

Helymeghatározás alapjai gyakorlat

Navigációs szolgáltatások és alkalmazások (VITMMA07)

Hollósi Gergely

Budapesti Műszaki és Gazdaságtudományi Egyetem,
Távközlési és Médiainformatikai Tanszék

1. Bevezetés

A *Navigációs szolgáltatások és alkalmazások* tantárgy *Helymeghatározás alapjai* gyakorlat két fő részből áll, az első részben a hely és orientációval kapcsolatos feladatok megoldása kerül terítékre, míg a második részben a helymeghatározási feladat megoldását végezzük el.

A számítások során vagy kézi erőt alkalmazunk, vagy a Matlab programot használjuk. A Matlab kiváltható az ingyenes Octave¹ programcsomaggal.

2. Hely és orientáció feladatok

Példa Írjuk fel azt a forgatási mátrixot, amely először az Y tengely körül forgat 30 fokot, majd az X tengely körül forgat 20 fokot!

Megoldás Első lépésként írjuk fel a két tengely körüli forgatást jelképező mátrixokat:

$$R_Y = \begin{bmatrix} \cos 30^\circ & 0 & \sin 30^\circ \\ 0 & 1 & 0 \\ -\sin 30^\circ & 0 & \cos 30^\circ \end{bmatrix} \quad R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos 20^\circ & -\sin 20^\circ \\ 0 & \sin 20^\circ & \cos 20^\circ \end{bmatrix}$$

A teljes forgatási mátrix a két mátrix szorzata. Ne feledjük el, hogy a szorzásokat fordított sorrendben kell elvégezni a transzformációkhoz képest!

$$R = R_X R_Y$$

Magát a forgatási mátrixot MATLAB vagy Octave segítségével könnyen kiszámíthatjuk:

¹<https://www.gnu.org/software/octave/index>

```

1 octave:1> Ry=[cosd(30),0,sind(30); 0,1,0; -sind(30),0,cosd
    (30)]
2 Ry =
3
4 0.86603    0.00000    0.50000
5 0.00000    1.00000    0.00000
6 -0.50000    0.00000    0.86603
7
8 octave:2> Rx=[1,0,0;0,cosd(20),-sind(20);0,sind(20),cosd(20)
    ]
9 Rx =
10
11 1.00000    0.00000    0.00000
12 0.00000    0.93969   -0.34202
13 0.00000    0.34202    0.93969
14
15 octave:3> R=Rx*Ry
16 R =
17
18 0.86603    0.00000    0.50000
19 0.17101    0.93969   -0.29620
20 -0.46985    0.34202    0.81380
    
```

Példa Legyen egy három dimenziós forgatási mátrix

$$\mathbf{R} = \begin{bmatrix} 0.5 & 0.5 & \frac{\sqrt{2}}{2} \\ 0.5 & 0.5 & -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \end{bmatrix}$$

Határozzuk meg a forgatás tengelyét és szögét!

Megoldás Egy forgatási mátrix tengelye (\mathbf{v}) a forgatás során nem változik, tehát az a forgatás sajátvektora, valamint a hozzá tartozó sajátérték $\lambda = 1$, hiszen $\mathbf{R}\mathbf{v} = \mathbf{v}$, amiből $(\mathbf{R} - \mathbf{I})\mathbf{v} = 0$. Tehát keressük az a \mathbf{v} vektort, amelyre:

$$(\mathbf{R} - \mathbf{I})\mathbf{v} = \begin{bmatrix} -0.5 & 0.5 & \frac{\sqrt{2}}{2} \\ 0.5 & -0.5 & -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & -1 \end{bmatrix} \mathbf{v} = 0$$

A középső sor megfelelő skalárszorosaát kivonva az első és utolsó sorból (megtehetjük, hiszen az egyenlet homogén), kapjuk, hogy

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -1 & -\sqrt{2} \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Nyilvánvaló, hogy a tengely a $\mathbf{v} = [1, 1, 0]^T$ vektorral párhuzamos, origón áthaladó egyenes (behelyettesítéssel láthatjuk). Az egyenlet persze megoldható tetszőleges másik módszerrel (pl. Gauss elimináció, SVD felbontás). A szög megállapításához válasszunk egy erre merőleges vektort, például az $\mathbf{a} = [0, 0, 1]$ vektort (hiszen $\mathbf{a}\mathbf{v} = 0$)! Könnyű észrevenni, hogy $\mathbf{Ra} \perp \mathbf{a}$, így a forgatás szöge 90 fok.

