Grafikus folyamatmonitorizálás

1. A gyakorlat célja

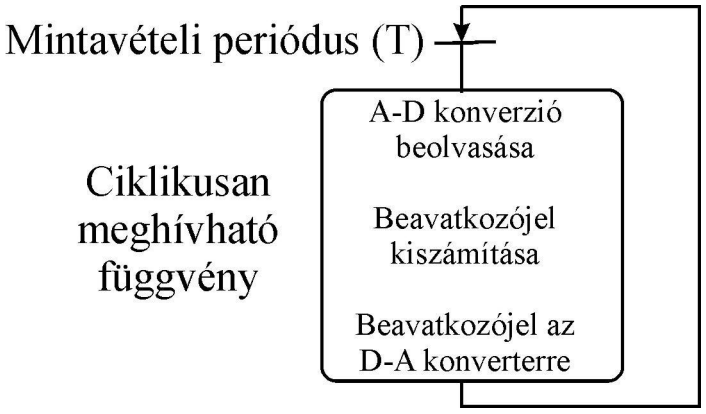
Ipari folyamatok irányítását megvalósító program alapjának megismerése, fejlesztése, lassú folyamatok grafikus monitorizálásának megvalósítása.

2. Elméleti bevezető

Az irányítási algoritmusok megvalósításának legelterjedtebb módja a szoftveres implementálás. Az irányítási algoritmust megvalósító függvények, eljárások általában egy nagyobb, ipari folyamatok felügyeletére, irányítására, monitorizálására kialakított szoftver része. Az irányítási algoritmus a folyamat mért kimenete és az előírt kimenet (alapjel) alapján számítja ki a beavatkozó jelet. A folyamat kimenetét analóg-digitális átalakítón keresztül olvassuk be. Az alapjelet ugyancsak analóg-digitális átalakítón keresztül olvashatjuk be vagy megadhatjuk számítógépes periférián keresztül is (például billentyűzet segítségével). A második esetben nincs szükség analog-digitális átalakításra az alapjel esetében. A kiszámolt beavatkozó jelet digitális-analóg átalakítón keresztül küldjük a beavatkozó bemenetére.

2.1 A mintavételi periódus

A mintavételes megvalósításnál szükségünk van egy ciklikusan (konstans periódusonként) meghívható függvényre. A konstans periódust *mintavételi periódus*nak nevezzük. Az ipari szoftver főszála ciklikusan megszakítódik és végrehajtódik az irányítási algoritmust megvalósító eljárás. Az eljárás főbb részei a következők:
- a folyamat bemenetének beolvasása analóg-digitális átalakítóról.
- a beavatkozó jel kiszámítása az alapjel és a folyamat bemenetének függvényében.
- a beavatkozó jel kiküldése a digitális-analóg átalakítón keresztül.



1 Ábra: Mintavételes szabályozás megvalósítása

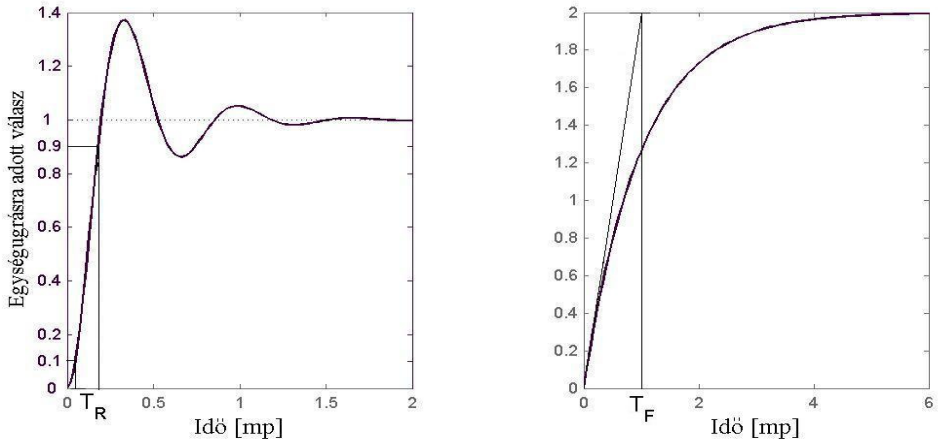
A mintavételi periódus megválasztása kritikus a szabályozás helyes működésének biztosításához. Az irányítási feladatokban *fix, konstans mintavételi periódust* alkalmazunk. Mivel a mintavételi periódus gyakoriságával olvassuk be a folyamat kimenetét, ami alapján az irányítási algoritmus kiszámítja a vezérlőjelet, a periódust úgy kell megválasztani, hogy a beolvasott mintavételezett jel tartalmazza a folytonos jel jellegzetességeit. Így a periódus megválasztása minden esetben alkalmazásfüggő. Gyors válaszú rendszerek esetében kis mintavételi periódust választunk, lassú válaszú rendszerek esetében nagyobb mintavételi periódus alkalmazható.
A mintavételi periódus megválasztásánál empirikus, tapasztalati szabályok alkalmazhatóak: amennyiben az irányított rendszer elsőfokú, vagy jól approximálható elsőfokú modellel a periódus megválasztása a rendszer időállandója függvényében történhet:

