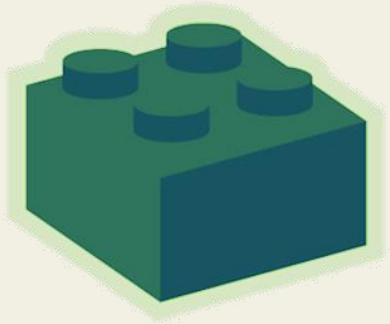
C# és V-REP alapismeretek

Horváth Ernő, Dr. Pozna Claudiu Radu



Elérhetőségek

Dr. Pozna Claudiu Radu
pozna@sze.hu

Horváth Ernő
http://www.sze.hu/~herno/
herno@sze.hu

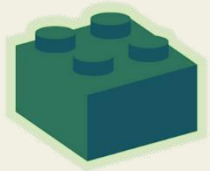
Tanszéki honlap
http://it.sze.hu

RS1.SZE.HU/~HERNO/

Követelmények

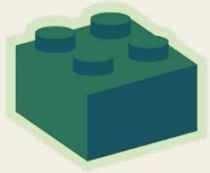
- Féléves jegy (szorgalmi időszakban szerezhető)
- ZH nincs
- Katalógus nincs (de az órán való aktív részvétel elengedhetetlen a jegy szerzéséhez)

XAML



```
<Window x:Class="Ev3.Elso.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="példa" Height="350" Width="525" Loaded="Window_Loaded">
    <Grid>
        <StackPanel HorizontalAlignment="Left" Width="200">
            <Button Click="EloreClick">Előre</Button>
            <Button Click="HatraClick">Hátra</Button>
            <Button Click="BalClick">Balra</Button>
            <Button Click="JobbClick">Jobbra</Button>
            <Button Click="EmelClick">Emelés</Button>
            <Button Click="LerakClick">Lerakás</Button>
            <TextBlock x:Name="txtUltra" Text="Ultrahang"
FontSize="30"></TextBlock>
        </StackPanel>
    </Grid>
</Window>
```

Data binding

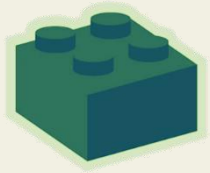


- Az adatkötés megértéséhez vegyük a következő példát:
  xaml: `<TextBlock x:Name="txtData" Text="Valami" />`
  cs: `txtData.Text = "Valami más";`

- Ezzel az egyik gond, hogy amennyiben a **`TextBlock`** szöveg részét a programunkban több helyen is változtatni szeretnék, a kódunk mindannyiszor ismételni kell. A másik pedig, hogy amennyiben a **`TextBlock`** tartalmát más szálból kell változtatni, úgy a **`Dispatcher`**t kell igénybe venni vagy a **`Task`** befejeztével új taskot kell indítani, ami a WPF saját szálában fut le

- Erre megoldás, ha egy változó tartalmát összekötjük egy adott változóval, és annak változása esetén a GUI értesül erről, valamint ki is jelzi a változást.

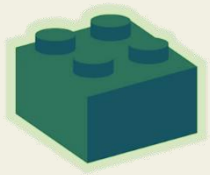
Data binding



Az adatkötés 3 fontos eleme:

- Egy publikus tulajdonság. Egy tagváltozó publikussá tétele nem elég, a WPF tulajdonságokon keresztül kommunikál.
- Az adatforrás beállítása. (pl: `DataContext = this`). A `DataContext` nem csak a dokumentum egészére,hanem bármely WPF elemre egyénileg beállítható. Pl. `GridName.DataContext = this`
- A kötés beállítása. `Text="{Binding TextValue}"`.

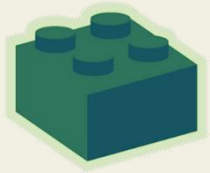
Data binding



- `using System.ComponentModel;`
- `public partial class MainWindow : Window, INotifyPropertyChanged`
- `private string _posString = "position";`

```
public string PositionString
{
  get { return _posString; }
  set
  {
     _posString = value;
     OnPropertyChanged("PositionString");
  }
}
```

Data binding

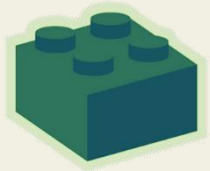


```
protected virtual void OnPropertyChanged(string property)
{
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(property));
}
public event PropertyChangedEventHandler PropertyChanged;
```

XAML:

```
DataContext="{Binding RelativeSource={RelativeSource Self}}
<TextBlock x:Name="txtData" Text="{Binding PositionString}" />
```

Programszálak



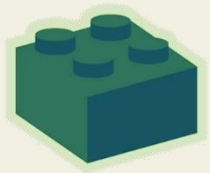
Bármely osztály tetszőleges metódusa elindítható külön szálban a következőképp:

- `using System.Threading;`
- `upThread = new Thread(new ThreadStart(update));`
  `upThread.Start();`

Leállítani például így lehet:

- `upThread.Abort();`

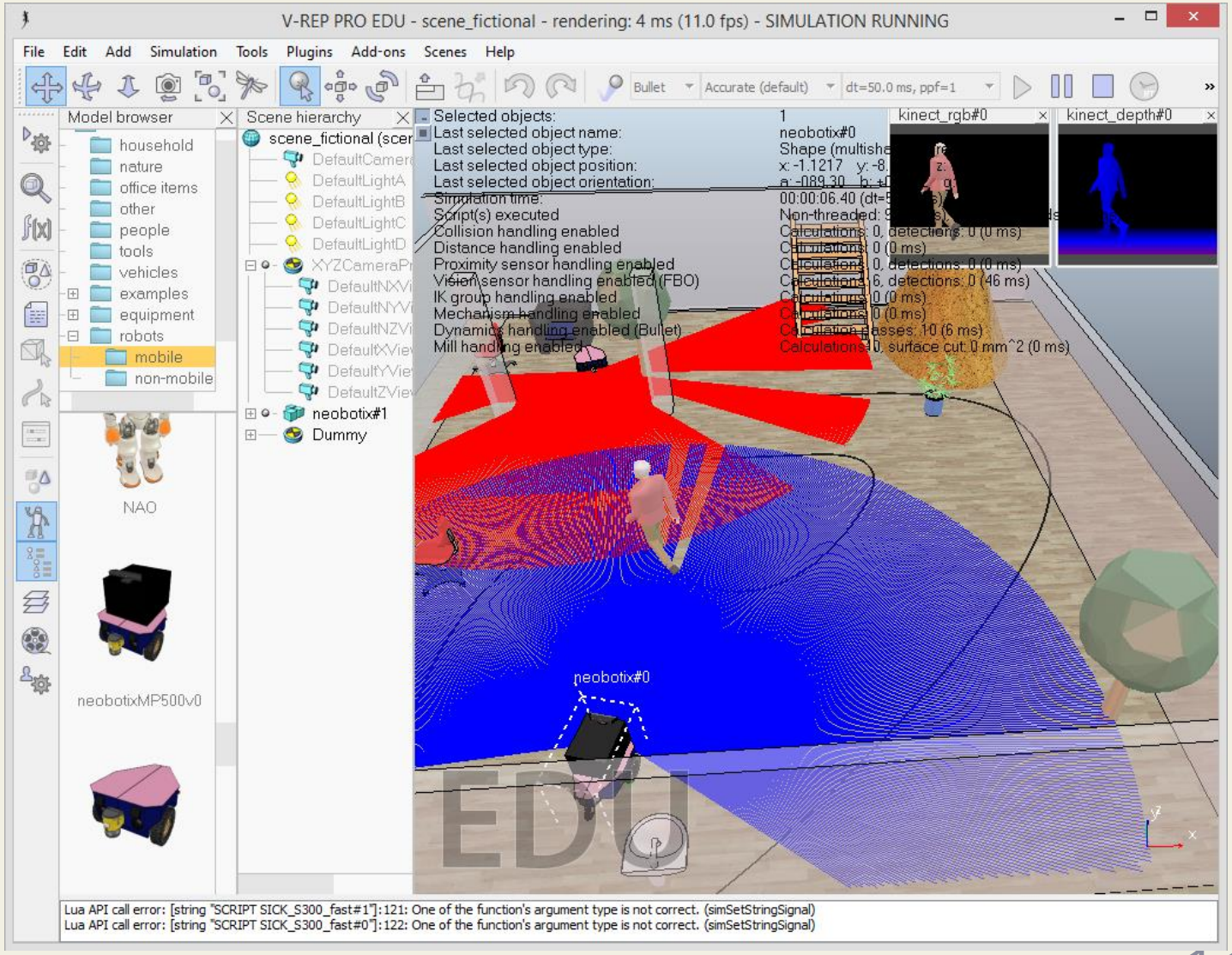
Programszálak



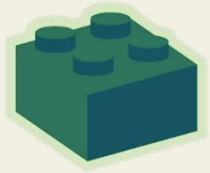
- `public Thread(ThreadStart startMethod)` - konstruktor
- `public static Thread CurrentThread {get;}` - az aktuálisan futó szál
- `public static void Sleep(int millisec)` a szál "elaltatása" adott időre
- `public string Name {get; set;}` - a szál neve
- `public ThreadPriority Priority {get; set;}` - a szál prioritása - ld. `enum ThreadPriority`
- `public ThreadState State {get; set;}` - a szál állapota ld. `enum ThreadState`
- `public bool IsAlive {get;}` - a szál fut-e még
- `public bool IsBackground {get; set;}` - Ha egy szál háttérben van, akkor miatta nem fut tovább a program. Tehát, ha minden előtérben lévő szál befejezte a futását, akkor a program leáll, és a háttérszálak futása megszakad.
- `public void Start()` - elindítja a szál indító metódusát (ld. konstruktor)
- `public void Join()` - a hívó megvárja, míg az adott szál befejezi a futását
- `public void Abort()` - a szálból való kilépésre szolgál, `ThreadAbortException`t vált ki