Példa Írjuk fel a $\mathbf{v} = [1, 2, 3]$ tengely körüli $\alpha = 40^\circ$ forgatáshoz tartozó forgatási kvaterniót! Írjuk fel az ugyanekkora szöggel, de az X tengely körüli forgatást jelképező kvaterniót!

Megoldás Emlékezzük vissza, hogy egy tengely körüli forgatást jelképező kvaterniót az alábbi képlettel írhatjuk fel:

$$\mathbf{q} = e^{\frac{\theta}{2}(u_x\mathbf{i}+u_y\mathbf{j}+u_z\mathbf{k})} = \cos \frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}) \sin \frac{\theta}{2}$$

Fontos kiemelni, hogy a forgatást jelképező tengely **egységvektor** kell legyen! Tehát,

$$\mathbf{u} = \frac{\mathbf{v}}{\|\mathbf{v}\|} = \left[\frac{1}{\sqrt{14}}, \frac{2}{\sqrt{14}}, \frac{3}{\sqrt{14}} \right]$$

Ebből már nagyon egyszerű a forgatási kvaternió felírása:

$$\mathbf{q} = \cos 20^\circ + \left(\frac{1}{\sqrt{14}}\mathbf{i} + \frac{2}{\sqrt{14}}\mathbf{j} + \frac{3}{\sqrt{14}}\mathbf{k} \right) \sin 20^\circ$$

◇

Hasonlóképpen, a X tengely körüli forgatáshoz felírhatjuk a tengelyt, mint egységvektort:

$$\mathbf{u} = [1, 0, 0]$$

, amiből a forgatási kvaternió:

$$\mathbf{q} = \cos 20^\circ + \mathbf{i} \sin 20^\circ$$

Példa Forgassuk el a $\mathbf{p} = [1, 2, 1]$ vektort a $\mathbf{u} = [1, 1, 1]$ tengely körül 30 fokkal! Oldjuk meg a feladatot kvaterniók és a Rodrigues formula segítségével is!

Megoldás Első lépésként vegyük fel az egyes változókat:

```

1 u=[1,1,1];
2 p=[1,2,1];
3 a=30;
    
```

Normalizáljuk a tengely vektorát, hiszen látható, hogy $\|u\| \neq 1$. Ez egy nagyon fontos lépés, hiszen a kvaternió elkészítéséhez egységvektorra van szükségünk:

```

1 u=u/norm(u);
    
```

Ezután már létrehozhatjuk a forgatáshoz szükséges kvaterniót a $q = \cos \frac{\theta}{2} + (u_x i + u_y j + u_z k) \sin \frac{\theta}{2}$ képlet alapján MATLAB alatt:

```

1 q=[cosd(a/2),u(1)*sind(a/2),u(2)*sind(a/2),u(3)*sind(a/2)];
    
```

Természetesen a forgatni kívánt vektort is kvaternióvá kell alakítanunk a $v = p_x i + p_y j + p_z k$ formában, hogy egy tiszta kvaterniót kapjunk:

```

1 v=[0,p(1),p(2),p(3)];
    
```

A forgatást végül a $p' = qpq^{-1}$ képlet alapján tudjuk elvégezni. Ehhez szükség van a forgatási kvaternió inverzére $q^{-1} = \frac{q^*}{\|q\|^2}$:

```

1 qinv=[q(1) -q(2:4)]/norm(q)^2;
    
```

A szorzás során kihasználjuk a $(q_0 + \mathbf{q})(p_0 + \mathbf{p}) = q_0 p_0 + q_0 \mathbf{p} + p_0 \mathbf{q} - \mathbf{q} \cdot \mathbf{p} + \mathbf{q} \times \mathbf{p}$ egyenlőséget:

```

1 res = [q(1)*v(1) - dot(q(2:4),v(2:4)), q(1)*v(2:4) + v(1)*q
        (2:4) + cross(q(2:4),v(2:4))];
2 res = [res(1)*qinv(1) - dot(res(2:4),qinv(2:4)), res(1)*qinv
        (2:4) + qinv(1)*res(2:4) + cross(res(2:4),qinv(2:4))];
    
```

Itt a `pvQ` változó fogja tárolni az elforgatott vektort, ahol az elforgatott vektort az imaginárius részek tartalmazzák. A teljes kód alul látható:

```

1 u=[1,1,1];
2 p=[1,2,1];
3 a=30;
4
5 u=u/norm(u);
6
7 q=[cosd(a/2),u(1)*sind(a/2),u(2)*sind(a/2),u(3)*sind(a/2)];
8 v=[0,p(1),p(2),p(3)];
9
10 qinv=[q(1) -q(2:4)]/norm(q)^2;
11 res = [q(1)*v(1) - dot(q(2:4),v(2:4)), q(1)*v(2:4) + v(1)*q
        (2:4) + cross(q(2:4),v(2:4))];
12 res = [res(1)*qinv(1) - dot(res(2:4),qinv(2:4)), res(1)*qinv
        (2:4) + qinv(1)*res(2:4) + cross(res(2:4),qinv(2:4))];
    
```

