Rendszer- és alkalmazástechnika labor 2 beugrók

Hibák lehetnek!

1. Ipari szerelőrobot programozása I. - NOKIA PUMA 560

Nincs beugró? V:Nekünk nem volt.

- Nekünk sem volt
- Nekünk sem volt
- Nekünk sem van
- Nekünk sem lesz
- Nektek lesz. Csak nem

2. Ipari szerelőrobot programozása II. - Mitsubishi MELFA

1. Hány csuklója van a MELFA robotnak? Miért szükséges ennyi csukló, és miért nincs szükség ennél többre?

2. Hogyan mozog a robot a következő koordináta rendszerekben: JOINT, XYZ,TOOL?

3. Ismertesse rajzban (vektorok segítségével) hogyan határozható meg a robot szerszámközéppontjának aktuális középpontja a következő koordináta rendszerek (keretek) felhasználásával: Világ, Bázis, Mechanikai interfész, Szerszám koordinátarendszer!

4. Valósítsa meg egy induktív közelítésérzékelő megszakításkezelését MELFA BASIC V programnyelven. A megszakításnak az érzékelő 1 kimenete esetén kell érvényre jutni. Az induktív érzékelő a robot 31 bemenetén érhető el. Definiálja a megszakítást az 1 sorszámúként és rendelje hozzá a INDSENSINT (Inductive Sensor Interrupt) nevű megszakítás kezelő szubrutint! A megszakítás kezelő rutin az alábbiakat hajtsa végre a következő sorrendben:

- Robot 1. megfogójának kinyitása
- P1 pontba mozgás csukló interpolációval
- OUT2 kimenet értékének 1-be állítása
- A szubrutin megírása előtt feltételezheti, hogy az 1. sorszámú megszakítás engedélyezve van, azonban a megaszkítás kezelő szubrutinban le kell tiltani.

5. Mi a működési elve a robot ütközés detektálás funkciójának? Milyen utasítással állítható be a detektálás küszöbszintje? Hogyan használható programból az ütközésdetektálás funkció? A robotvezérlő melyik változóból olvasható a maximális különbség a motorok becsült nyomatéka és a tényleges nyomatékok között?

6. Milyen utasítással működtethető a mozgáskövetés (Motion Tracking)? Mik a paraméterei? Milyen változóban érhető el a futószalag motoraira szerelt encoderek pillanatnyi értéke?

Nincs beugró.
Nekünk sem volt.

3. Digitális szabályozó algoritmusok

Melyek az UDP és TCP közötti lényeges különbségek?
Az UDP (User Datagram Protocol/ Internet Protocol) a TCP/IP (Transmission Control Protocol / Internet Protocol) mellett elterjedten használt szabvány.

UDP ~ "képeslap"	TCP ~ "telefonhívás"
Kapcsolat nélküli szolgáltatás, nem jön létre munkamenet az állomások között.	Kapcsolatorientált szolgáltatás. munkamenet jön létre az állomások között.
Nem garantálja, és nem igazolja vissza sem a kézbesítést, sem az adatok ütemezését.	A visszaigazolások és az ütemezett adatküldés által garantálja a kézbesítést.
Az adatküldés biztonságának biztosítása a programok feladata.	A programok számára az adatküldés biztonságát a protokoll garantálja.
Gyors, kicsi a költségigénye, támogatja az egy pontból több pontba kommunikációt is.	Lassabb, nagyobb költségigényű, csak a két pont közötti kommunikációt támogatja.

UDP-t általtában azok a programok alkalmazzák, amelyek kis mennyiségű adatot küldenek, vagy valós idejű követelményeknek kell megfelelniük.

Milyen parancsokat használunk a PC – PLC kommunikáció során?
GetData - A parancs vétele után visszaküldi a folyamat aktuális adatait egy 220 byte-os csomagban
SetMode - Beállítja a paraméterül kapott szabályzó üzemmódját, ha az REMOTE állapotban van
SetOut - a fölérendelt gép által küldött kimenő jelet (zo) átadja a paraméterben megadott szabályzónak.
SetError - Az összes távvezérlő szabályzót ERROR állapotba állítja

Hogyan szinkronizálódik a két számítógép?

A PLC-ben a kommunikációt az 1. számú szervező blokkban (OB1) kezeljük. Ez a szabadonfutó programrész msec nagyságrendű időnként ismétlődik. Az irányító programokat az óra megszakításra csatolt OB35 blokk futtatja 100 msec-ként. Ez a programrész megszakítja az OB1-ben szervezett programot. Azaz egy PC-ről érkező irányító jel a következő OB35 futáskor jut érvényre. A PC-ben a mintavételi időt 100 msec-nél nagyobbra, pl. 1 sec-ra állítjuk be. Ekkor max. 100 msec további holtidőt iktatunk be a rendszerbe, ami a folyamat dinamikájához mérve elenyésző.

Megjegyzés, hogy nem beszélhetünk amúgy semmilyen szinkronizációról, mert ez csak sima kommunikáció.