$$T = (4 \div 10)T_F$$

Ha az irányított rendszer viselkedése másodfokú lengőrendszer modelljével közelíthető, akkor a mintavételi periódus az alábbi tartományban történhet:

$$T = (4 \div 10)T_R$$

T_R (*Rise Time*) azt az időintervallumot jelöli, amely alatt a rendszer egységugrásra adott válasza 10% -ról 90% -ra emelkedik (100% a kimenet az állandósult állapotban).



2 Ábra: A mintavétel megválasztása különböző rendszerek időállandóinak függvényében

2.2. Az irányítás mellett fellépő feladatok

Az irányítást megvalósító számítógépes programnak a beavatkozó jel kiszámítása mellett több feladatot is el kell végeznie. A feladatokat két csoportba sorolhatjuk:

1. Nagy prioritású feladatok: Tipikusan biztonsági problémák, amelyek prioritása nagyobb a beavatkozó jel számításánál, megszakíthatják a vezérlőjel számolását. Például fordulatszám szabályozás esetén, ha a beavatkozó szervként használt motor károsodik, a motor védelme ezt egy digitális bemeneten keresztül jelzi az irányítást megvalósító számítógépes programnak. Ebben az esetben az irányítási algoritmus a beavatkozó jel számítását nem kell elvégezze, a futása felfüggesztődik.
2. Kis prioritású feladatok: Ezek a feladatok ugyancsak párhuzamosan futnak az irányítást megvalósító ciklikusan meghívható függvénnyel, de nem szakíthatják meg az a beavatkozó jel számítását. Ilyen feladatok:
- Megjelenítés (monitorizálás) - a beolvasott, kiküldött, számított jelek numerikus, grafikus kijelzése.
- Naplózás – a rendszer viselkedésének (beolvasott, kiküldött, számított jelek) archiválása adattárolókra az utólagos feldolgozás érdekében (irányítás kiértékelése, rendszerhibák megtalálása). A hasznos adatok elmentése mellett általában tároljuk a mentés idejét is (óra, dátum) ezért az eljárást naplózásnak is nevezzük.

3. A *mérés menete*

3.1. A ciklikusan meghívható függvény megvalósítása Windows operációs rendszerben.

A ciklikus meghívható függvények implementálásához a Windows szoftverek fejlesztésére alkalmas programozási környezetek egy külön eseményt biztosítanak. Ez az úgynevezett *Timer* esemény. A *Timer* eseményt a program futása során engedélyezni vagy letiltani lehet, illetve módosítható a periódusa. MFC osztályokat alkalmazva, amikor az eseményt (*WM_TIMER*) hozzárendeljük a programunkhoz, akkor létrejön az *OnTimer* függvény, de ez még nem aktív.
Ahhoz, hogy aktív legyen, a programban meg kell hívni a *SetTimer* függvényt:

UINT SetTimer(UINT *nIDEvent*, UINT *nElapse*, void (CALLBACK EXPORT* *lpfnTimer*)(HWND, UINT, UINT, DWORD));

nIDEvent – egy (nullánál szigoruan nagyobb) azonosító, amellyet hozzárendelünk az eseményünkhöz
nElapse – a periódus, ezredmásodpercben megadva
lpfnTimer – a timer függvény

A Timer megállításánál *KillTimer* függvényt kell alkalmazni.