V-REP

- Az egyik legfunkciógazdagabb robot szimulációs környezet, amely sok technikával és technológiával kompatibilis
- Az oktatási verzió ingyenes
- A telepítő 122 MB és viszonylag gazdaságosan bánik a számítógép erőforrásaival



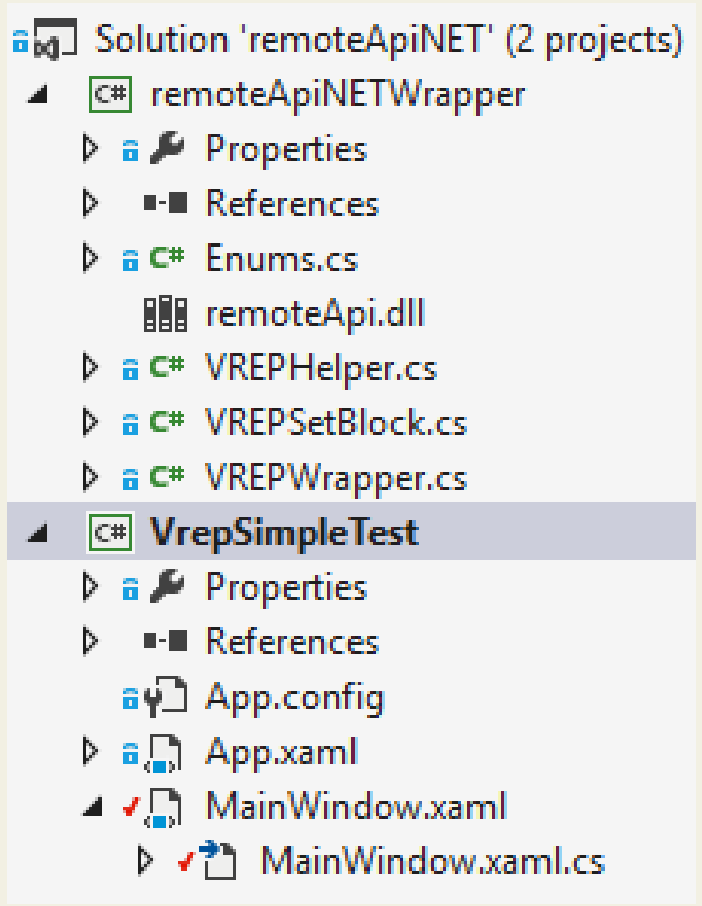
V-REP remote API



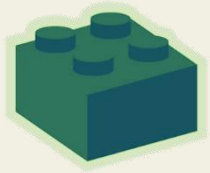
- A remote API interfészt biztosít számos nyelv írt programok számára a szimulációs környezet funkcióinak eléréséhez. Többek között C-ben használható dll-eket tudnk elérni a C# **`DllImport`**jának segítségével. Egy ilyen csomagoló (wrapper) funkcionalitású tesztprogram elérhető itt: https://github.com/horverno/sze-academic-robotics-projects/tree/master/VrepCsharpInteroperation
- API dokumentáció itt: http://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm
- V-REP help: http://www.coppeliarobotics.com/helpFiles/

V-REP remote API – pár példa

- Remote API helper functions
  - » **`simxStart`**
  - » **`simxFinish`**
- General object handle retrieval
  - » **`simxGetObjectHandle`**
- Vision sensor functionality
  - » **`simxGetVisionSensor`**
- Joint object functionality
  - » **`simxSetJointTargetVelocity`**



V-REP remote API



- C#:

```
_clientID = VREPWrapper.simxStart("127.0.0.1", 19997, true, true, 5000, 5);

VREPWrapper.simxGetObjectHandle(_clientID, "wheel_left#0", out _handleLeftMotor0,
simx_opmode.oneshot_wait);

VREPWrapper.simxSetJointTargetVelocity(_clientID, _handleLeftMotor0, 10,
simx_opmode.oneshot_wait);
```

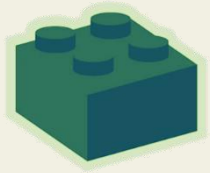
- MATLAB:

```
clientID = vrep.simxStart('127.0.0.1', 19997, true, true, 5000, 5);

[err,motorLeft] = vrep.simxGetObjectHandle(clientID, 'wheel_left#0',
vrep.simx_opmode_oneshot_wait);

vrep.simxSetJointTargetVelocity(clientID, motorLeft, 10,
vrep.simx_opmode_oneshot_wait);
```

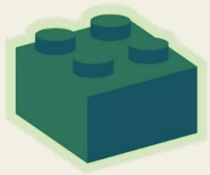
V-REP remote API



```
// A szimuláció újraindítása
private void buttonResetSim_Click(object sender, RoutedEventArgs e)
{
   VREPWrapper.simxStopSimulation(_clientID, simx_opmode.oneshot_wait);
   Thread.Sleep(400);
   VREPWrapper.simxStartSimulation(_clientID, simx_opmode.oneshot_wait);
}

// a wrapperbe új funció magvalósítása
[DllImport("remoteApi.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern simx_error simxStopSimulation(int clientID, simx_opmode opmode);
```

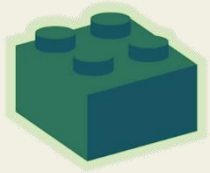
V-REP remote API opmode



A legtöbb remote API hívás kliens szerver kommunikációt jelent, az "**operation mode**" dönti el, hogy ez a kommunikáció milyen módon megy végbe, pl:

- **`simx_opmode_oneshot`**: nem blokkoló mód, a funkció nem vár a válaszra. Így a parancs elküldésekor az előző ugyanilyen parancs válasza jön meg, ha volt ilyen.
- **`simx_opmode_oneshot_wait`**: blokkoló mód, a parancs elküldésekor a függvény megvárja a választ (kivéve timeot). A válasz kikerül az inbox buffer-ből.
- **`simx_opmode_streaming + alpha`**: nem blokkoló mód, a parancs elküldésekor az előző ugyanilyen parancs visszatérési értéke lesz a válasz, amennyiben volt ilyen. A parancs a szerver oldalon folyamatosan végrehajtásra kerül. Az alpha értéke 0-65535 ms lehet és a szerver oldali delay beállítására alkalmas.
- **`simx_opmode_oneshot_split + beta`** (nem ajánlott)
- **`simx_opmode_streaming_split + beta`** (nem ajánlott)
- **`simx_opmode_discontinue`**
- **`simx_opmode_buffer`**
- **`simx_opmode_remove`**

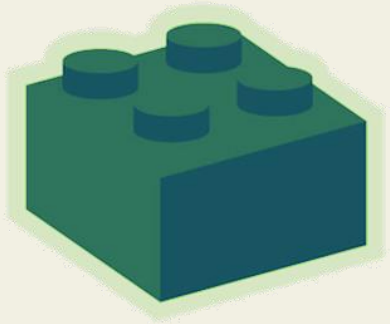
NuGet



- Tools » Library Package Manager » Package Manager Console

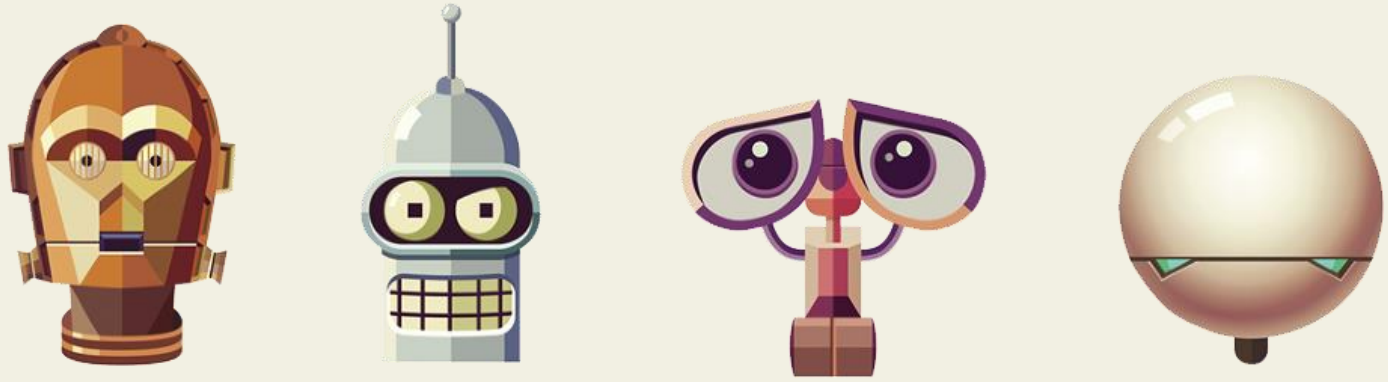
```
PM> Install-Package OxyPlot.Wpf -Pre
```