Hogyan lehetne pontosabb szinkronizációt biztosítani?
Az OB35 blokk 100ms-es megszakítási prioritásban van, ha egy alacsonyabbszámú, kisebb periódusidejű blokkba tesszük a programot az megoldás. A másik, hogy nem csak a PLC, hanem a PC órajelét is nézhetnénk, de akkor át kellene szervezni az egész kódot. Szóval ekkora másodperces időállandónál ezeknek semmi értelme… (Bézi mondta így)

Hogyan használjuk a QNX időzítőjét? Hogyan jelezhet?
A QNX Neutrino az idő és a dátum kezelését 1970. január és 2554 január között támogatja. A POSIX szabványban jelenleg a 2038. év a határ. Ha olyan rendszer, illetve szoftver készül, amelyiknek valamelyik határérték után is működnie kell, akkor különös figyelmet igényel a dátum kezelése! A rendszer óra átállítása, elkerülendő az időugrásokat, ugyanúgy a lépések felgyorsításával, vagy lelassításával történik, mint más POSIX illetve UNIX rendszerekben.
A QNX Neutrino biztosítja a POSIX időzítő szolgáltatásokat. Időzítő lejárhat: abszolút időpontban, relatív időtartamot mérve, ciklikusan egy időintervallumot mérve. A ciklikus működés abban az esetben nagyon hasznos, ha periodikus működés szükséges. Ha a szálak prioritása miatt az adott szál nem is kap futási jogot azonnal, ahogy az időzítő lejárt, nem kell újraprogramozni az időtartamot, újraindítani az időzítőt, mert az a háttérben tovább fut. A lejáratot jelző esemény később is, folyamatosan feldolgozható.

Mi az elintegrálódás? Hogyan küszöböljük ki az egyes szabályozó algoritmusokban?
Az elintegrálódás, amikor valamilyen nagy alapjel változásra (pl egységugrásalapjel) az irányító jel a telítési határán túlhalad.A folyamatra viszont csak a D/A átalakióval vagy az erősítővel korlátozott értéke jut. A belső és a külső irányító jel értékének különbsége az integrálódás mértéke.

PI-nél és PID-nél, csak berakunk két if ágat a szabályzó végére, hogy ne fusson tovább egy maximumon/minimumon. Ezt mondják korlátozásos szabályozásnak.
Belső korlátozás beiktatására, illesztésére használják a FOXBORO szabályzót.

Hogyan oldjuk meg a szabályozó lökésmentes indítását?
Bemérjük a végrehajtó jel előző értékét, és az algoritmust innen indítjuk.

Milyen üzemmódjai vannak a PLC-ben megvalósított szabályozóknak?

A szabályozók üzemmódjai:

M (manual): kézi üzemmód, a végrehajtó jelet a kezelő állítja be.

A (automatic): automatikus üzemmód, a végrehajtó jelet a szabályozó algoritmus adja.

R (remote): a szabályozó hálózaton keresztül távvezérelhető. Ez esetben az alábbi üzemmódokban működhet:

- **NC** (No Control): mint az automatikus üzemmód, de felkészült a távvezérlésre.
- **SPC** (Set Point Control): a küldött alapjelre szabályoz.
- **DDC** (Direct Digital Control): a fölérendelt gép számítja és küldi a végrehajtó jelet, a PLC szabályozója ez esetben háttérszabályozó (back-up controller).
- **E** (Error): hibajelzés esetén átveszi az irányítást az automatikus üzemmódnak megfelelően, csak kezelői nyugtázás után kapcsolható ismét távvezérelhető üzemmódba.

Milyen üzemmódok között kell kapcsolni a PLC szabályozóját a mérés során?

Remote DDC kell egyedül

Mi történik, ha az irányító programot üzemmód átkapcsolás nélkül leállítjuk? Van egy, eddig nem említett megoldás, mi lehet az? Van egy másik kézenfekvő (egyszerű) védelem is, kitalálja?

Itt Timeout van nem watch-dog de az a megoldás.

Kérdés volt még, hogy írjuk le az egyes szabályozók programrészletét. (Mind a 3 ami az útmutatóban van

PI

A programrészlet:

```
ek=rk–yk;
ik=ik1+ki*ek;
```

A telítődés kezelése:

```
vk=kc*ek+ik;
if(vk<Umin) uk=Umin;
else if(vk>Umax) uk=Umax;
else uk=vk;
ik1=ik+(uk–vk);
```

FOXBORO

A programrészlet:

```
ek=rk–yk;
uk=kc*ek+fk;
if(uk<Umin) uk=Umin;
if(uk>Umax) uk=Umax;
fk=beta*fk+(1–beta)*uk;
```

Rekurziv poz:

```
ek=rk–yk;
uk=uk1+b0*ek+b1*ek1;/PI szabályozóban b2=0
if(uk<Umin) uk=Umin;
if(uk>Umax) uk=Umax;
uk1=uk;
ek1=ek;
```

Integralo rekeszes:

