperature();

// Read humidity
dummy=dht.readHumidity();
float humidity = dht.readHumidity();

if (temperature >30 || humidity >80)
{
  digitalWrite(relej,HIGH);
  delay(5000);
  fan_status=1;

}
else
{
  digitalWrite(relej,LOW);
  fan_status=0;
}
```

Slika 2.6 hlađenje

Ovaj dio koda dohvata vrijeme funkcijom Rtc.GetDateTime da bi imali tacno vrijeme kada je kod kompajliran. A dio koda //upsend prosljeđuje vrijednosti koje dobije na com odnosno serijski monitor.

```
    RtcDateTime now = Rtc.GetDateTime();
    delay(5000); // five seconds
    float alt ,az;

//upsends values to program

    get_sun_pos(Lat, Lon, &alt, &az);
    Serial.print(alt);
    Serial.print("*");
    Serial.print(az);
    Serial.print("*");
    Serial.print(stepper.currentPosition());
    Serial.print("*");
    Serial.print(temperature);
    Serial.print("*");
    Serial.print(humidity);
    Serial.print("*");
    Serial.print(fan_status);
    Serial.println(" ");
```

Slika 2.7 upsend

Ovaj dio koda se koristi za kontrolu položaja koračnog motora na osnovu željenog ugla elevacije. Prva dva reda koda izračunavaju ciljnu poziciju koračnog motora na osnovu željenog ugla elevacije.

Varijabla "stepsPerDegree" izračunava broj koraka potrebnih za pomjeranje koračnog motora za jedan stepen. To radi tako što se ukupan broj koraka potrebnih za punu revoluciju koračnog motora (STEPS_PER_REV) podijeli sa 360 stepeni. Ovo daje broj koraka potrebnih da se koračni motor pomjeri za jedan stepen.

Sljedeći red koda izračunava ciljnu poziciju koračnog motora na osnovu željenog ugla elevacije. To radi množenjem ugla elevacije (u stepenima) sa vrijednošću "stepsPerDegree".

Sljedeći red koda oduzima 30 od ciljne pozicije. Ovo vjerovatno prilagođava ciljnu poziciju da kompenzira mehaničku postavku i osigura da se koračni motor pomjeri u ispravan položaj.

Konačno, kod koristi funkciju "moveTo" za pomicanje koračnog motora na izračunatu ciljnu poziciju. Dok petlja osigurava da se koračni motor kreće sve dok ne dostigne ciljnu poziciju. Funkcije "runToPosition" i "runSpeedToPosition" koriste se za kontrolu brzine i položaja koračnog motora.

Sve u svemu, ovaj kod omogućava preciznu kontrolu položaja koračnog motora na osnovu željenog ugla elevacije.

```
//servo

  float elevation = alt;

  // Calculate target position based on elevation
  float stepsPerDegree = STEPS_PER_REV / 360.0;
  float targetPosition = elevation * stepsPerDegree;