Robotika alapismeretek



Robotika

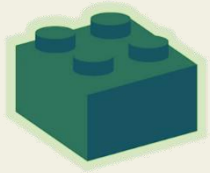
"Robotics is the branch of mechanical engineering, electrical engineering and computer science that deals with the design, construction, operation, and application of robots, as well as computer systems for their control, sensory feedback, and information processing. These technologies deal with automated machines that can take the place of humans in dangerous environments or manufacturing processes, or resemble humans in appearance, behavior, and/or cognition."
- Wikipedia



SLAM

Simultaneous localization and mapping (SLAM) alatt olyan algoritmusokat értünk, amelyek egy mozgó eszközön futnak, egy terület bejárása során térképet hoznak létre (vagy frissítenek) miközben párhuzamosan, egyidejűleg nyomon is követik az adott eszköz pozícióját.

Miért SLAM?

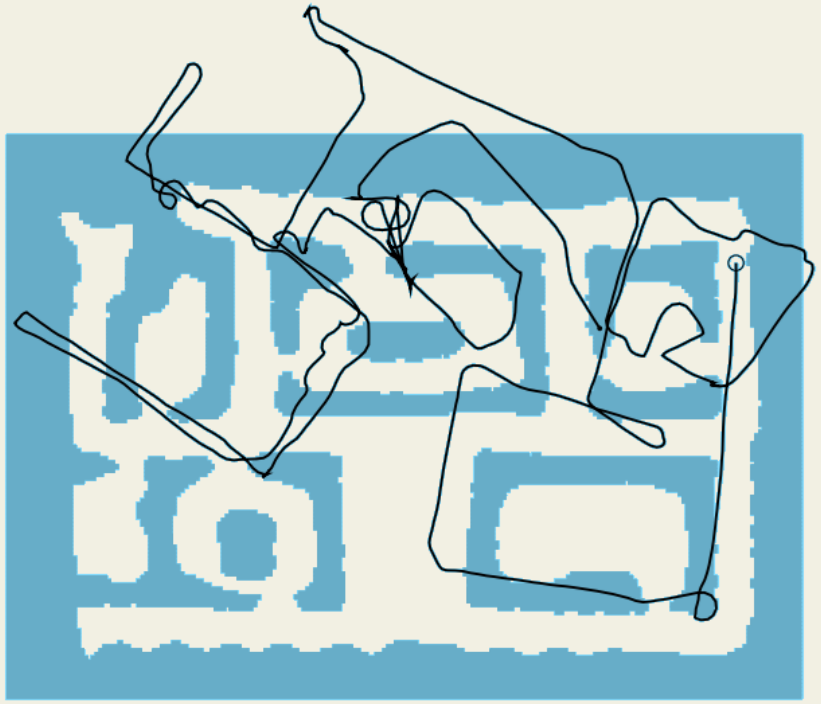


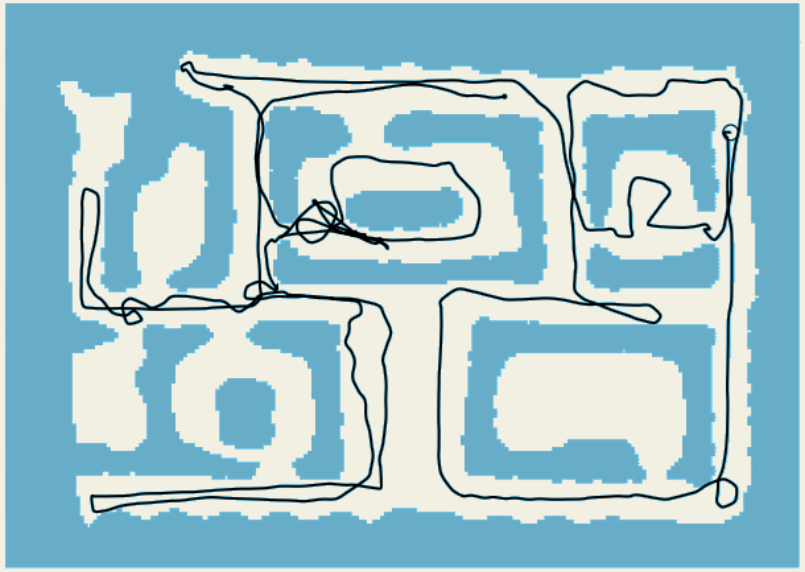
Problémák a pozícionálással / navigációval

- A robot mozgás modellje nem pontos
- A kerék csúszásából adódó hibák összeadódnak » kumulált hiba (nagyobb távolság » nagyobb hiba)
- Szenzor pontatlanságok

Szenzoros és becsült navigációs értékek

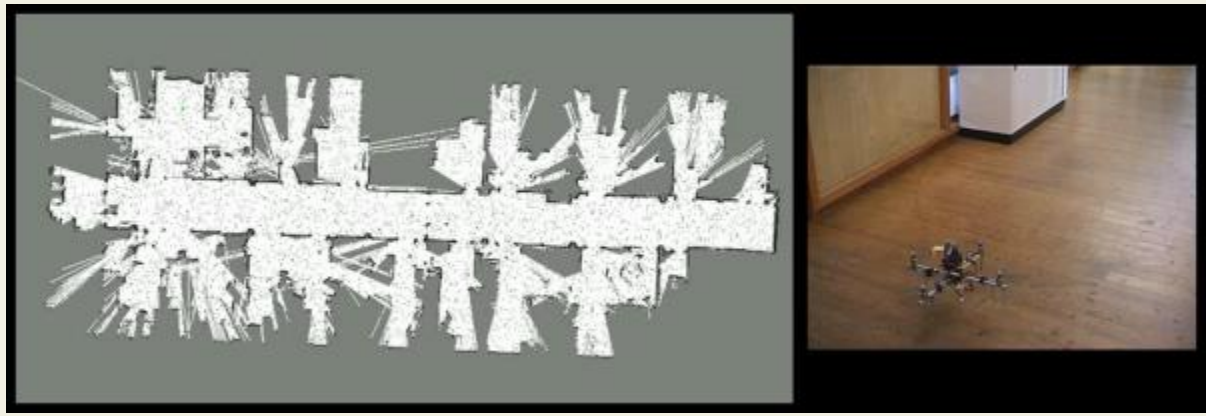
- A navigáció során számolt pozícióadatok (odometria) és a korrigált adatok





Videók

- Például:
  - » http://youtu.be/CYlb6ZcCSzU
  - » http://youtu.be/SeNLUW79
- Tutorial
  - » http://youtu.be/O5Zu19-tjY8
  - » http://youtu.be/h8gXn-E1ZFg