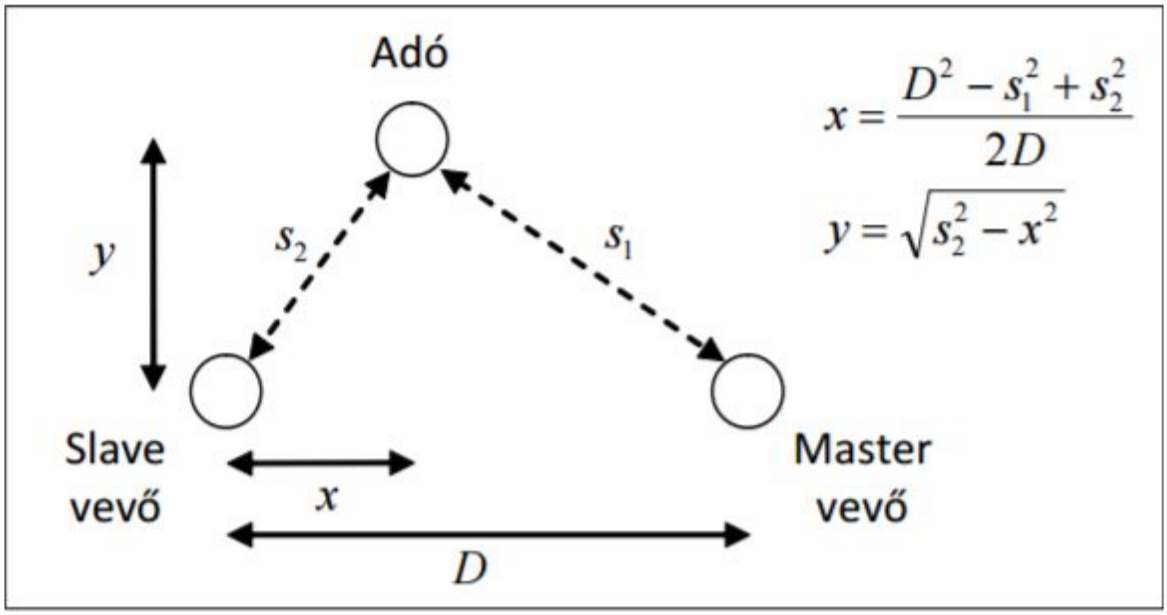
```
ek=rk–yk;
ik=ik+ki*ek;
vk=kc*ek+ik;
if(vk<Umin) uk=Umin;
else if(vk>Umax) uk=Umax;
else uk=vk;
ik=ik+(uk–vk); //ez az integrátor visszaállítás
```

Foxboro:

```
ek=rk–yk;
uk=kc*ek+fk;
if(uk<Umin) uk=Umin;
if(uk>Umax) uk=Umax;
fk=beta*fk+(1–beta)*uk;
```

4. Autonóm robot navigációs rendszere I. - Ultrahangos helymeghatározás

1. **Hogyan lehet meghatározni egy két vevőből és egy adóból álló ultrahangos helymeghatározó rendszer esetén az adó- és vevőegységek távolságából az adó pozícióját? (ábra, képlet)**



2. ábra: Trilateráció

2. **Miért szükséges a környezeti hőmérséklet ismerete a helymeghatározás során? Hogyan függ a távolságmérés abszolút hibája a hőmérséklettől, a hőmérsékletmérés hibájától és a távolságtól?**

   A hangsebességen alapul a mérés. A hangsebesség pedig hőmérséklet függő.

$$c = 331{,}3\frac{m}{s} \cdot \sqrt{\frac{T}{273{,}15K}}$$

   A távolság mérési hiba egyenesen arányos a mérni kívánt távolsággal

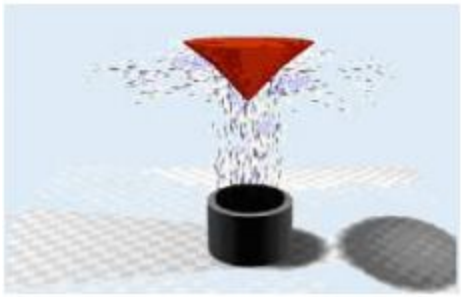
   +- 1°C hőmérsékletmérési hibánál 5 méteres távolságban +- 1 cm pontosságban tudunk mérni.

3. **Mi okozza az adó által kiadott négyszögimpulzus-csomag “elkenődését” a vevőoldalon?**

   A jelátviteli lánc sávszűrő karakterisztikája.

4. **Hogyan küszöbölhető ki az ultrahangos adókapszula szelektív iránykarakterisztikájának zavaró hatása?**

   Az adóegységben található kapszula függőlegesen sugároz egy megfelelően kemény kúpra, így a lesugárzott hang karakterisztikája hengerszimmetrikussá válik.



5. **Mekkora távolsághibát okoz 40kHz-es hangjel esetén, ha a vett jel maximumhelyének meghatározásakor +-1 periódust tévedünk a jelre szuperponálódó zaj miatt?**

   Közel 1 cm hibát, hiszen a hang hullámhossza 0.85 cm.

6. **Hogyan történik a hang terjedési idejének mérése a vevőegységekben? Milyen hasznos és hibakomponensekből határozható meg a tényleges idő?**

master ütemez -> szinkronizál az adóval, -> felszólítja a slavet az időmérésre majd megkezdődik az időmérés. ->A vevőegységben Window Interrupt generálódik, mely leállítja az időmérést.

Levonjuk az offset hibákat és a korrekciós hibát (hibakomponensek). A kapott eredmény az idő (többi hasznos komponens).

$$t_{corr} = \frac{s_m - s_0}{c} \tag{1.6}$$

$$t = t_1 + T_s \cdot k_{\max} - t_{O1} - t_{O2} - t_{corr} \tag{1.7}$$

ahol s_0 a valódi, s_m a mért távolság, c a hangsebesség, t_{corr} pedig a kalibrációból adódó korrekciós időtag.

ahol t_1 az A/D átalakításig eltelt idő, T_s a mintavételi idő, $k_{\max}$ pedig a maximális értékű minta indexe.