  float targetPosition_val=targetPosition-30;
  stepper.moveTo(targetPosition_val);
  while (stepper.distanceToGo() != 0) {
    stepper.runToPosition();
    stepper.runSpeedToPosition();
  }
```

Slika 2.8 servo upravljanje

Načini slanja c#

Prije samog početka pisanja programa moramo definisati način na koji želimo da šaljemo podatke.

- Upotreba klase SerialPort: C# obezbjeđuje klasu SerialPort koja vam omogućava laku komunikaciju sa serijskim portovima. Ovu klasu možete koristiti za čitanje i pisanje podataka na serijski port, kao i za konfiguriranje postavki porta, kao što su brzina prijenosa, paritet, stop bitovi i kontrola protoka.

- Upotreba klase Stream: klasa Stream je osnovna klasa za sve tokove u C#. Ovu klasu možete koristiti za čitanje i pisanje podataka na serijski port, kao i za konfigurisanje postavki porta, baš kao i kod klase SerialPort. Međutim, korištenje klase Stream zahtijeva više koda niske razine i općenito je složenije od korištenja klase SerialPort.

- Upotreba biblioteka trećih strana: Postoji nekoliko biblioteka nezavisnih proizvođača dostupnih za serijsku komunikaciju u C#, kao što je popularna biblioteka pod nazivom "SerialPortNet". Ove biblioteke pružaju interfejs višeg nivoa od klase Stream i olakšavaju komunikaciju sa serijskim portovima.

Sve u svemu, klasa SerialPort je najčešće korištena metoda za serijsku komunikaciju u C#. Pruža jednostavan i lak za korištenje sučelje za rad sa serijskim portovima. Također to je način koji će biti korišten u programu. Ona se nalazi u biblioteci **System.Io.Ports**

Threadovi (niti)

Threading je način postizanja istovremenog izvršavanja u programu. Nit je poseban tok izvršenja unutar programa. Program može imati više niti pokrenutih u isto vrijeme, od kojih svaka izvršava različiti dio programskog koda.

Svaka nit ima svoj stog i programski brojač, ali dijeli isti memorijski prostor kao i druge niti u istom procesu. To znači da niti mogu komunicirati jedna s drugom i pristupiti istim strukturama podataka u memoriji.

Nitovima obično upravlja operativni sistem ili vrijeme izvođenja jezika, koji ih planira za izvršenje na dostupnim CPU jezgrama. Planer odlučuje koje će se niti pokrenuti, kada će ih pokrenuti i koliko dugo treba da rade, na osnovu skupa algoritama i politika za planiranje.

Niti se mogu kreirati u programu korištenjem API-ja za niti koji pruža vrijeme izvođenja jezika ili operativni sistem. Jednom kada je nit kreirana, može se pokrenuti i pokrenuti asinhrono s drugim nitima u programu.

Niti se mogu koristiti u različite svrhe, kao što je poboljšanje performansi paraleliziranjem računanja, implementacija istodobnog pristupa dijeljenim resursima ili implementacija responzivnih korisničkih interfejsa koji ne blokiraju glavnu nit. Međutim, uvođenje niti uvodi nove izazove, kao što su uslovi trke, zastoji i problemi sa sinhronizacijom, kojima se mora pažljivo upravljati kako bi se osiguralo ispravno i efikasno ponašanje programa.

This.invoke()

Konkretno, kada se koristi ova metoda, određena metoda kontrola korisničkog sučelja se stavlja na "thread-safe" pozivnu listu, što znači da će se ta metoda izvršiti u niti koja je stvorila kontrolu, a ne u niti koja je pokušala pristupiti toj kontroli. Ovo rješava problem pristupa kontrolama korisničkog sučelja iz druge niti i sprječava greške u programu.

Delegat je objekat koji predstavlja metodu sa određenim potpisom. Delegat se može koristiti za prosljeđivanje metode kao argumenta drugoj metodi ili za asinhrono pozivanje metode.

Ključna riječ delegate se koristi za definiranje anonimne metode, koja je metoda bez imena. U ovom slučaju, anonimna metoda sadrži kod koji želite da izvršite na UI niti.

```
namespace ReadSerialDataUsingThreading
{
    4 references
    public partial class Com : Form
    {
        public Thread ReadSerialDataThread;
        public string readserialvalue;
        public float[] niz = new float[4];


        1 reference
        public Com()
        {
            InitializeComponent();
            label24.Text = Convert.ToString(System.TimeZoneInfo.Local);
            label25.Update();
            sys_time_lbl.Text = Convert.ToString(System.DateTime.Now);

        }

        1 reference
        private void Mainfrm_Load(object sender, EventArgs e)
        {
            this.Text = @"Read Serial Data Using Thread";
            LoadConfigurationSetting();
            gbConnection.Enabled = false;
            Btnsave.Enabled = false;
            Btndisconnect.Enabled = false;
            Btnsend.Enabled = false;


        }
```

Slika 3.0 C# thread

varijable definirane u klasi uključuju:

- ReadSerialDataThread: objekt Thread koji se koristi za čitanje serijskih podataka na zasebnoj niti.
- readserialvalue: niz koji se koristi za pohranjivanje podataka pročitanih sa serijskog porta.
- niz: float niz sa 4 elementa.

Com klasa također definira konstruktor, koji inicijalizira obrazac pozivanjem metode InitializeComponent() i postavlja svojstvo Text forme na "Čitanje serijskih podataka pomoću niti". Također postavlja svojstvo Text dvije kontrole Label na lokalnu vremensku zonu i trenutni datum i vrijeme.

Metoda Mainfrm_Load je rukovalac događaja koji se poziva kada se obrazac učita. Postavlja svojstvo Text forme na "Čitanje serijskih podataka pomoću niti", onemogućuje nekoliko kontrola (gbConnection, Btnsave, Btndisconnect, Btnsend) i učitava postavke konfiguracije.

Ovaj dio koda postavlja osnovnu strukturu Windows forme za čitanje serijskih podataka pomoću zasebne niti i inicijalizira neka svojstva forme i varijable članova.