A Rodrigues formula segítségével is könnyen elvégezhetjük a forgatást. Itt is először felveszzük a bemeneti adatokat, majd a tengely vektorát normalizáljuk.

```

1 u=[1,1,1];
2 p=[1,2,1];
3 a=30;
4
5 u=u/norm(u);
    
```

Ezután létrehozuk a keresztszorzat mátrixot az

$$[\mathbf{u}]_{\times} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

képlet alapján.

```

1 K=[0 -u(3) u(2)
2 u(3) 0 -u(1)
3 -u(2) u(1) 0];
    
```

Ezután a Rodrigues formulát alkalmazzuk ($\mathbf{R} = \mathbf{I} + \sin \phi [\mathbf{u}]_{\times} + (1 - \cos \phi) [\mathbf{u}]_{\times}^2$), majd egyszerű mátrixszorzással elvégezzük a forgatást:

```

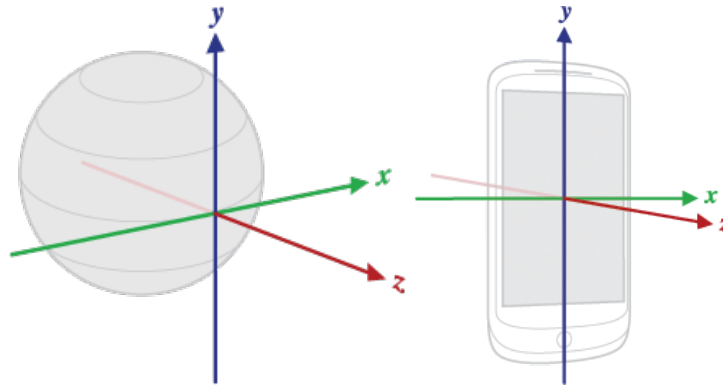
1 R=eye(3)+sind(a)*K+(1-cosd(a))*K*K;
2 pvRodr=R*p';
    
```

A teljes kód itt található:

```

1 u=[1,1,1];
2 p=[1,2,1];
3 a=30;
4
5 u=u/norm(u);
6
7 K=[0 -u(3) u(2)
8 u(3) 0 -u(1)
9 -u(2) u(1) 0];
10 R=eye(3)+sind(a)*K+(1-cosd(a))*K*K;
11 pvRodr=R*p';
    
```

Példa Egy nyugalomban lévő okostelefon gyorsulásérzékelő szenzora az $\mathbf{a} = [0.28 \frac{m}{s^2}, -0.108 \frac{m}{s^2}, 9.936 \frac{m}{s^2}]$ értéket mutatja egy adott pillanatban, míg a mágneses szenzor a $\mathbf{h} = [18.75 \mu T, 10.31 \mu T, -44.43 \mu T]$ értéket. Határozzuk meg a telefon megközelítő orientációját egy olyan koordináta-rendszerben, ahol az Y tengely észak irányába mutat, a Z tengely az föld középpontjával ellentétes irányba, és az X tengely ezekre jobbkéz szerint merőleges (a koordináta-rendszereket lásd. az ábrán)!



Megoldás Amint azt tudjuk, a mágneses vektor és a gyorsulás vektor (jelen esetben a nyugalmi helyzet miatt a gravitáció) segítségével 3 szabadságfokú orientációt határozhatunk meg. A két vektor a telefon koordinátarendszerében (az ábrán a jobb oldalon) értelmezett. Vizsgáljuk meg a két vektort!

```

1 a=[0.28,-0.108,9.936]
2 h=[18.75,10.31,-44.43]
3
4 alength=norm(a)
5 hlength=norm(h)
6
7 % alength = 9.9405
8 % hlength = 49.314
    
```

Láthatjuk, hogy a mágneses térerősség nagyjából $49.314\mu T$ értékű, míg a gyorsulás $9.9405\frac{m}{s^2}$ értékű, amely utóbbi közel esik az elfogadott $9.81\frac{m}{s^2}$ értékhez. Tudjuk, hogy a gravitációs vektor megközelítőleg a föld közepe felé mutat, tehát „függőleges”. Ugyanakkor a gyorsulás vektor éppen ellentétes a gravitációs vektorral, hiszen nyugalomban az ellenerő éppen ellentétesen mutat. Tehát az **a** vektorunk az *ég felé mutat*, azaz **$g = -a$** !