5. Autonóm robot navigációs rendszere II. - A dead reckoning elv

Ellenőrző kérdések

1. Milyen előnyökkel és hátrányokkal rendelkeznek a dead reckoning algoritmusok?

A dead reckoning módszerek lényege az, hogy a robot pillanatnyi elmozdulása és elfordulása (odometria), vagy pillanatnyi sebessége és gyorsulása ismeretében (inerciális navigáció) integrálással határozzuk meg az aktuális pozíciót és orientációt. Ilyen módon az abszolút rendszereknél általában **olcsóbb, egyszerűbb és gyorsabb helymeghatározás**t érhetünk el. Számolnunk kell viszont azzal, hogy a rendszer az **integrálás miatt kumulált hibával rendelkezik**

2. Milyen szisztematikus és nem szisztematikus hibái vannak a dead reckoning eljárásoknak?

Szisztematikus:

- A kerekek egymástól eltérő átmérője
- A kerekek tényleges átmérőjének eltérése a névlegestől
- A kerekek tényleges távolságának eltérése a névlegestől
- A kerekek ideálistól eltérő elhelyezkedése
- Az enkóderek véges felbontása
- Az enkóderek mintavételezése

Nem szisztematikus

- A talaj egyenetlensége
- Kerekek érintkezési pontjának bizonytalansága
- A kerekek csúszása
    - Síkos talaj miatt
    - Nagy gyorsulás
    - Külső hatások

3. **Röviden foglalja össze a dead reckoning eljárás algoritmusát!**

Az algoritmussal a kerekek szögelfordulását mérjük, amelyből kiszámíthatjuk a robot által megtett utat és az elfordulást. A műveletet kellően gyakran elvégezve, az időszelet alatti elmozdulások összegzésével meghatározhatjuk a robot aktuális pozícióját és orientációját egy ismert kiindulási helyzethez képest.

4. Hogy működik az UMBmark kalibrációs eljárás, és milyen értékeket lehet a segítségével kalibrálni?

A kalibrációs eljárás során egy előre beprogramozott, 1 × 1 méter oldalhosszúságú (ez persze lehet hosszabb is, úgy a kalibráció is jobb lesz) négyzet alakú pályán kell végighaladnia a robotnak, az óramutató járásával megegyező, és azzal ellentétes irányban egyaránt. A robotnak a saját dead reckoning algoritmusa szerint kell egyenesen haladnia 1 métert, illetve fordulnia derékszögben, nem pedig a valós egyenes és derékszög szerint! A műveletet többször el kell végezni, és minden esetben fel kell jegyezni a kiindulási és a végső helyzet közötti eltérést. A névlegestől eltérő keréktávolság kanyarodáskor szöghibát (úgynevezett A típusú hibát) okoz, a kerekek eltérő mérete miatt pedig nem egyenes vonalú a mozgás a négyzet oldalain (B típusú hiba). Ezt a két esetet szemlélteti a 4. ábra. A két hiba egyszerre jelentkezik, de geometriai megfontolások alapján egymástól különválasztható A keresett kalibrált értékek: Dr és Dl a jobb, illetve bal kerék átmérője, Wactual pedig a tényleges keréktávolságot jelenti

$$D_r = \frac{2}{\frac{1}{E_d} + 1} \cdot D_{nominal} \qquad (2.11)$$

$$D_l = \frac{2}{E_d + 1} \cdot D_{nominal} \qquad (2.12)$$

$$W_{actual} = W_{nominal} \cdot E_w \qquad (2.13)$$

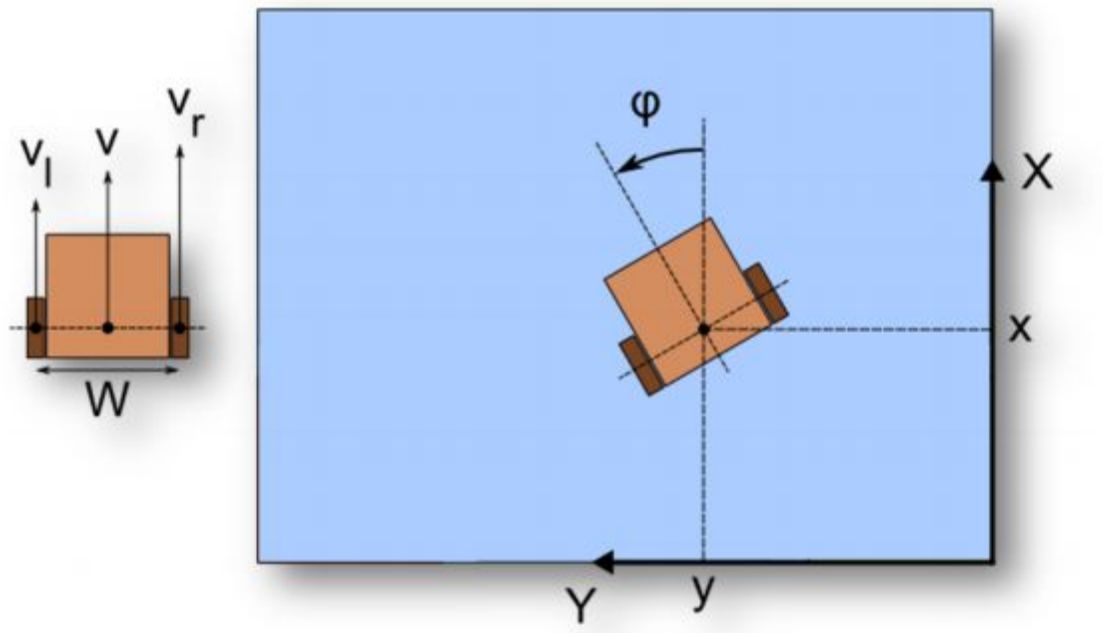
5. Ismertesse a differenciális hajtású robot kinematikáját leíró egyenleteket, illetve vázolja fel az értelmezéséhez szükséges koordináta rendszert!