```
1 reference
private void LoadConfigurationSetting()
{
    txtportname.Text = ConfigurationManager.AppSettings["comname"];
    txtbaudrate.Text = ConfigurationManager.AppSettings["combaudrate"];
    txtparity.Text = ConfigurationManager.AppSettings["comparity"];
    txtdatabits.Text = ConfigurationManager.AppSettings["comdatabits"];
    txtstopbits.Text = ConfigurationManager.AppSettings["comstopbits"];
}
```

Slika 3.2 konfiguracija

Metoda LoadConfigurationSetting je privatna metoda koja učitava postavke konfiguracije za serijski port iz vanjske konfiguracijske datoteke i postavlja vrijednosti odgovarajućih kontrola na obrascu.

Metoda koristi klasu ConfigurationManager za pristup odjeljku AppSettings konfiguracijske datoteke. Odjeljak AppSettings je zbirka parova ključ/vrijednost koji se mogu koristiti za pohranjivanje postavki specifičnih za aplikaciju.

U ovoj metodi, vrijednosti sljedećih ključeva se preuzimaju iz odjeljka AppSettings pomoću svojstva ConfigurationManager.AppSettings:

- "comname": naziv serijskog porta
- "combaudrate": brzina prijenosa serijskog porta
- "comparity": postavka pariteta serijskog porta
- "comdatabits": broj bitova podataka koje koristi serijski port
- "comstopbits": broj stop bitova koje koristi serijski port

Vrijednosti ovih ključeva se zatim koriste za postavljanje svojstva Text odgovarajućih kontrola na obrascu (txtportname, txtbaudrate, txtparity, txtdatabits, txtstopbits), tako da kontrole prikazuju trenutne postavke konfiguracije.

Ovaj dio koda nam je jako bitan jer u njemu možemo postavljat vrijednosti koje sun am potrebne za ostvarenje komunikacije. Tačne vrijednosti koje smo postavili na Arduinu su (pogledaj sliku 2.3) u slučaju da ne unesemo tačne vrijednosti konekcije se neće uspostaviti i bit će bačen izuzetak (exception) da j konekcija na dati port sa datom konfiguracijom nije moguća.

Povezivanje

Koristimo SerialPortIn da otvorimo port odnosno iz biblioteke **System.IO.Ports;** koristimo funkciju **open()** unutar try bloka. Isto tako unutar try bloka postavljamo uslov ako je port otvoren program počinje da cita podatke sa serijskog porta i isključuje dugmad connect,edit,disconnect.

```
try
{
    serialPortIn.Open();
    if (serialPortIn.IsOpen)
    {
        ReadSerialData();
        Btnconnect.Enabled = false;
        Btnedit.Enabled = false;
        Btndisconnect.Enabled = true;
        Btnsend.Enabled = true;
    }
}
catch (Exception exc2)
{
    MessageBox.Show("Error" + exc2.Message);
}
```

Slika 3.3 povezivanje

Poslije try mora biti catch jer to će nam omogućiti da uklonimo mogućnosti krešovanja programa exception odnosno izuzetak koji program uzima sacuva se u varijablu exc2 koja se unutar catch bloka ispisuje.

Čitanje podataka

```
1 reference
private void ReadSerialData()
{
    try
    {
        ReadSerialDataThread = new Thread(ReadSerial);
        ReadSerialDataThread.Start();

    }
    catch (Exception e)
    {
        MessageBox.Show(@"Read Serial thread. " + e.Message);
    }
}
```

Slika 3.4 čitanje podataka

U ovom isječku koda unutar try bloka ReadSerialData() metode imamo
ReadSerialDataThread = new Thread(ReadSerial);

Ova linija nam omogućuje da preko threada kojeg smo pozvali čitamo podatke a sljedeća linija

ReadSerialDataThread.Start();

Započinje process koji smo dodijelili niti. A catch nam služi da zaustavimo kresovanje programa u ovom slučaju to je jako bitno mimo krešovanja programa radimo i sa nitima koji bi mogli krešovati i dodatne programme koji rade na njemu stoga je bitno uvijek ispisati prevenciju zbog boljeg rada programa.