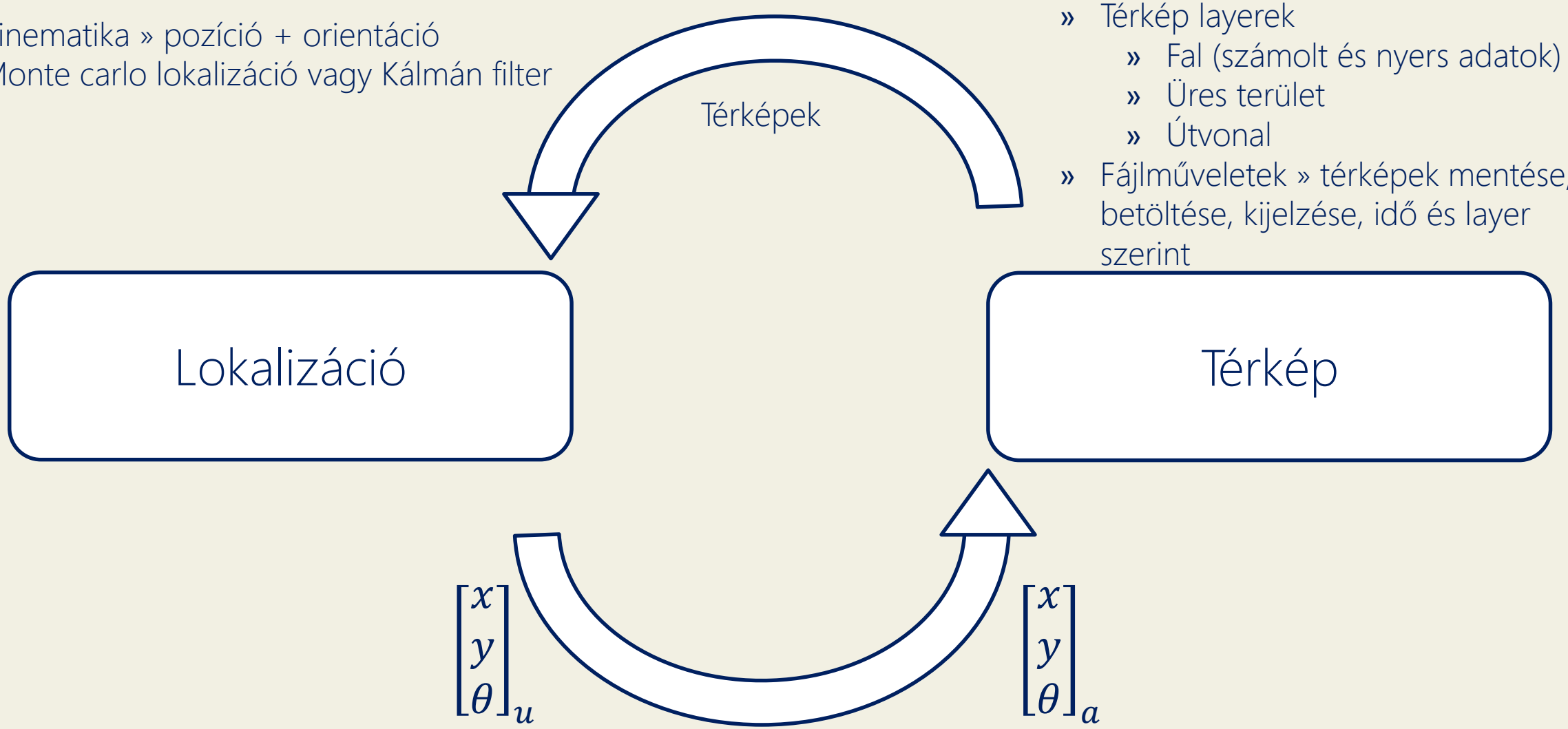


A feladat

- A robot számára ismeretlen környezet
- Térkép építése csak a
  - » kinematikai modell és a
  - » szenzor adatok felhasználásával
- Manuális irányítás (esetleg automata irányítás továbbfejlesztésiként)
- Először működő algoritmus kifejlesztése a szimulátorral
- Valós robotra (neobotix, lego) implementálás

Csapatok

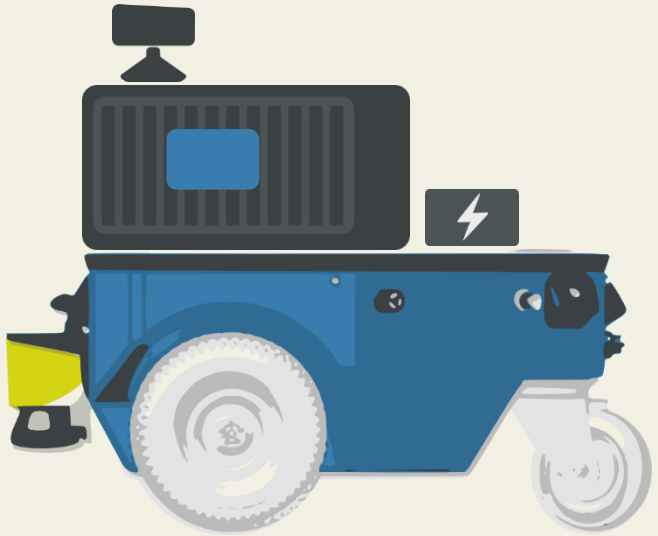
- Kinematika » pozíció + orientáció
- Monte carlo lokalizáció vagy Kálmán filter



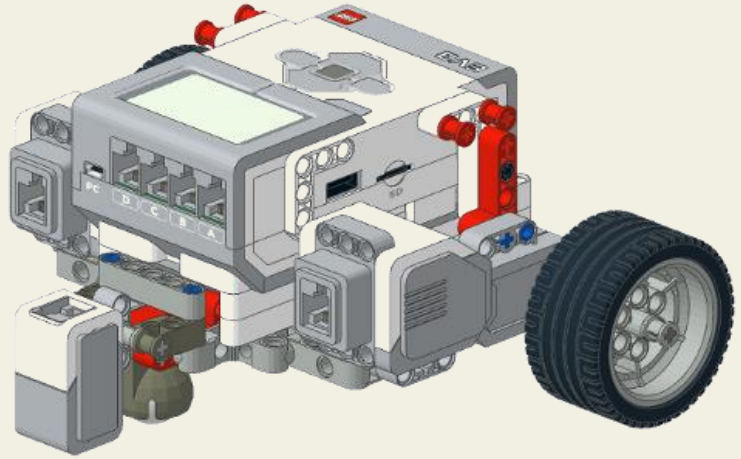
- Térkép layerek
  - Fal (számolt és nyers adatok)
  - Üres terület
  - Útvonal
- Fájlműveletek » térképek mentése, betöltése, kijelzése, idő és layer szerint

A rendelkezésre álló robotok

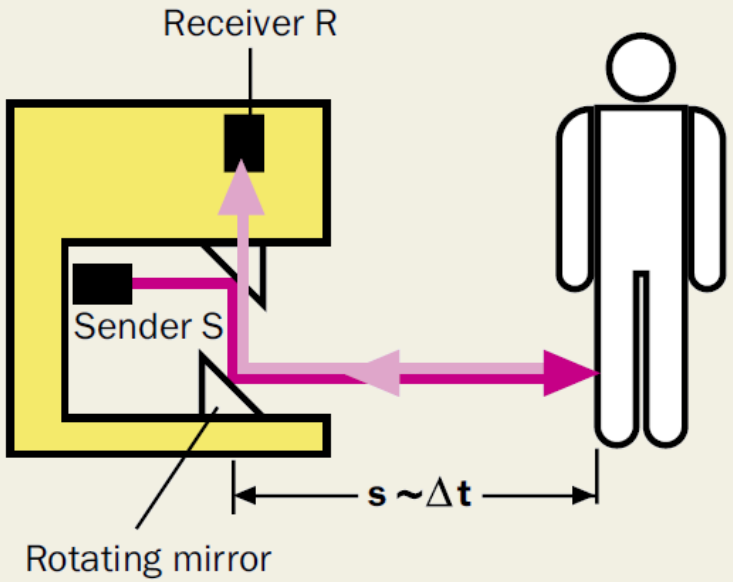
- Neobotix
  - » Laser scanner (LIDAR szenzor)
  - » Kinect szenzor
  - » Differenciális kinematikai modell
  - » Sebesség referencia irányítás

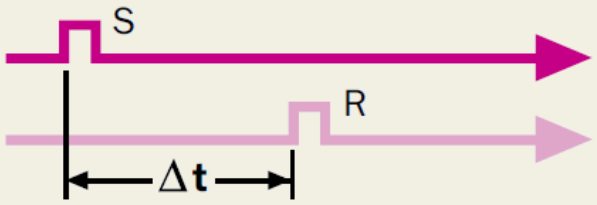


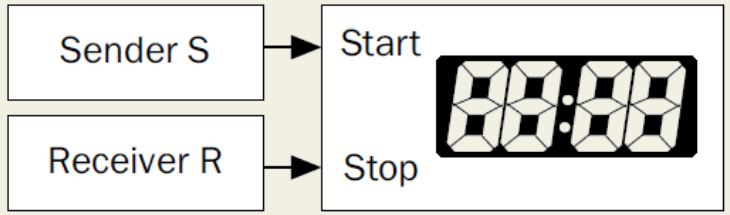
- Ev3
  - » Infravörös pásztázó szenzor
  - » Ultrahang pásztázó szenzor
  - » Differenciális kinematikai modell
  - » Sebesség referencia irányítás