$$v = \frac{v_r + v_l}{2} \qquad (1.1)$$

$$\omega = \frac{v_r - v_l}{W} \qquad (1.2)$$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{pmatrix} = \begin{pmatrix} v \cdot \cos\varphi \\ v \cdot \sin\varphi \\ \omega \end{pmatrix} \qquad (1.3)$$

ahol v a robot pályamenti sebessége, a robot szögsebessége, r v és l v a jobb, illetve bal kerék kerületi sebessége, W a két kerék távolsága, x és y a robot referencia pontjának koordinátái, pedig a robot orientációja (1. ábra).



1. ábra: A robot koordinátarendszere

6. Írja fel a PID szabályozó folytonos idejű átviteli függvényét, valamint a szabályozó diszkrét alakját téglalap szabály szerinti integrálás, és kétpontos deriválás alkalmazásával!

A folytonos PID szabályozó átviteli függvénye:

$$C(s) = k_c(1 + \frac{1}{sT_I} + sT_D)$$

A differenciáló tag ideális, nevezőjében nem szerepel az időállandó, mert diszkrét alakban fogjuk megvalósítani. A téglány szabály szerinti integrálás és a kétpontos deriválás alkalmazásával a PID szabályozó diszkrét alakja:

$$C(z) = \frac{u(z)}{e(z)} = k_c\left[1 + \frac{T_S}{T_I(1-z^{-1})} + \frac{T_D}{T_S}(1-z^{-1})\right] = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1}}$$

7. Ismertesse pár mondatban a mérés során alkalmazott mozgásszabályozásokat egyenes vonalú mozgás és saját tengely körüli forgás esetén! Hogyan határozzuk meg a robot beavatkozójeleit a két mozgás esetén?

Egyenes vonalú mozgás esetén egy PID szabályozó segítségével történik a mozgás, míg saját tengely körüli forgás esetén egy PI szabályozót alkalmazunk. Mindkét szervó tartalmaz egy-egy analóg sebességszabályozót, így ezzel a szoftverben nem kell foglalkoznunk.

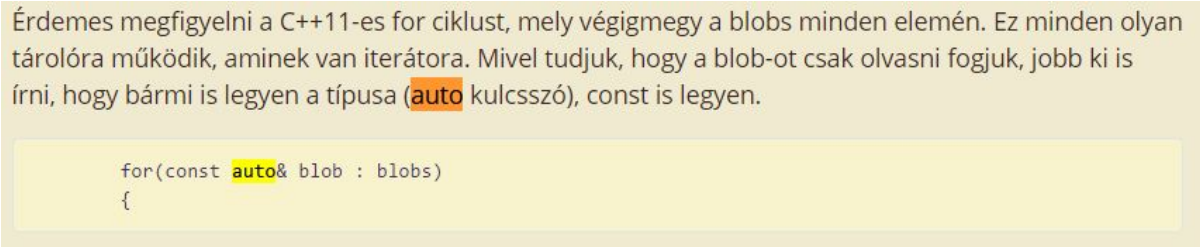
Egyenes vonalú szabályozás esetén az egyenestől való távolságot minimalizálja a PID szabályozó, ahogyan az 5. ábrán is látható. A szabályozásra leginkább azért van szükség, mert a két szervó analóg sebességszabályozója nem ideális, így a két kerék különböző sebességgel haladhat. A PID szabályozó beavatkozójele (kimenete) a robot szögsebessége, a hibajele az egyenestől való távolság. A robot sebességalapjelét egy trapéz alakú sebességprofil alapján határozzuk meg. Ez a gyakorlatban azt jelenti, hogy a mozgás elején, végén egy gyorsító és lassító sebességszakaszt, a kettő között pedig konstans sebességalapjelet alkalmazunk. Ezzel elérhető, hogy a robot a kívánt célba pontosan álljon meg, és ne a célpont elhagyása után kezdjen fékezni. A robot sebesség és szögsebesség alapjeleiből már meghatározható a két kerékre/szervóra vonatkoztatott sebességalapjel.

Saját tengely körüli forgás esetén a szabályozás igen hasonló az egyenes vonalú szabályozáshoz. Ebben az esetben egy PI szabályozót alkalmazunk, amely a robot sebességét szabályozza. A szabályozó hibajele pedig a két kerék által megtett utak különbsége két mintavétel között. A szabályozóval az a célunk, hogy a két kerék azonos nagyságú utat járjon be, tehát a kialakuló pozícióhibát minimalizáljuk. Ennél a mozgásnál a robot szögsebességét határozzuk meg egy trapéz alakú szögsebességprofil alapján. Ezzel a fokozatos elindulást, lefékezést szeretnénk megvalósítani, ahogyan az egyenes vonalú mozgás esetében is.

6. mérés: Objektumorientált szoftverfejlesztés és verziókövetés

Mi az a staging?
Repora változások létrehozása, fájlokat add parancsal stageljük.
Amelyik változtatásokat stageled, azok fognak a commit során mentődni.