Mi a helyzet a mágneses vektorral? Ez utóbbi egyrészt nem a valós észak felé mutat (ezt most elhanyagoljuk), másrészt nem is vízszintes. Tekintsük a bezárt szögüket:

```

1 a=a/norm(a)
2 h=h/norm(h)
3 angle=acosd(dot(a,h))
4
5 % angle = 153.14
    
```

Itt normalizáltuk a két vektor hosszát, majd egyszerű skaláris szorzattal kiszámítottuk a bezárt szögüket fokban. Mivel az **a** vektorunk függőlegesen az ég felé mutat, azt mondhatjuk, hogy a mágneses vektorunk meredeken a föld felé mutat, ez azonban nem meglepetés a 47-dik szélességi fok környékén. A fontos az, hogy a két vektor *nem merőleges* egymásra!

Az orientáció meghatározásához készítsünk először egy merőleges bázisrendszert a vektorainkból! Legyen ez a bázisunk az $\mathbf{x}, \mathbf{y}, \mathbf{z}$ bázisrendszer. Kérdés, hogy az orientációt melyik vektorra alapozzuk elsősorban? Mivel a függőleges irányt sokkal pontosabban mutatja a gyorsulási vektor, mint az északi irányt a mágneses vektor (hiszen az nem csak északra, hanem lefelé is mutat), használjuk a gyorsulási vektort. Legyen $\mathbf{z} = -\mathbf{a}$, tehát a gravitációs vektor iránya. Válasszuk az \mathbf{x} irányt merőlegesnek jobbkéz szabály szerint a gravitációs vektor és mágneses vektor által megadott síkra, azaz $\mathbf{x} = \frac{\mathbf{z} \times \mathbf{h}}{\|\mathbf{z} \times \mathbf{h}\|}$ (itt a vektort egy lépésben normalizáltuk). Az \mathbf{y} irányt pedig természetesen válasszuk merőlegesnek mindkét vektorra, azaz $\mathbf{y} = \mathbf{x} \times \mathbf{z}$ (nincs szükség normalizálásra, hiszen mindkét vektor egység hosszúságú és merőlegesek egymásra)! Ugyanezek a lépések MATLAB-ban:

```

1  z=-a
2  x=cross(z,h)
3  x=x/norm(x)
4  y=cross(x,z)
    
```

Az orientáció meghatározásához már csak a forgatási mátrixot kell felírnunk, amely természetesen a két koordinátarendszer bázisvektorainak a viszonyát írja le. Vegyük észre, hogy a \mathbf{z} vektorunk a gravitáció vektora a *telefon koordinátarendszerében*! Ugyanígy, az \mathbf{y} vektor a mágneses vektor merőleges komponense a gravitációs vektorra a *telefon koordinátarendszerében*. Ebből tehát az orientációs mátrix:

$$R_{\text{earth to phone}} = [\mathbf{x}^T \quad \mathbf{y}^T \quad -\mathbf{z}^T]$$

Az utolsó oszlopot meg kell szoroznunk mínusz eggyel, hiszen a gravitációs vektor a globális koordinátarendszerben pont ellentétesen mutat a Z tengely irányával. A kapott orientációs mátrix az a transzformáció, amely a *globális koordinátarendszer pontjait transzformálja a telefon koordinátarendszerébe*.

```

1  a=[0.28,-0.108,9.936]
2  h=[18.75,10.31,-44.43]
3
4  alength=norm(a)
5  hlength=norm(h)
6
7  a=a/norm(a)
8  h=h/norm(h)
9
10 angle=acosd(dot(a,h))
11
12 z=-a
13 x=cross(z,h)
14 x=x/norm(x)
15 y=cross(x,z)
16
17 R=[x' y' -z']
    
```

Példa Tegyük fel, hogy egy klasszikus, kétdimenziós távolságmérésen alapuló helymeghatározási rendszerünk létezik. A rendszerben 5 darab fix helyen található horgonypont (ún. anchor) található, melyektől egy mozgó eszköz valamilyen technológia segítségével megméri a távolságát (minden mérés esetében 5 távolságadat). Határozza meg a felhasználó helyzetét Newton-Gauss iteráció, Kalman-szűrő és RANSAC algoritmus segítségével!