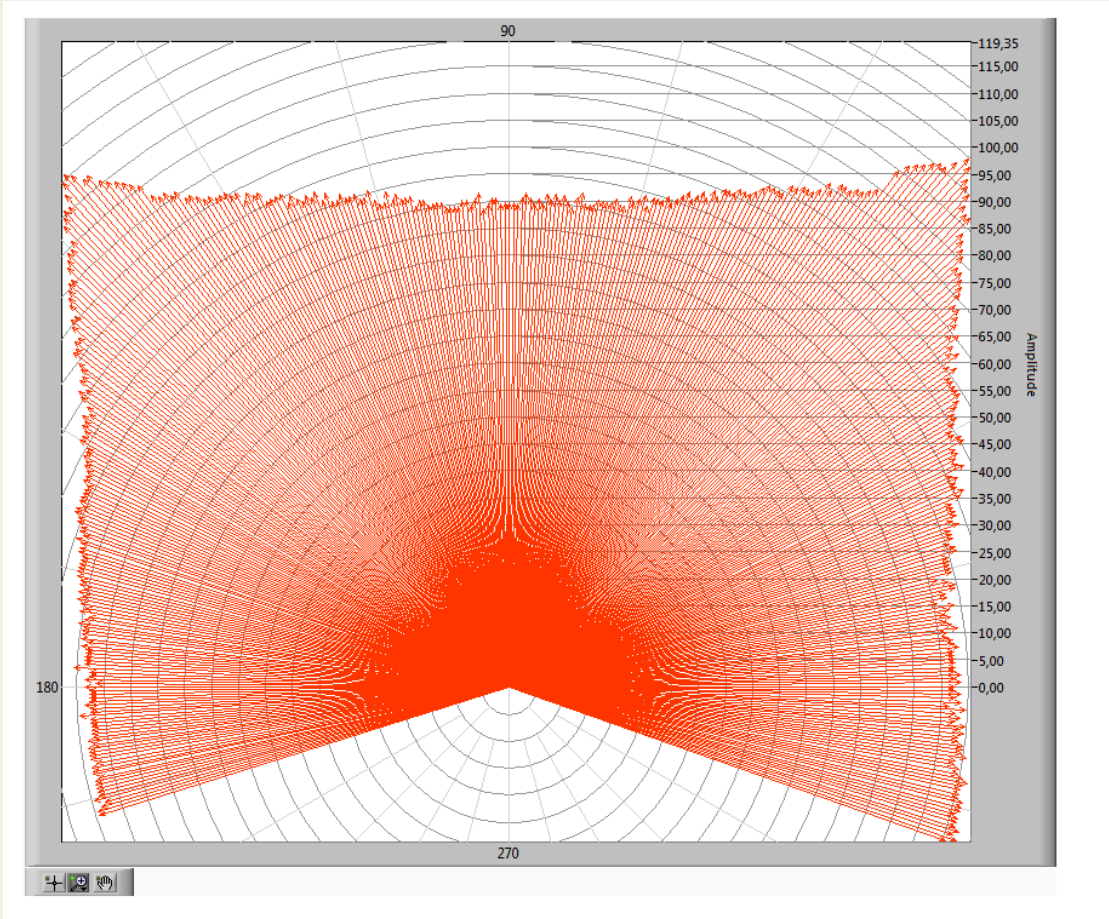


Laser scanner (LIDAR)