Mi az auto kulcsszó és mire használható?
git config esetén alkalmazzuk, alapbeállítás??
cpp11



?

Mik a lambda függvények és mikor érdemes őket használni?
A lambda kifejezés gyakorlatilag egy olyan függvény, aminek nincsen neve. Három részből áll:

- a [] zárójelek között azt adjuk meg, hogy mely, a létrehozáskor látható változókat akarjuk majd ott is elérni, ahol fut a kifejezés (vagyis melyeket “vigye magával”).
- a () zárójelek között a paraméter listája van
- a {} zárójelek között pedig a törzse

Sokszor a létrehozás helyén látható változókból semmit nem kell magunkkal vinni, de ha mégis, változókat referencia vagy érték szerint is magával tud vinni a lambda kifejezés. A C++11-ben a [&] az összes változót elérhetővé teszi (referencia szerint).

Mi a különbség a shared_ptr és a unique_ptr között?
A unique_ptr birtokolja a rá bízott objektumot (tulajdonos szemantika). **Egy pointert pontosan egy unique_ptr birtokolhat**, ezért a unique_ptr nem másolható. A unique_ptr megszűnésekor a rá bízott memória felszabadul. Mivel a unique_ptr-t általában a stacken hozzuk létre (tehát nem new hívással), pontosan tudjuk, hogy mikor fog megszűnni: a blokk végén vagy kivétel keletkezésekor.
A shared_ptr a unique_ptr-rel szemben nem egyedül birtokolja a nyers pointert, hanem egy vagy több shared_ptr osztozik és **akkor szabadul fel a memóriaterület, ha az utolsó shared_ptr is megszűnik**. Ezt az ún. referenciaszámlálással oldja meg, tehát a memóriaterület mellett **létezik egy számláló, ami számon tartja, hogy éppen hány shared_ptr hivatkozik az adott memóriaterületre**. Akkor hívódik a destruktor, amikor ez a számláló nullára csökken.
A weak_ptr olyan hivatkozás egy shared_ptr-ek által birtokolt területre, ami nem növeli a referenciaszámláló értékét. A weak_ptr kiürül (nullptr-ré változik), ha az általa mutatott területre megszűnik minden shared_ptr.

Mikor hívódik a destruktor? Legalább három esetet sorolj fel!
shared_ptr: Akkor hívódik a destruktor, amikor a számláló nullára csökken.
Stacken lévő változó esetén akkor hívódik a destruktor, amikor a változó kimegy a scope-ból**:**

- a függvény végén,
- a {}-vel határolt blokk végén,
- kivétel keletkezésekor a stack visszacsévélésével (stack unwinding),
- dinamikusan foglalt (new) objektumok esetén a delete operátor használatakor.

Mi az a tulajdonos szemantika?
Nyers pointerek használata rengeteg nehezen észrevehető hibalehetőséget hordoz magában. Az egyik hibalehetőség a memóriaszivárgás, egy másik hibalehetőség a pointerrel visszatérő függvények kezelése. A pointerrel visszatérő függvények használatakor felmerül a kérdés, hogy **ki felelős a memóriaterület felszabadításáért**, vagyis ki a tulajdonosa az objektumnak (tulajdonos szemantika).
A unique_ptr birtokolja a rá bízott objektumot (tulajdonos szemantika).

Nevezz meg 3 db C++11 nyelvi elemet!
extended initializer list, auto kulcsszó, explicitely defaulted and deleted functions, lambda

7. Alkalmazásfejlesztés C++ nyelven beágyazott környezetben

A mérés elején feltett kiskérdések az alábbiakhoz hasonlóak lesznek, de nem csak pontosan ezek lehetnek.

• Hol lehet módosítani, hogy melyik nézet legyen a kezdő nézet (debug vagy nice)?

A SwitchSensor konstruktora az inicializáló listában kezdőértéket ad a kapcsolónak.

```
5  SwitchSensor::SwitchSensor(SwitchButton& switchButton)
6  : isOn(false), switchButton(switchButton)
7  {
8  }
```

Az SW4 kapcsoló két megjelenítési mód között vált (Debug vagy „Nice”).

• Mi a felelőssége az ObservingView osztálynak? - ezt kérdezte
Egy View ki tudja magát rajzolni (Redraw metódus) egy paraméterként kapott Display-re. A Display egy absztrakt ősosztály, mely megjelenítési funkciókat tartalmaz.
Mivel sok View egy szenzort (valamit, ami
Observable) figyel és jelenít meg, az ObservingView egy olyan absztrakt ősosztály, mely egyben View és Observer is.
Akkor rajzolják majd újra magukat, ha változott az általuk megjelenített érték.

• Mi az egyetlen felelőssége a DebugAccelerationView osztálynak? - meg ezt
Debug nézetben a gyorsulás megjelenítése

• A NiceView private View típusú attribútumai miért nem referenciák?
A parent lehet NULL, a legfelsőnek nincs szülője. Referencia pedig nem lehet NULL.

• A View parent attribútuma miért pointer és miért nem referencia?
Mert lehet olyan View, amelynek nincs szülője. pl TopLevelView

• Mire jó az Observable osztály?
Minden Sensor egyben Observable is, mivel a Poll() metódus ha változást érzékel, akkor a saját notifyObservers() metódussal tájékoztatja a feliratkozott Observereket a változásról. (Observer tervezési minta.) Ezt elsősorban a nézetek fogják kihasználni, akik akkor rajzolják majd újra magukat, ha változott az általuk megjelenített érték. Így nem kell majd nekik folyamatosan lekérdezni a hozzájuk tartozó szenzor értékét, mert értesítést kapnak minden változásról. A feliratkozottak értesítést kapnak, ha változás történik.