```
1 reference
private void ReadSerial()
{
    try
    {
        while (serialPortIn.IsOpen)
        {
            readserialvalue = serialPortIn.ReadExisting();


            //splits data from a string by "*" character

            string[] values = readserialvalue.Split('*');
            foreach (string value in values)
            {

                if (!string.IsNullOrEmpty(value))
                {
                    ShowSerialData(value);
                    try
                    {
                        this.Invoke((MethodInvoker)delegate
                        {
                            try
                            {
                                label33.Text = values[0];
                                label34.Text = values[1];
                                label35.Text = values[2];
                                label27.Text = values[3];
                                label36.Text = values[4];
                                label39.Text = values[5];
                            }
                            catch (Exception exc)
                            {
                                MessageBox.Show(exc.Message);
                            }
                        });
                    }
```

Slika 3.5 Segmentiranje podataka

Sada kada imamo vezu sa serijskim portom I dobivamo podatke oni su svi zbijeni potrebno ih je sortirati ako se vratimo na (sliku 2.7 upsend) možemo vidjeti da nakon svake vrijednosti imamo znak `*`. Vrijednosti koje dobivamo su u stringu I treba nam način da nađemo vrijednosti do znaka `*`.

While nam služi da program čita dok god name je port otvoren i spremna vrijednosti u readserialvalue pomoću serial.PortIn.ReadExisting(); koji čita trenutne podatke na serijskom portu.

Unutar while deklarišemo string niz varijable values i koristimo Split() funkciju

u zagradi upisujemo karakter koji će da odvoji vrijednosti a to će biti `*`

String[] values = readserialvalue.split(`*`);

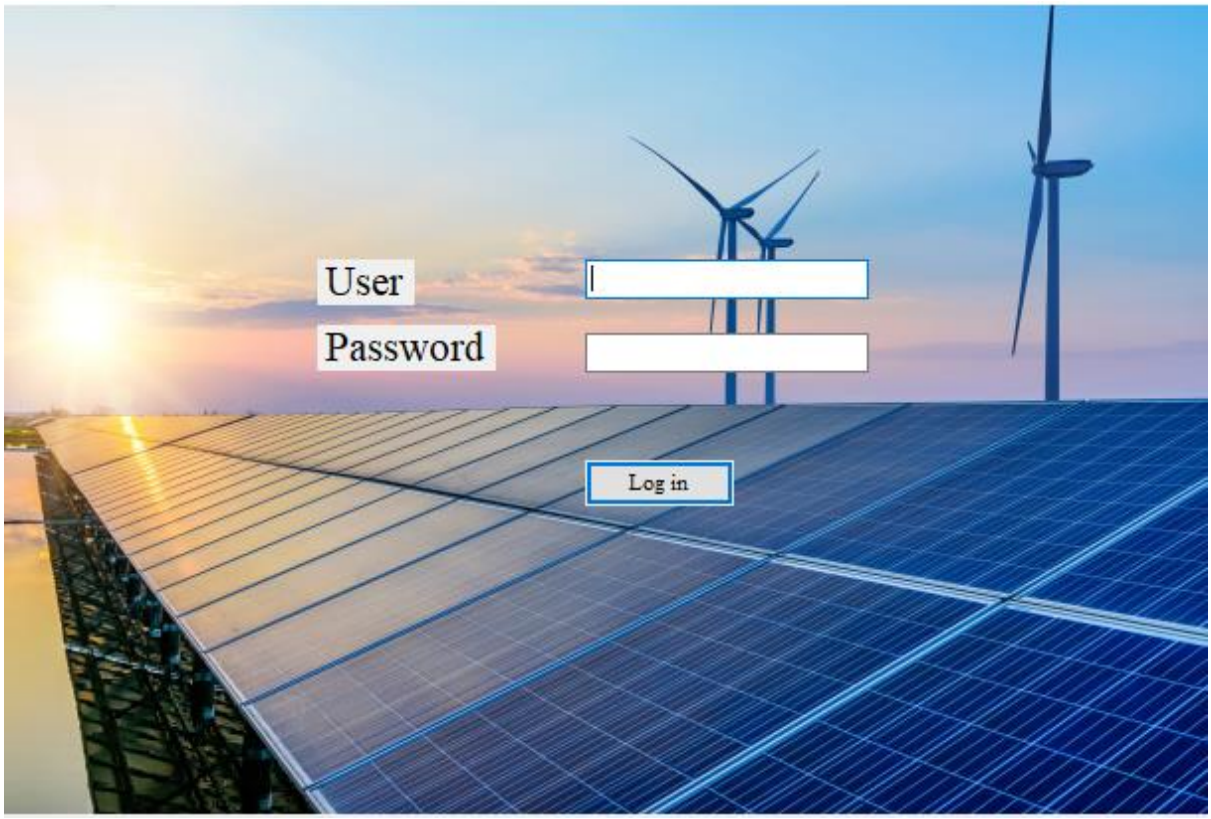
Pošto smo kreirali labele 33,34,35,27,36,39 pomoću druge niti moramo pozvati ih preko niti koja čita podatke. Linija koda koja nam to omogućava je

this.Invoke((MethodInvoker)delegate

Sada možemo dodjeljivati vrijednosti u labele.

Log in forma

Da bi zabranili pristup svima programu bilo je potrebno napraviti jednostavnu log in formu na kojoj imamo dvije labele 2 texboa I dugme.



Slika 4.1 log in forma

Kod za ovaj dio fome je sljedeći (slika 4.2) kao što možemo vidjeti postavili smo da je maksimalna dužina user imena 12 karaktera kao i šifra onda smo unutar try bloka stavili uslov koji mora imati tačnu unseen password i user da bi otvorio nofu formu inače dobit će error da je unos pogrešan I polje u kojem je unesena šifra bit će izbrisano.

Catch blok sa izuzetkom invalidExpressionException je tu da uhvati mogući unos karaktera koji nije podržan.

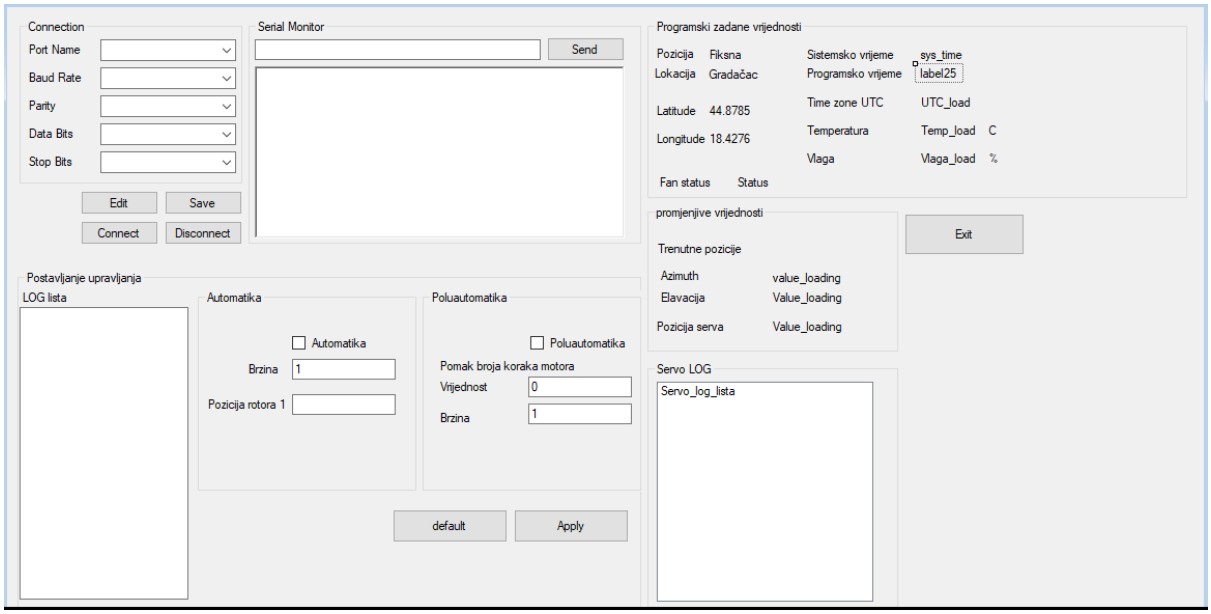
```
private void Log_in_btn_Click(object sender, EventArgs e)
{
    User_txtb.MaxLength = 12;
    password_txtb.MaxLength = 12;