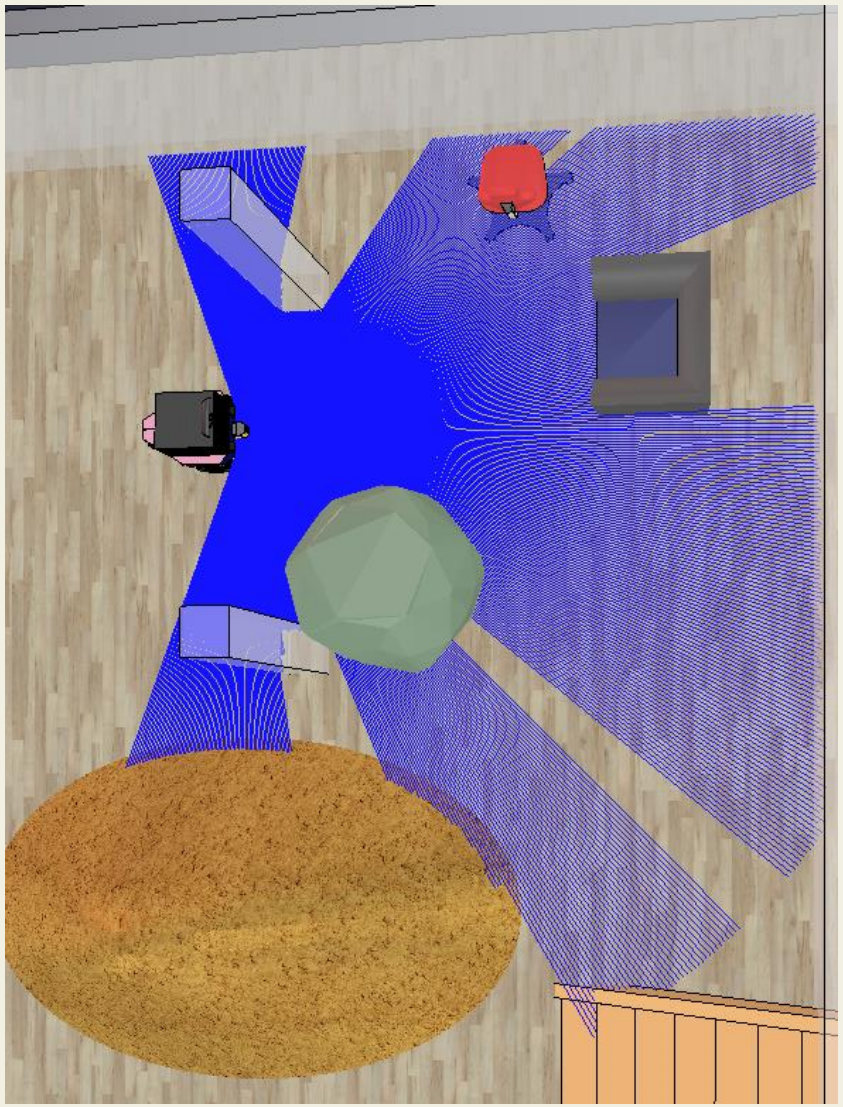


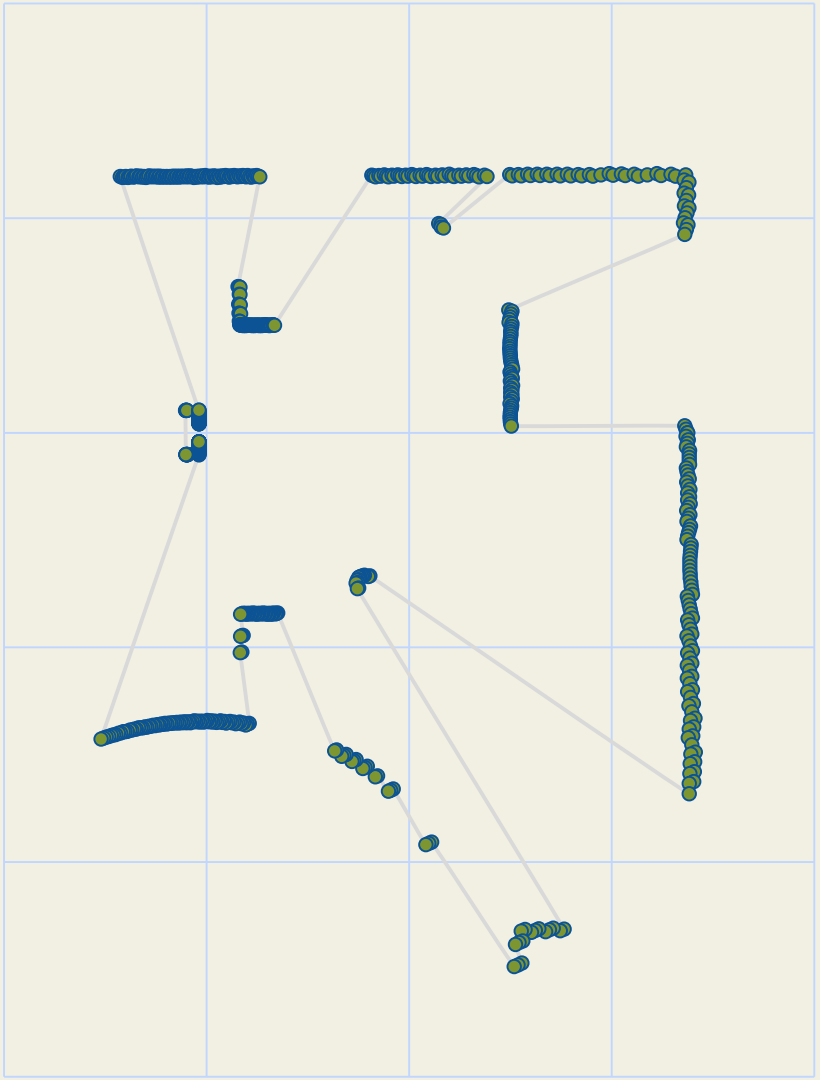


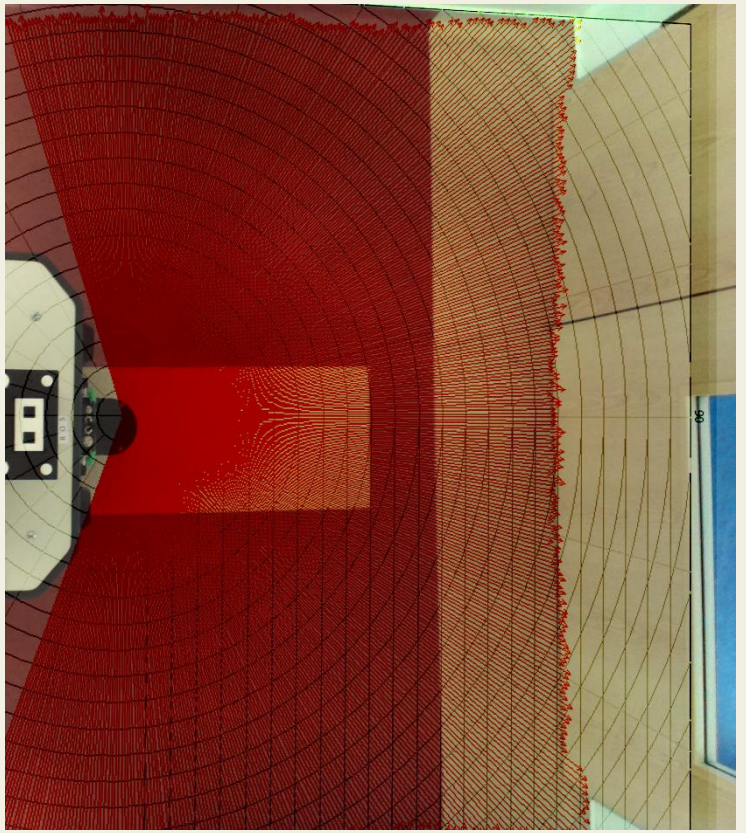




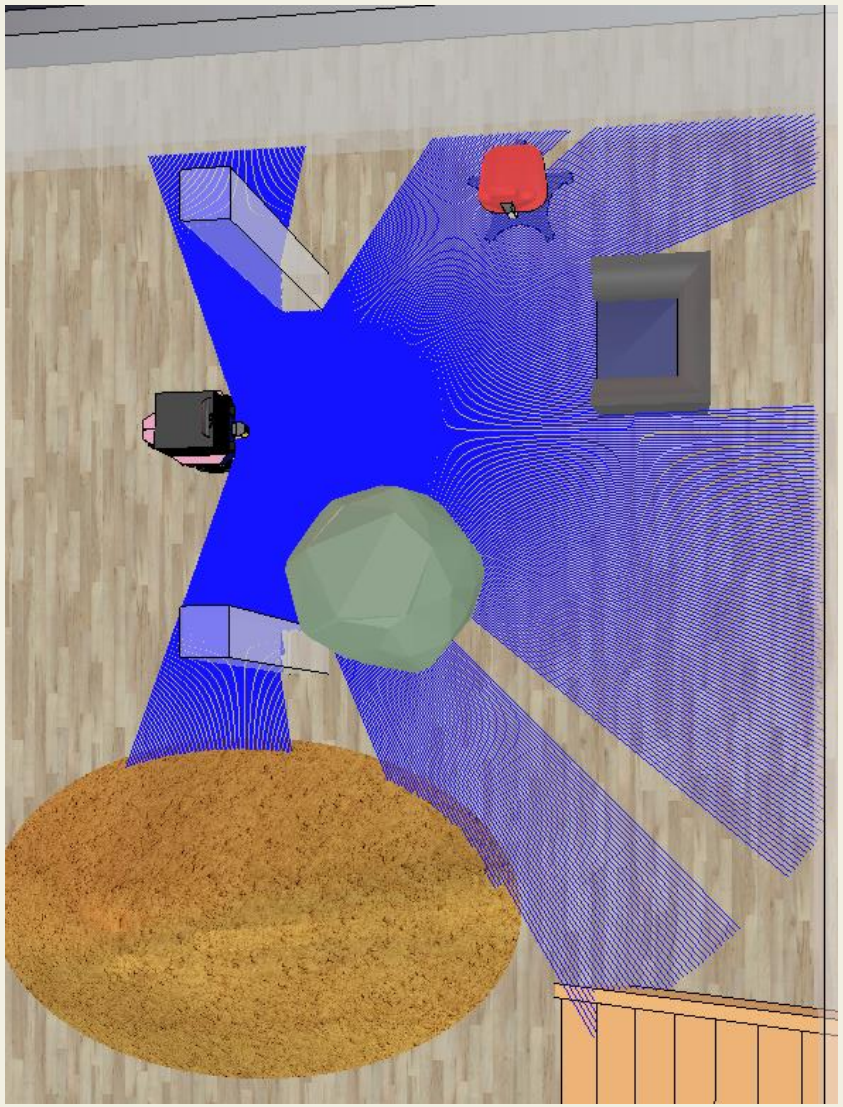
Laser scanner (LIDAR)

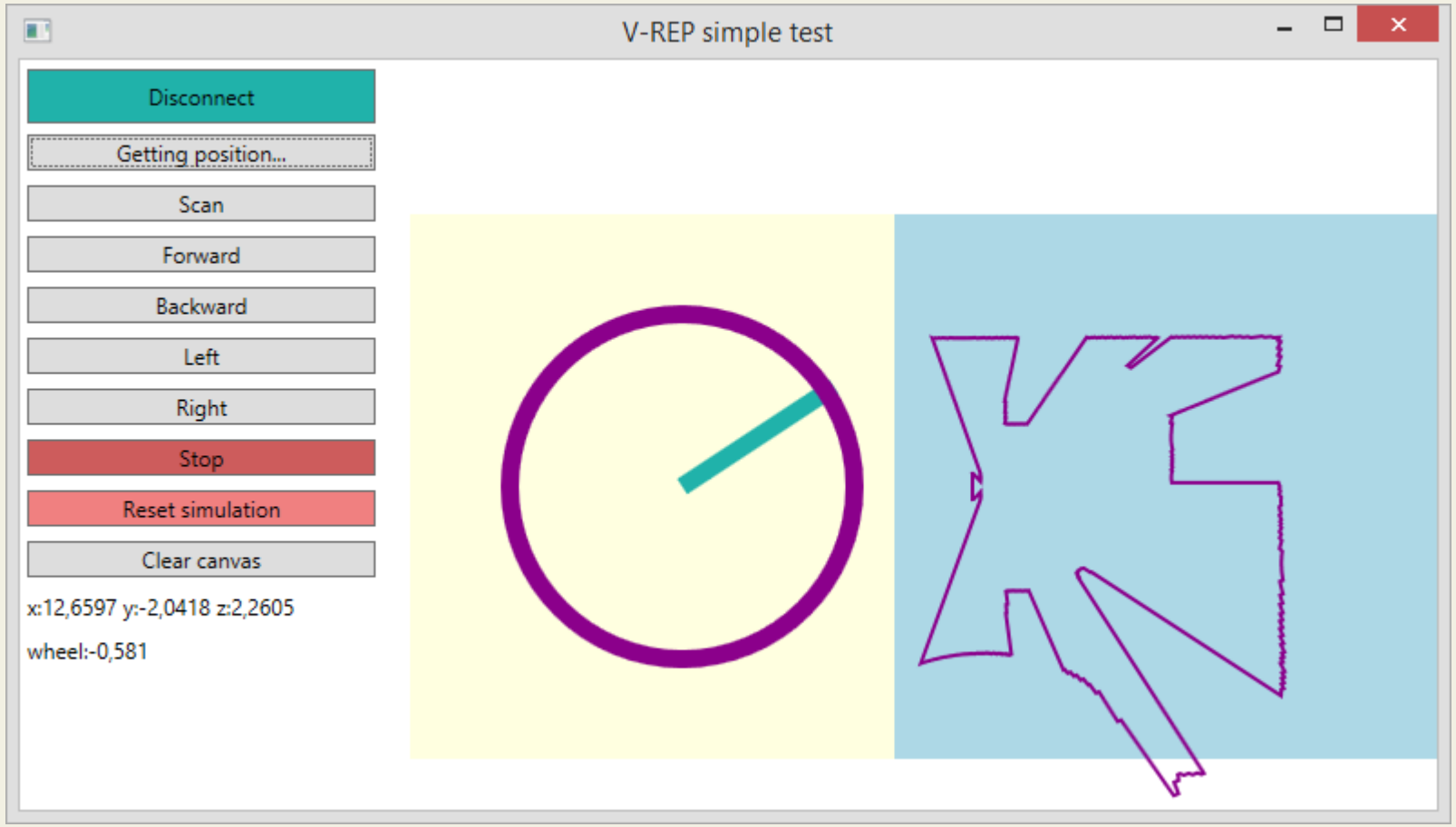






Laser scanner (LIDAR)