BOOL KillTimer(int nIDEvent);

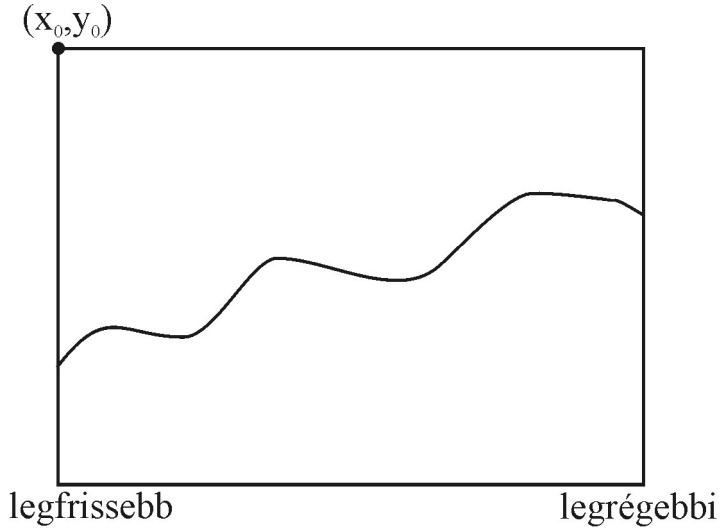
A függvény paramétere (*nIDEvent*) a Timerhez a SetTimerben hozzárendelt azonosító.
Ha felhasználjuk a programozási környezet programvarázslóját, akkor a Timer - függvényt a programozási környezet automatikusan hozzárendeli az *OnTimer* függvényt.

Feladat: Készítsünk egy MFC segítségével fejlesztett programot, és adjunk egy Timer eseményt a programunkhoz.

- Egy gomb lenyomásával indítsunk el egy Timert (1 azonosítóval, 1 másodperc periódussal)
- A gomb lenyomásával a Timer eseményt tudjuk megállítani.
- Az *OnTimer* függvényben tegyünk ki egy üzenetet a képernyőre az *AfxMessageBox* függvény felhasználásával.

3.2. Folyamatok grafikus monitorizálása

Ahhoz, hogy a folyamat aktuális és múltbeli állapotáról információnk legyen, a folyamat mért jeleinek grafikus megjelenítést alkalmazhatjuk. Ez egy időfüggvény grafikus megjelenítése, amelyben az utolsó N mintavételi periódusban begyűjtött mintákat ábrázoljuk grafikusan. Ahhoz, hogy mindig a legutolsó mérések eredményét lássuk a képernyőn, a grafikont folyamatosan el kell tolni.



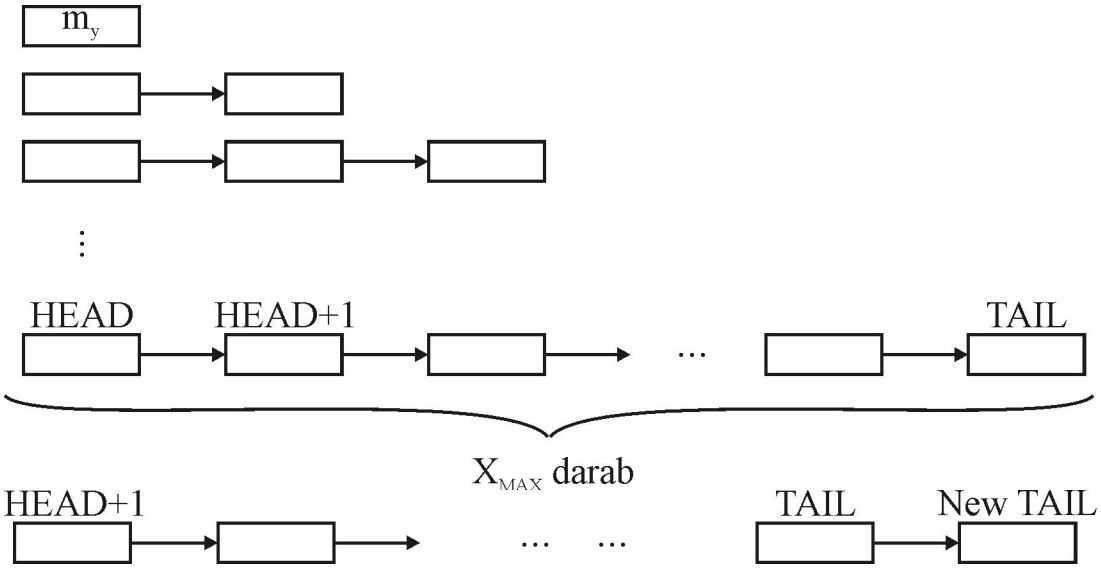
3. Ábra. Mérési adatok grafikus kijelzése