A feladathoz három Octave file tartozik, melyekben a kiindulási adatok találhatóak:

anchors.mat A horgonypontok koordinátáit tartalmazó mátrix

measurements A horgonypontoktól való távolságmérések adatai (távolság és a távolságméréshez tartozó szórás), a mérések 100 ms időközönként történtek

real_loc.mat A felhasználó valós helyzetét tartalmazó adathalmaz

A feladathoz kiindulási pontként használhatjuk a mellékelt fájlokat, melyek bizonyos részfeladatokban segítenek, illetve egy kiindulási csonkot tartalmaz a **main.m** fájlban, melynek kiegészítésével a feladatot könnyebben elvégezhetjük.

Megoldás Első lépésként töltsük be az adatokat, majd ábrázoljuk azokat.

```
1 clear all;
2 close all;
3
4 % Mérések betöltése
5 % d(i,j) - i-dik időpontban az eszköz távolsága a j-dik
   anchortól
6 load measurements;
7
8 % A mérőeszközök helyzetének betöltése
9 % anchors(k,1) - az k-dik anchor X koordinátája
10 % anchors(k,2) - az k-dik anchor Y koordinátája
11 load anchors;
12
13 % Az eszköz valós helyzete a hiba vizsgálatához
14 % real_loc(i,1) - az i-dik időpontban az eszköz X koordinátá
   ja
15 % real_loc(i,2) - az i-dik időpontban az eszköz Y koordinátá
   ja
16 load real_loc;
```


Az környezet törlése után a `load` paranccsal betöltjük a méréseket, és a horgonypontok koordinátáit. A horgonypontok egy `anchors` mátrixban találhatók, soronként egy horgonypont, oszlopokban pedig a koordinátáik találhatók. A mérések egy `d` mátrixban találhatók, itt minden sor egy-egy időpillanat, az oszlopok pedig az adott időpontban történt távolságmérések. Az első oszlop az első horgonyponttól történt mérés, és így tovább.

Rajzoljuk ki az egyes horgonypontokat, majd a felhasználó bejárt útvonalát is. A `circle` függvény segítségével pedig rajzoljuk ki az egyes időpillanatokban a horgonypontokhoz tartozó méréseket!

```

1 for step=1:size(d,1) % Iteráció az időpillanatokra
2     clf;
3
4     % Horgonypontok kirajzolása
5     main_plot=scatter(anchors(:,1),anchors(:,2),'r');
6     axis([-12 22 -12 12]);
7     axis equal;
8     hold on;
9
10    % Valós helyzet ábrázolása
11    plot(real_loc(1:step,1),real_loc(1:step,2),'rx-',',
        LineWidth',2);
12
13    % Távolságok kirajzolása körökként
14    for i=1:size(d,2)
15        circle(anchors(i,1),anchors(i,2),d(step,i),'b');
16    end;
17
18    %
19    % Az algoritmusokat ide implementáljuk
20    %
21
22    pause;
23 end;
```

Láthatóan egy `for` ciklusban implementáltuk a kirajzolást, amely időben engedi futtatni az algoritmust, ha lefuttatjuk, akkor a `pause` függvény megállítja a futtatást egy gomb megnyomásáig. Így egy billentyű megnyomásával a következő időpillanatra léphetünk, és áttekinthetjük, miként alakul a felhasználó valós helyzete, és a helyhez tartozó távolságmérések.

Mérési modell Adott időpillanatban a mérések egy távolságvektort alkotnak, azaz $\mathbf{d} = [d_1 \ d_2 \ d_3 \ d_4 \ d_5]$, ahol

$$d_k = \sqrt{(x - x_k)^2 + (y - y_k)^2}$$

Az egyenletben a felhasználó helyzete $\mathbf{l}(x, y)$, a horgonypontok koordinátái pedig $\mathbf{p}_k(x_k, y_k)$. A mérési modell tehát

$$\mathbf{d} = \begin{bmatrix} d_1 \\ \vdots \\ d_5 \end{bmatrix} = h(\mathbf{l}) = \begin{bmatrix} \|\mathbf{p}_1 - \mathbf{l}\| \\ \vdots \\ \|\mathbf{p}_5 - \mathbf{l}\| \end{bmatrix}$$

Jól látható, hogy ez egy túldefiniált, nem-lineáris egyenletrendszer, mely zárt alakban nem megoldható.