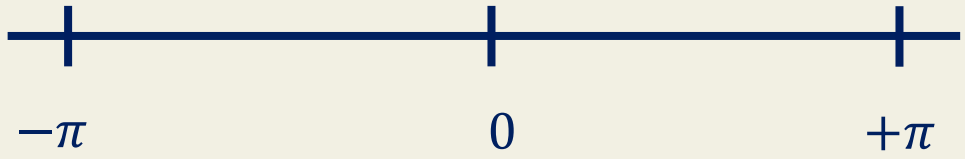


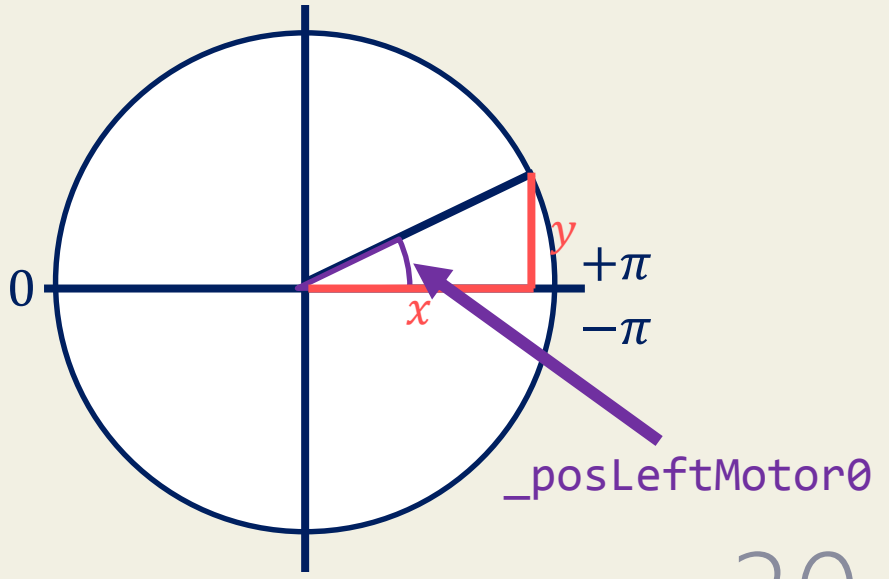


A kerék pozíciójának lekérése

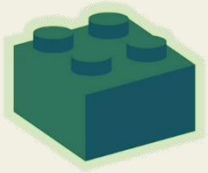
```
VREPWrapper.simxGetJointPosition(_clientID, _handleLeftMotor0,
ref _posLeftMotor0, simx_opmode.oneshot_wait);
```

Motor pozíció: [-3,14 .. +3,14]





Megtett távolság



- Elsőször is tudnunk kell a kerék kerületét (perimeter)

$$K = 2\pi r = d\pi$$

- Egyszerűség kedvéért vegyük az egyenes vonalú egyenletes haladást: Minden teljes elfordulás után K értéket hozzá kell adni az eddig megtett távolsághoz, valamint az aktuális kerékpozíció alapján a K arányos részét. (~távolság integrálása)

Ismert képletek

Gyorsulás » Sebesség » Út (pozíció) összefüggései

$$a = \frac{dv}{dt}$$

A gyorsulás a sebesség idő szerint deriváltja

$$v = \int a\,dt$$

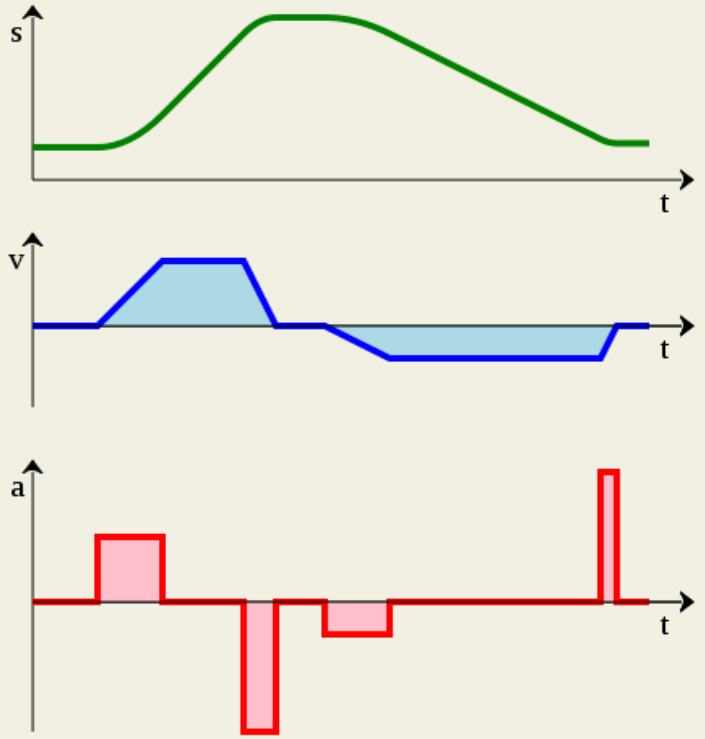
A sebesség gyorsulás idő szerinti integráltja

$$v = \frac{dx}{dt}$$

A sebesség a pozíció idő szerint deriváltja

$$x = \int v\,dt$$

A pozíció a sebesség idő szerinti integráltja



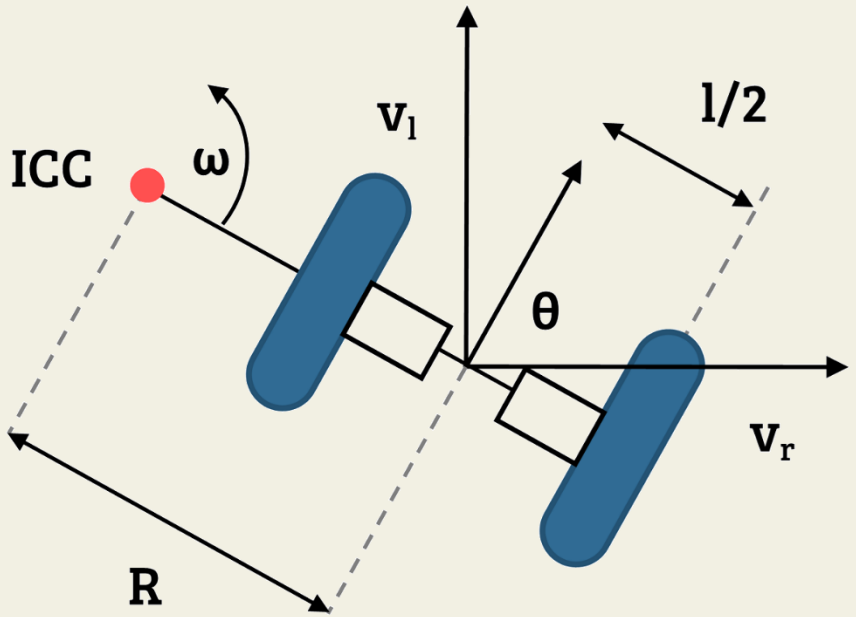
Út (pozíció)

Sebesség

Gyorsulás

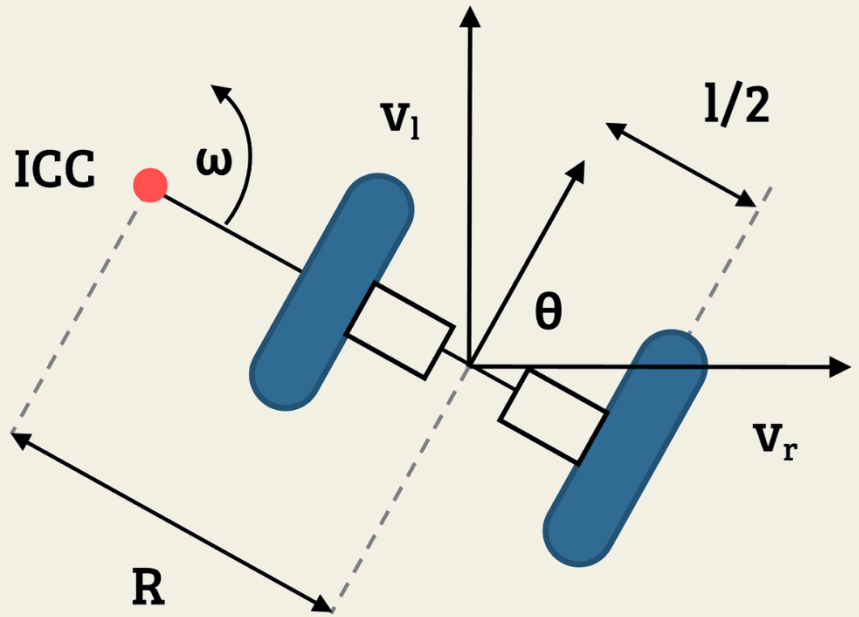
Kinematikai modell

- Differenciális, vagyis kerekenként eltérő meghajtás (differential drive) modell
- A bal kerék sebességet leíró V_l és a jobb kerék sebességet leíró V_r a következő egyenletekkel határozható meg:
- $V_l = \omega\left(R - \frac{l}{2}\right)$
- $V_r = \omega(R + \frac{l}{2})$



Kinematikai modell

- Esetünkben l jelenti a bal és a jobb kerék távolságát, R jelenti a pillanatnyi fordulási középpontot (ICC) és a kerekeket összekötő szakasz középpontja közti távolságot és ω pedig az elfordulás szögét. Ekkor az R és az ω meghatározható bármely időpontban:
- $R = \frac{l}{2} \frac{V_r + V_l}{V_r - V_l}$
- $\omega = \frac{V_r + V_l}{l}$



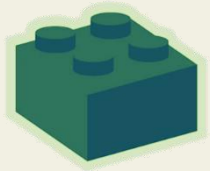
Kinematikai modell

A robot új pozíciója (x', y') és orientációja θ' a $t + \delta t$ időpillanatban, vagyis δt idő elteltével a következőképp számolható:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix}$$

A mozgásegyenletek sokat egyszerűsödnek, amennyiben csak a középpont körüli forgatást és az egyenes vonalú mozgást alkalmazzuk.