A kijelzéshez feltételezzük, hogy a képernyőn X_{MAX}, Y_{MAX} pixelméretű terület (grafikus ablak) áll rendelkezésünkre a grafikus kijelzéshez. Az egyszerűség kedvéért a grafikus ablak bal felső sarkának a pozíciója $x_0=0$, $y_0=0$.
A kijelzéshez az első lépésben a mérési adatokat kell kalibráljuk a grafikus ablakba. *Grafikus kalibrálás:* Feltételezzük, hogy a méréseket egy m változóba kapjuk. A mérések tartománya: $m_{MIN}…m_{MAX}$. Ezt a tartományt kell transzformáljuk a $0…$ Y_{MAX} grafikus tartományba. Figyelembe kell venni, hogy a grafikus kijelzőkön az Y függőleges tengely iránya fenntről lefele néz. Így, ha m_y-vel jelöljük a kijelzendő értéket:

$$m_y = \left[Y_{MAX} - \frac{Y_{MAX}}{m_{MAX} - m_{MIN}} \cdot m \right]_{INT}$$

INT azt jelenti, hogy a kapott érték egész részét kell venni.

Az adatok tárolása a kijelzéshez: Mivel egyszerre több adatot kell ábrázolni, a grafikus kalibráció után a méréseket érdemes egy listába tárolni. A lista végére mindig az új mérést tesszük. Ha a lista hossza elérte az X_{MAX} értéket, a lista fejét levágjuk. Így a listában mindig a legutolsó X_{MAX} darab mérés lesz.



Az aktuális listában található mérési adatok ábrázolásához a *LineTo()* függvényt alkalmazhatjuk. Ha a megjelenítést frissíteni szeretnénk (amikor új mérési adat érkezett és a lista tartalma megváltozott) az előző adatokat a képernyőről törölni kell. Ezt úgy tehetjük meg, hogy az ábrázolás előtt egy kitöltött téglalapot rajzolunk a képernyőre a háttér színével. Ehhez alkalmazhatjuk például a
Tárolásra alkalmazott listaként használhatjuk a *CList* osztályt. Néhány függvény az osztályból:
CList<int, int> m_list - egész elemekből álló lista generálása
int x = m_list.RemoveHead() – elem hozzáadása a lista fejére
m_list.AddTail(x) - az lista utolsó elemének eltávolítása/kiolvasása
int n = m_list.GetCount() – a lista hosszának lekérdezése

Feladat: A *TemperatureControlLab* tervben oldja meg egy periodikusan generált véletlenszerűen generált számsorozat grafikus kalibrálását, mentését listába és ábrázolását. A grafikus ablak méreteit a MAX_TEMP és MAX_TIME_TEMP konstansokból kapjuk.
A megoldását egy, az OnTimer függvényben meghívható függvényben (*GetTemperatureList*) implementáljuk, az alábbi módon:
- Generálunk egy véletlenszerű számot a *rand()* függvénnyel.
- a grafikus kalibráláshoz alkalmazzuk, hogy a rand() függvény a számot 0..RAND_MAX tartományban generálja.
- a generált elemet elhelyezzük a listába.
-ha a lista hossza nagyobb, mint a MAX_TIME_TEMP vágjuk le a lista végét.

A *TemperatureControlLab* projektben listában található elemek grafikus ábrázolása az *OnTimer* függvényben automatikusan történik.

4. Kérdések és feladatok

1. Vizsgálja meg, hogyan kell a Timer függvényt alkalmazni C#/.NET fejlesztő környezetben.
2. Oldja meg a grafikus kijelzést úgy, hogy a mérési adatok tárolásához nem listát, hanem tömböt alkalmaz.
3. Módosítsa a programot úgy, hogy a listában nem egy, hanem több mérési adatot tárol, és mindegyiket kiteszi ugyanarra a grafikus ablakra.