Newton-Gauss iteráció Kezdjük a megoldást a Newton-Gauss implementációval, amely lehetővé teszi, hogy a nem-lineáris egyenletrendszert iteratív módszerrel megoldjuk. A Newton-Gauss iteráció iterációs lépése a következő:

$$\mathbf{l}_{n+1} = \mathbf{l}_n - (J^T J)^{-1} J^T (f(\mathbf{l}_n) - \mathbf{d})$$

, ahol \mathbf{l}_n a felhasználó helye az n -dik iterációban, f a mérési modell, J az f függvény Jacobi-mátrixsa, \mathbf{d} a mérési vektor. A Jacobi-mátrix a parciális deriváltakból álló mátrix, azaz:

$$J = \begin{bmatrix} \frac{\partial d_1}{\partial x} & \frac{\partial d_1}{\partial y} \\ \vdots & \vdots \\ \frac{\partial d_5}{\partial x} & \frac{\partial d_5}{\partial y} \end{bmatrix}$$

, ahol a deriváltak

$$\frac{\partial d_k}{\partial x} = \frac{x - x_k}{\sqrt{(x - x_k)^2 + (y - y_k)^2}}$$

$$\frac{\partial d_k}{\partial y} = \frac{y - y_k}{\sqrt{(x - x_k)^2 + (y - y_k)^2}}$$

Ebből a Newton-Gauss algoritmus megvalósítása meglehetősen egyszerű MATLAB-ban. Defináljuk a fájl legelején (a fő iteráción kívül) a Newton-Gauss iteráció eredményeképpen létrejött helyek eltárolására alkalmas mátrixot:

```
1  nghist=[];
```

Ezután készítsük el a Newton-Gauss iterációt. A példában 10 iterációt alkalmazunk, valós körülmények között természetesen valamilyen leállási feltételre van szükség. Az \mathbf{l}_n iterációs állapotvektort az `ngloc` változó tárolja. Az egyes eredményeket végül az `nghist` változóba konkatenáljuk, így a Newton-Gauss algoritmus által visszaadott hisztorikus útvonalat ki tudjuk rajzolni (zöld színt használva).

```
1  % Newton-Gauss algoritmus, 10 iterációval
```

```
2  ngloc=[0;0];
```

```

3 for iter=1:10
4     J=[]; % Jacobi mártix
5
6     % A Jacobi mártix kitöltés soronként (horgonypontonként)
7     for ai=1:size(anchors,1)
8         J(ai,1)=(ngloc(1)-anchors(ai,1))/sqrt((ngloc(1)-anchors(
9             ai,1))^2+(ngloc(2)-anchors(ai,2))^2);
10        J(ai,2)=(ngloc(2)-anchors(ai,2))/sqrt((ngloc(1)-anchors(
11            ai,1))^2+(ngloc(2)-anchors(ai,2))^2);
12    end
13
14    % Newton-Gauss iterációs lépés
15    ngloc=ngloc-inv(J'*J)*J'*eps;
16 end
17
18 % Historikus útvonal gyűjtése és kirajzolása
19 nghist=[nghist; ngloc'];
20 plot(nghist(:,1),nghist(:,2),'gx-','LineWidth',2);
    
```

Kalman-szűrő Mivel a mérési modellünk nem lineáris, ezért kiterjesztett Kalman-szűrőt kell alkalmaznunk. Legyen a Markov modellünk a következő:

$$\begin{aligned}
 \mathbf{l}_{i+1} &= \mathbf{l}_i + \mathbf{v}_i \Delta t + \frac{1}{2} \mathbf{a}_i \Delta t^2 \\
 \mathbf{v}_{i+1} &= \mathbf{v}_i + \mathbf{a}_i \Delta t \\
 \mathbf{d}_i &= h(\mathbf{l}_i) + \mathbf{r}
 \end{aligned}$$

, ahol $\mathbf{l}_i = (l_x^i, l_y^i)$ a felhasználó helye az i -dik időpillanatban, $\mathbf{v}_i = (v_x^i, v_y^i)$ és $\mathbf{a}_i = (a_x^i, a_y^i)$ rendre a felhasználó sebessége és gyorsulása az i -dik időpillanatban, \mathbf{d}_i a távolságmérések az i -dik időpillanatban, h a mérési modell (lásd. Newton-Gauss). Ebből az állapotvektorunk

$$\mathbf{x}_i = \begin{bmatrix} \mathbf{l}_i \\ \mathbf{v}_i \end{bmatrix} \quad (1)$$

Feltételezve, hogy a gyorsulás a zaj komponenshez járul hozzá, a teljes dinamikus modellünk a következő:

$$\mathbf{x}_{i+1} = F\mathbf{x}_i + G\mathbf{a}_i \quad (2)$$

ahol

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$$G = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \quad (4)$$

Az $\mathbf{r} \sim N(0, R)$ zaj normális eloszlású, és az R kovariancia mátrix diagonális, és ismert (`dvar` változóban található értékek). A $a_x, a_y \sim N(0, \sigma_a^2)$ zaj ismert, fix kovariancia mátrixszal rendelkezik. Ebből a teljes zaj eloszlása $G\mathbf{a}_i \sim N(0, G\sigma_a^2 G^T)$. A dinamikus modell szerint tehát az eszköz newtoni mozgást végez, és a gyorsulása ismeretlen. A kiterjesztett Kalman-szűrő szerint \mathbf{l} eloszlása is normális, méghozzá $\mathbf{l} \sim N(\mathbf{m}, P)$.