• A Misc osztály időzítő funkciója egy timer interruptot használ a háttérben. Az interrupt kezelő függvény hogyan tudja elérni a Misc egyik példányának Tick() metódusát?
A Misc-nek egyetlen példánya van (singleton), amely a GetInstance metóduson keresztül érhető el

```
extern "C" void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *handle)
{
    UNUSED(handle);
    Misc::GetInstance().Tick();
}
```

• Honnan tudják a nézetek (View leszármazottak, pl. DebugTemperatureView), hogy hova kell kirajzolniuk a tartalmukat?
Konstruktorban kapnak Location-t

8. Alkalmazásfejlesztés Qt környezetben

A mérés elején feltett kiskérdések az alábbiakhoz hasonlóak lesznek, de nem csak pontosan ezek lehetnek.

• Miért szükséges, hogy a SerialConnector (a soros portot kezelő osztály) a QObject-ből származzon le?

Mivel signalokat csak QObjectből származó objektumokhoz lehet kötni.

• Miért nem célszerű connect-nek hívni a SerialConnector csatlakozást létrehozó függvényét?

Mert a QObject-ből származó osztályokban a connect a signal-slot összekötést valósítja meg.

• Hogyan lehet összekötni egy signal - t egy slot - tal?

connect(forrás objektum, signal, cél objektum, slot)

• Mire való a qml.qrc állomány?

QML fájl és az elérési útja (URI-je): a QRC fájl egy un. resource leíró fájl. Ezt a bal oldali könyvtárszerkezetben is láthatjuk. A lényege, hogy az ebben a qml.qrc fájlban szereplő fájlok belefordulnak az exe fájlba, így a program viszi őket magával, nem kell külön fájlként odamásolni őket a telepítéskor. Ilyen erőforrások tipikusan a QML fájlok és például képek szoktak lenni.

• Miért jobb a signal-ra várni, mint közvetlenül a megfelelő mennyiségű bejövő adatra egy lassú soros porti kommunikáció esetében?

Ne akadjon meg a kezelőfelület.

• Hogyan tudjuk megoldani egy QML állományban, hogy két elem (pl.: Rectangle) felső oldala egy vonalban legyen? Mutasson rá példát!

```
RowLayout{
        Rectangle{}
        Rectangle{}
}
```

• Miért, illetve mikor kell nevet adni (object name) egy QML objektumnak?

- objectName: C++ oldalról ez alapján a név alapján tudjuk megkeresni az elemeket.

9. Felhasználói felület megvalósítása QML nyelven

A mérés elején feltett kiskérdések az alábbiakhoz hasonlóak lesznek, de nem csak pontosan ezek lehetnek.

• Miért szükséges, hogy Message osztályban QPROPERTY-ket deklaráljunk?

Hogy a ListView meg tudja hívni majd a megfelelő függvényeinket.

• Mi a különbség a HTTP GET, illetve a POST üzenet között?

GET: Adatok lekérése egy megadott forrásból
POST: Adatok elküldése feldolgozásra egy adott erőforrás számára

• Mikor van rá lehetőség, hogy egy QJsonDocument-et QJsonArray-á konvertáljunk?

Mikor az utolsó readAll() hívás is lefutott, mivel ezután áll össze az üzenetek tömbje.

• A ListView-nak miért, illetve mikor érdemes a model-t kódból megadni, mik az előnyei?

A model kódból való megadása esetén
van könnyedén változtatni a ListView paramétereit, pl ha futási időben változnak a ListView paraméterei (?)

• Miért kell a ListView model-jét közvetlenül frissíteni? Figyeli-e automatikusan a model változásait?

• Mi a különbség a TextField és a TextEdit között?

TextField: már kész, egyből használható layout-okban, form-okban. Automatikusan alkalmazkodik a célplatformhoz, egyszerűbb API-val rendelkezik

TexEdit: Bonyolultabb API-val rendelkezik, nekünk kell megadni a paramétereket (background, frame size, hint, default behavior), egyedi tervezésű UI-k esetén ezt érdemes használni.

10. Kliens-szerver alkalmazás fejlesztése Qt/QML környezetben

A mérés elején feltett kiskérdések az alábbiakhoz hasonlóak lesznek, de nem csak pontosan ezek lehetnek. Valaki ezt a sok szennyet ki tudná dolgozni, illetve aki volt már, mik a tapasztalatok?

Mi az az url kódolás? Honnan lehet felismerni, mutasson egyetlen egyszerű példát!

Az elérési útvonalakban (URL) csak ékezet nélküli latin betűk, arab számok és a következő speciális karakterek szerepelhetnek:

```
- _ . ~
```

Néhány további karakternek sajátos jelentése van az URL-ekben. Ezek a fenntartott karakterek a következők:

```
! * ' ( ) ; : @ & = + $ , / ? % # [ ]
```

Ha a fenntartott karakterek egyike kijelölt céljától eltérő szerepben lenne jelen egy adott URL-ben, helyére egy százalékjel (%) és a karakter ASCII kódja kerül hexadecimális formátumban.