    try
    {
        if (User_txtb.Text == "Admin" && password_txtb.Text == "password")
        {

            Com frm = new Com();
            frm.ShowDialog();

        }
        else
        {
            MessageBox.Show("Pogrešan password ili korisničko ime");
            password_txtb.Clear();

        }
    }catch (InvalidExpressionException ex)
    {
        MessageBox.Show("User error" + ex.Message);
        password_txtb.Clear();
    };


}
```

Slika 4.2 kod log in forme

Izgled programa

Program koji smo napravili kada se pokrene bez uređaja spojenog na njega (slika 5.1)

Da bi se veza uspješno uspostavila u paneli Connection potrebno je postaviti parameter. Većina parametara se automatski postavlja da bi korisniku olakšali korištenje ako je Arduino jedini uređaj spojen na računaru Com port će se postaviti automatski a sve ostale postavke po default su one koje mu odgovaraju. U slučaju da na računaru postoji još uređaja koji komuniciraju preko com porta potrebno je saznati na koji se port Arduino spojio i korigovati com preko edit opcije. Nakon što uđemo u edit pojavit će se drop down opcija za com port sa svim mogućim com portovima koji su trenutno otvoreni ako nema porta koji tražite uređaj koji spajate nije ispravan. Nakon izabranih postavki potrebno ih je sacuvati save opcijom I spojiti se na postavljeni port pritiskom na dugme connect.



Slika 5.1 Forma

Također u programu imamo Serial Monitor koji nam omogučuje da vidimo sve vrijednosti koje se ispisuju na jednnom mjestu I imamo mogućnost slanja posebnih naredbi na Com port. Programski zadane vrijednosti su tip pozicije, lokacija, latitude, longitude koje smo zadali u Arduinu sistemsko vrijeme će nam pokazivati kada je program pokrenut a programsko vrijeme će nam davati datum sa nuliranim vremenom, UTC nam pokazuje kao i u Arduinu vremensku zonu ako je vremenska zona na softveru I uređaju različita program u uređaju se mora korigovati da odgovara.