A kiterjesztett Kalman-szűrő megoldási egyenlete tehát:

$$\begin{aligned} m_i^- &= m_{i-1} \\ P_i^- &= P_{i-1} + Q_{i-1} \\ v_i &= d_i - h(m_i^-) \\ S_i &= H_x(m_i^-)P_i^-H_x^T(m_i^-) + R_i \\ K_i &= P_i^-H_x^T(m_i^-)S_i^{-1} \\ m_i &= m_i^- + K_iv_i \\ P_i &= P_i^- - K_iS_iK_i^T \end{aligned} \quad (5)$$

, ahol F_x az f Jacobi-mátrixsza (lásd. Newton-Gauss). Ezek alapján a Kalman-szűrő könnyedén implementálható MATLAB-ban. Kezdetben definiáljuk az alapvető állapotvektorokat és zaj kovarianciákat a fájl elejére:

```

1 % Kalman filter állapotvektor
2 kfhist=[];
3 m=[];
4 P=eye(4);
5
6 sd=10;      % a gyorsulás szórása
7 dt=0.1;    % az időalap
8 A=[1 0 dt 0;
9     0 1 0 dt;
10    0 0 1 0;
11    0 0 0 1;];
12 G=[.5*dt*dt 0;
13     0 .5*dt*dt;
14     dt 0
15     0 dt];
16 Q=G*G'*sd*sd;
```

Itt a dinamikus modell zajának 1 métert adtunk meg, így az eszköz elmozdulását 1 méter szórásra állítottuk, tehát a prediktált pozíciója az előző hely nagyjából 3 méteres körzetében lesz 99% valószínűséggel.

```

1 % Kalman-szűrő
2 if isempty(m)
3     m=ngloc; % A kezdeti hely a Newton-Gauss iteráció alapján
4 end;
5
6 % Predikciós lépés
7 m=A*m;
8 P=A*P*A'+Q;
9
10 H=[];
11 for ai=1:size(anchors,1)
12     H(ai,1)=(m(1)-anchors(ai,1))/sqrt((m(1)-anchors(ai,1))^2+(
13         m(2)-anchors(ai,2))^2);
14     H(ai,2)=(m(2)-anchors(ai,2))/sqrt((m(1)-anchors(ai,1))^2+(
15         m(2)-anchors(ai,2))^2);
16     H(ai,3)=0;
17     H(ai,4)=0;
18
19     h(ai,1)=sqrt((m(1)-anchors(ai,1))^2+(m(2)-anchors(ai,2))
20         ^2);
21 end
22 R=diag(dvar(step,:));
23
24 % Frissítési lépés
25 v=d(step,:)'-h;
26 S=H*P*H'+R;
27 K=P*H'*inv(S);
28 m=m+K*v;
29 P=P-K*S*K';
30
31 % Historikus utvonal gyűjtése és kirajzolása
32 kfhist=[kfhist; m'];
33 plot(kfhist(:,1),kfhist(:,2),'bx-','LineWidth',2);
34 circle(kfhist(end,1),kfhist(end,2),P(1,1),'b'); % A hiba mé
35 rtékének jelzése

```

A következő fontos megjegyzéseket tesszük. A Kalman-szűrő kezdeti állapotának a Newton-Gauss iteráció eredményét vesszük (csak az első pozíció esetében). Az R kovariancia mátrixot figyeljük meg, hogy a bemeneti adatok alapján töltjük ki, méghozzá diagonál mátrixra, tehát az egyes távolságmérések között nem feltételezünk korrelációt. Az algoritmus végén külön sorban kirajzoljuk a hely szórását is körként, méghozzá a kovariancia mátrix első eleme segítségével (valóságban ez ellipszis is lehetne, vagy akár a kovariancia mátrixban lehetnek nem diagonális elemek is, ezeket most elhanyagoljuk).

RANSAC A RANSAC algoritmusok az ún. robusztus algoritmusok közé tartoznak, melyek lehetővé teszik a felállított sztochasztikus modellünktől eltérő hibák feltárását, és a modellnek nem megfelelő mérések, ún. outlier-

ek kiszűrését. A feladatok során csupán a legegyszerűbb, fix iterációs számmal dolgozó (tehát nem adaptív) RANSAC algoritmust implementáljuk.