Hasonlóképpen kódolandóak azok a karakterek is, melyek nem szerepelnek az eddig felsoroltak között.

pl: https://hu.wikipedia.org/wiki/URL-k%C3%B3dol%C3%A1s

Hogyan lehetséges, hogy az std::function mint változó, „meghívható mint függvény”?

```
Az std::function az tulajdonképpen egy "functor", vagyis implementálva van az "operator()" függvénye
így az objektum meghívható úgy mint egy függvény
```

Mit jelent C++-ban ha egy osztály absztrakt? Hogyan jelöljük, miről ismerhető fel? Hogy lehet példányosítani?

```
class AbstractClass {
public:
  virtual void AbstractMemberFunction() = 0; // Pure virtual function makes
                                             // this class Abstract class.
  virtual void NonAbstractMemberFunction1(); // Virtual function.

  void NonAbstractMemberFunction2();
};
```

Azért absztrakt, mert közvetlenül nem lehet példányosítani (nincs konstruktora), csak a leszármazottakat.

Mire jók a „virtual”, „override” és „const” jelölők függvények deklarációjakor?

virtual: a leszármazott felülírhatja
override: az ős függvényét felülírjuk vele
const: nem változtathatók benne az osztály privát változói

Melyek az std::queue 3-4 legfontosabb függvényei, amelyek egy minimális működéséhez használatosak?

The std::queue class is a container adapter that gives the programmer the functionality of a queue - specifically, a FIFO (first-in, first-out) data structure.
The class template acts as a wrapper to the underlying container - only a specific set of functions is provided. The queue pushes the elements on the back of the underlying container and pops them from the front. <- A Jóisten áldjon meg

- back() : access the last element
- front() : access the first element
- push_back() : inserts element at the end
- pop_front() : removes the first element

(http://en.cppreference.com/w/cpp/utility/functional/function)

Hogy épül fel egy lambda kifejezés, hogyan vesz át paramétereket? Mi az a capture-list?
A lambda kifejezés gyakorlatilag egy olyan függvény, aminek nincsen neve. Három részből áll:

- a [] zárójelek között azt adjuk meg, hogy mely, a létrehozáskor látható változókat akarjuk majd ott is elérni, ahol fut a kifejezés (vagyis melyeket “vigye magával”).
- a () zárójelek között a paraméter listája van
- a {} zárójelek között pedig a törzse

Sokszor a létrehozás helyén látható változókból semmit nem kell magunkkal vinni, de ha mégis, változókat referencia vagy érték szerint is magával tud vinni a lambda kifejezés. A C++11-ben a [&] az összes változót elérhetővé teszi (referencia szerint).

[<capture list>](<argumentumok>){<függvénytörzs>}
ha visszatérési típust is szeretnénk definiálni, akkor:
[<capture list>](<argumentumok>) -> <típus>{<függvénytörzs>}

a capture listben felsorolt változókat látja a lambda

// ez beállítja x-et 4-re és visszatér (double)28-cal (remélem, nem próbáltam ki)
int x = 2;
auto y = [&x]() -> double{x *= 2; return x*7;}();

Mutasson egy példát arra, hogy a hipotetikus „Timer” osztály „timeout” signálját, hogyan kötné az éppen jelenleg írt osztály saját „beep” slot-jára!
this -> timer = <valahogy itt értéket adunk a Timer-nek>;
connect(timer, SIGNAL(timeout()), this, SLOT(beep());

Reguláris kifejezések esetén mit jelent a „capture group”?

Amennyiben a kifejezésünkben csoportokat (a Group osztály valósítja meg) hozunk létre, minden egyes csoport egy külön találatnak (Match osztálynak) számít (a Match osztály a Group –ból származik). A teljes reguláris kifejezés Match metódusa által visszaadott Match osztály is egy csoport. Ennek az osztálynak a Groups tulajdonságával egy GroupsCollection típusú tömböt kapunk vissza, aminek az első eleme maga a Match típusú objektum, vagyis a teljes találatot reprezentáló objektum. A Group osztálynak van egy Captures tulajdonsága mely egy CaptureCollection típusú tömböt ad vissza (a Group pedig a Capture –ből származik), mellyel lekérdezhetjük, hogy az adott csoportot pontosan hányszor és hol találta meg, de csak a teljes találaton belül.

Csoportosításokat a ( ) (gömbölyű zárójelek) segítségével hozhatunk létre. Egy egyszerű csoportosítás például a (\s)(kek). Ez, megtalálja a szövegben a „kek” részletet, ha előtte whitespace állt. A nullás csoport lesz az egész reguláris kifejezés, az egyes a white-space a kettes pedig a „kek” kifejezés.

A rendszer a nem nevesített csoportokat automatikusan sorszámozza, így lehet rájuk hivatkozni.
(http://nyelvek.inf.elte.hu/leirasok/Csharp/index.php?chapter=18)
(abc){3} matches abcabcabc.

Mire fog általánosságban illeszkedni a következő reguláris kifejezés? Az általános illeszkedésen kívül mutasson 2-3 valós példát is! A kifejezés: "[1-9]* " (idézőjel nélkül)

http://regexr.com/3fsbg

[0-9]	Find any character between the brackets (any digit)

n*	Matches any string that contains zero or more occurrences of *n*

[1-9]* -> 1-9 közötti számokból felépülő tetszőleges hosszú sztring
pl: 1234, 11, 111

Kérdés volt: signal-slot és lambda kifejezés