Vrijednosti koje nisu definisane su

1. Temperatura
2. Vlaga
3. Fan status

Promjenjive vrijednosti su Azimuth, Elavacija i pozicija serva (steppera)

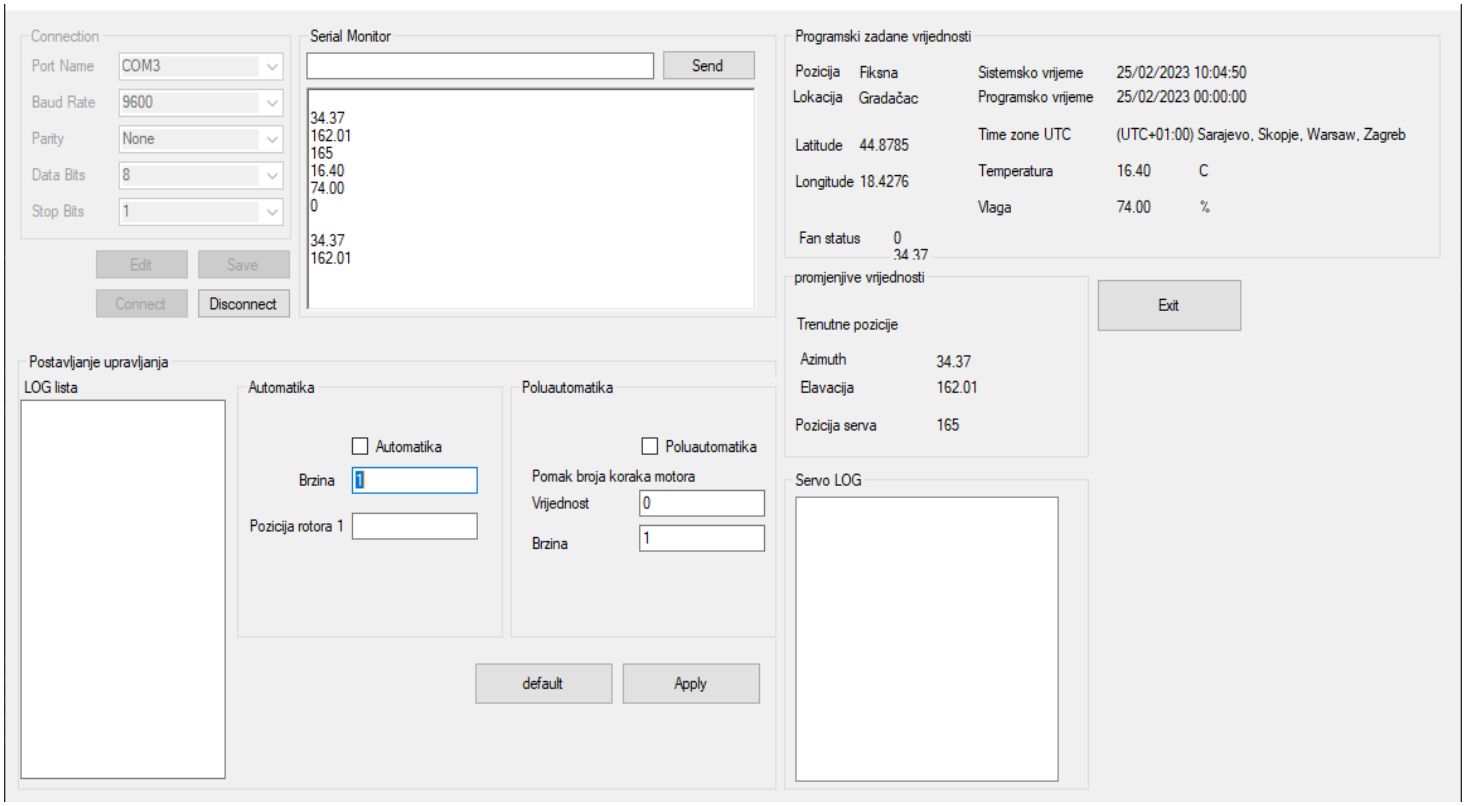
Elavacija nam govori koliko koraka servo treba napraviti i u ovisnosti je li ta vrijednost pozitivna ili negativna napravi korak.

Autoamtika po default u Arduinu je postavljeno da je automatika ali možemo postaviti i poluautomatiku (nije preporučeno) jer dolzi do velikih odstupanja nakon ponovnog vraćanja na automatiku ako nije rekalibriran program.

Servo LOG upisuje datum i vrijeme svake izmjene u programu tako da imamo svaku izmjenu na uviđaju pogotovo oko postavi automatike i poluautomatike.

Kada je uređaj uspješno spojen forma će izgledati kao na sljedećoj slici (slika 5.2).

Sve vrijednosti su popunjene imamo trenutnu temperaturu i vlagu (temperatura i vlaga se odnose na elektroniku u zatvorenom kućištu)



Slika 5.2 Sa spojenim uređajem

Zaključak

U ovom radu smo istražili različite načine praćenja položaja sunca i definirali različite izazove s kojima se susreću postojeći sustavi. Predložili smo bolji i efikasniji program za praćenje položaja sunca, koji se temelji na korištenju Arduina i .NET programa. Pokazali smo kako se ta dva programa mogu integrirati i kako se može stvoriti jednostavno sučelje koje olakšava praćenje položaja sunca i stanja povezanog sustava.

Korištenjem ove tehnologije, sustavi za praćenje sunca mogu se učinkovitije i preciznije pratiti, što može biti korisno u različitim aplikacijama, uključujući solarne elektrane, solarne sustave za grijanje vode, kao i u znanstvenim istraživanjima. Ova metoda integracije Arduina i .NET programa može se primijeniti i na druge slične sustave koji zahtijevaju brzo i pouzdano praćenje podataka.

Literatura