A megvalósítás során minden iterációban kiválasztunk két mérést, hiszen két mérés esetén két kört kapunk, amelyeknek kettő, egy, vagy nulla metszéspontja lehet. Első esetben véletlenszerűen választunk a két metszéspont közül, az utolsó esetben a két kör középpontját összekötő egyenesen vesszük fel a metszéspontot. A metszéspont ismeretében a méréseket aszerint osztályozzuk inlier és outlier mérésekre, hogy melyik található a pont $\varepsilon = 2$ méteres körzetében. Ennek megfelelően összeállítjuk a mérések indexeiből az inlier index halmazt. Amennyiben ez több elemet tartalmaz, mint az eddigi legtöbb elemet tartalmazó halmaz, a legjobb megoldásként kezeljük. Végül az iterációk lejátszása után a legtöbb inlier-t tartalmazó halmazban található méréseket vastagabb körként ábrázoljuk (valóságban ezeket a méréseket használjuk a további becslések során, pl. a Kalman-szűrőben, vagy a Newton-Gauss iterációban).

A RANSAC megvalósítása MATLAB környezetben ezek alapján nyilvánvaló:

```

1 % RANSAC
2 error_threshold=2; % Az inlier mérések maximális hibája, mé
    terben
3 best_inlier_idx=[]; % Az iterációk során eddig megtalált
    legnagyobb inlier halmaz
4 for iter=1:30
5     % Minimális mintahalmaz kiválasztása (2 pont a körök metsz
        éséhez)
6     anchor_idx1=randi(size(anchors,1));
7     anchor_idx2=randi(size(anchors,1));
8
9     % Figyeljünk arra, hogy ne legyen a két index azonos
10    while anchor_idx1 == anchor_idx2
11        anchor_idx2=randi(size(anchors,1));
12    end
13
14    % A két kiválasztott horgonyponthoz tartozó távolsági görb
        ék metszete (2 db, ebből egy kiválasztása véletlenszerű
        en)
15    test_point=circle_intersect(anchors(anchor_idx1,1),anchors
        (anchor_idx1,2),d(step,anchor_idx1),anchors(anchor_idx2
        ,1),anchors(anchor_idx2,2),d(step,anchor_idx2));
16    test_point=test_point(randi(2),:);
17
18    % A próbamegoldáshoz tartozó inlierek összegyűjtése
19    inlier_idx=[];
20    for ai=1:size(anchors,1)
21        distance=sqrt((anchors(ai,1)-test_point(1))^2+(anchors(
        ai,2)-test_point(2))^2);
22        if abs(distance-d(step,ai)) < error_threshold
23            inlier_idx=[inlier_idx ai];
24        end

```

```
25 end
26
27 % Ha a talált megoldás jobb, akkor mentsük le
28 if length(inlier_idx) > length(best_inlier_idx)
29     best_inlier_idx = inlier_idx;
30 end
31 end
32
33 % Az inlier távolságok ábrázolása vastagabb körként
34 for i=best_inlier_idx
35     circle(anchors(i,1),anchors(i,2),d(step,i),'LineWidth',2);
36 end
```

Figyeljük meg, hogy a körök metszéspontját a `circle_intersect` függvénnyel végezzük, amely a két kör középpontját és a két kör sugarát várja az argumentumban. A függvényt a `circle_intersect.m` fájlban találjuk, és mellékeljük a gyakorlat anyagához.

3. Kérdések

Az ellenőrző kérdések során a két csoportból 1-1 szabadon választott kérdésre kell a választ elküldeni a 3-3 pluszpontért.

3.1. Helymeghatározás alapjai

- Milyen viszonyítási rendszereket ismerünk a helymeghatározása témakörében?
- Írja le egy ϕ szöggel történő, tetszőleges $u = [u_x, u_y, u_z]$ tengely körüli forgatást jelképező kvaterniót!
- Írja le, miként számítjuk ki egy Rodrigues-vektorhoz tartozó forgatási mátrixot!

3.2. Helymeghatározási feladat megoldása

- Ha x jelöli a mérési vektorunkat, f a modellünket és θ a paraméterhalmazt, írja fel a legkisebb négyzetes eltérés költségfüggvényét!
- Írja fel a Newton-Gauss nemlineáris egyenletek megoldási módszerének frissítési egyenletét!
- Írja fel a Bayes tételt! Jelölje a prior, posterior és hipotézis fogalmakat!
- Írja fel a kiterjesztett Kalman-szűrő állapotegyenleteit!
- Írja fel a RANSAC valószínűségi megállási feltételét!