Kinematikai modell



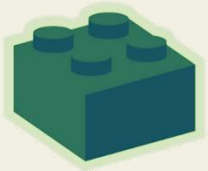
Általánosan le tudjuk írni a Θ_t irányba és $V(t)$ sebességgel mozgó robotot adott t időpillanatban és pozícióban a következő egyenletekkel.

$$x(t) = \int_0^t V(t)\cos[\theta(t)]\,dt$$

$$y(t) = \int_0^t V(t)\sin[\theta(t)]\,dt$$

$$\Theta(t) = \int_0^t \omega(t)\,dt$$

Kinematikai modell



Az egyenletek differenciális meghajtású esetére a következőképp pontosíthatóak:

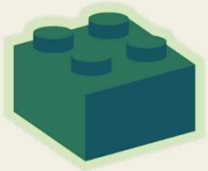
$$x(t) = \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]\cos[\theta(t)]\,dt$$

$$y(t) = \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]\sin[\theta(t)]\,dt$$

$$\Theta(t) = \frac{1}{l}\int_0^t [v_r(t) + v_l(t)]dt$$

Ebből adódik a kérdés miszerint hogyan tudjuk irányítani a robotot úgy, hogy elérjen egy adott pozíciót és irányt (x, y, θ)

Kinematikai modell



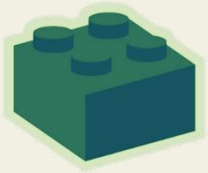
Ha mindkét kerék v sebességgel forog, tehát $v_r = v_l = v$

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + v\cos(\theta)\delta t \\ y + v\sin(\theta)\delta t \\ \theta \end{bmatrix}$$

Ha pedig a középpont körül forgatjuk a robotot az x és y koordináta nem változik, csak a θ szög, mégpedig az δt idő, az l szélesség és a v sebesség alapján

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + 2v\delta\, t/l \end{bmatrix}$$

Kinematikai modell



Pseudo kód

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + v\cos(\theta)\delta t \\ y + v\sin(\theta)\delta t \\ \theta \end{bmatrix}$$

C# kód

```
x = (int)(x + v * Math.Cos(DegreeToRadian(theta)));
y = (int)(y + v * Math.Sin(DegreeToRadian(theta)));
```

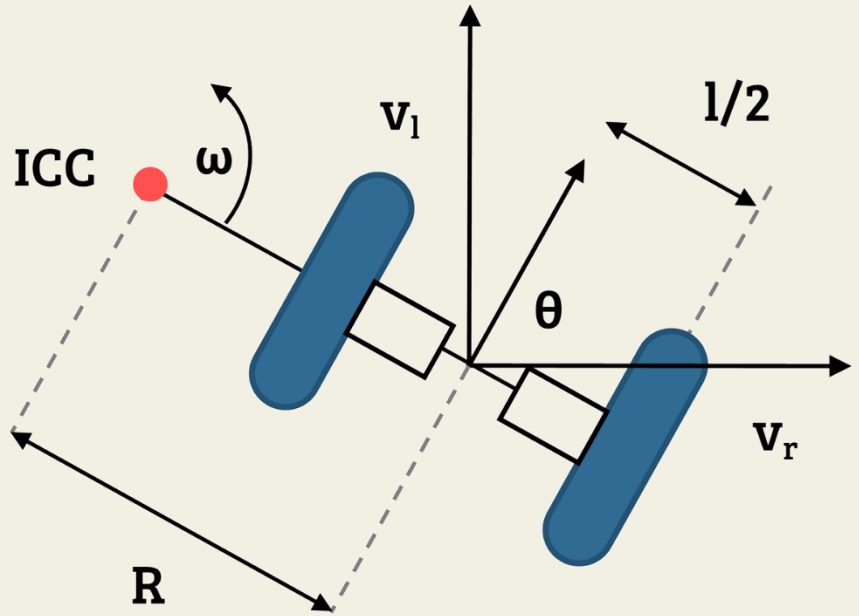
//egységnyi δt-vel számolva

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + 2v\delta\, t/l \end{bmatrix}$$

```
theta = theta + 2 * v / l
```

Milyen értékek szükségesek a modellhez?

Tulajdonság	*Érték*
A középponttól a kerék közepéig $\left(\frac{l}{2}\right)$	
Teljes szélesség: (D_{robot})	
Kerék teljes szélesség:	
Kerék futófelület szélessége:	
Kerék sugár: (r_{wheel})	
Két kerék távolsága (két kerék futófelületének a távolsága):	



Just in case! :)

Görög betűk

- δ - delta
- θ - theta
- ω - omega
- σ - sigma
- μ - mű

Mátrixszorzás

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1000 \\ 100 \\ 10 \end{bmatrix} =$$

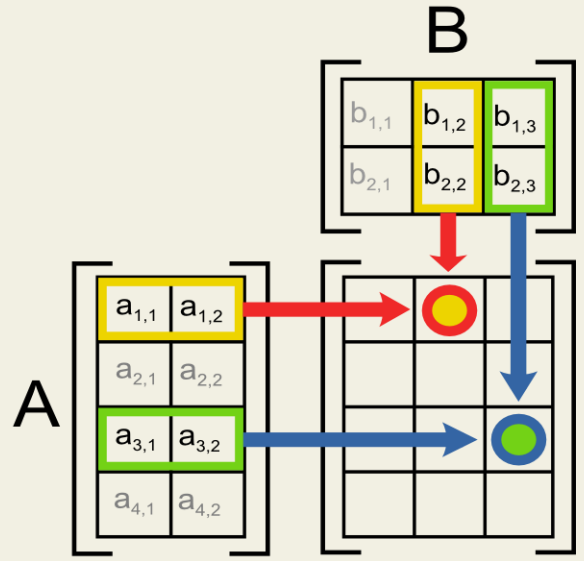
Just in case! :)

Görög betűk

- δ - delta
- θ - theta
- ω - omega
- σ - sigma
- μ - mű

Mátrixszorzás

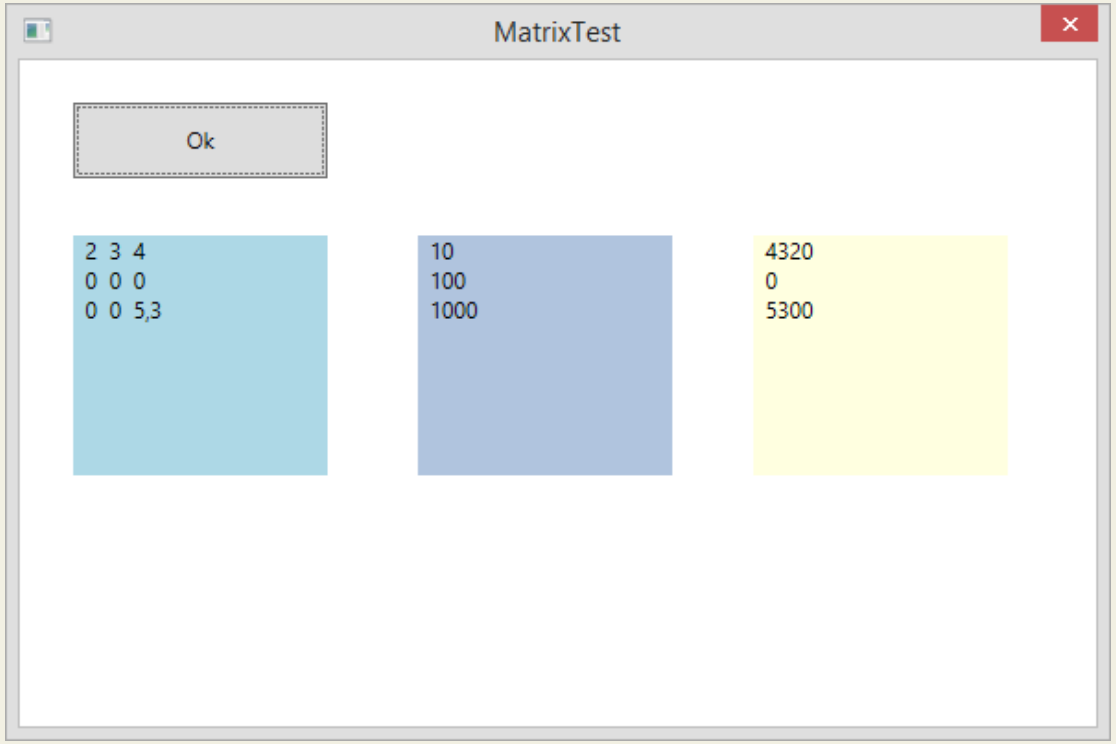
$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1000 \\ 100 \\ 10 \end{bmatrix} = \begin{bmatrix} 2340 \\ 1000 \\ 20 \end{bmatrix}$$

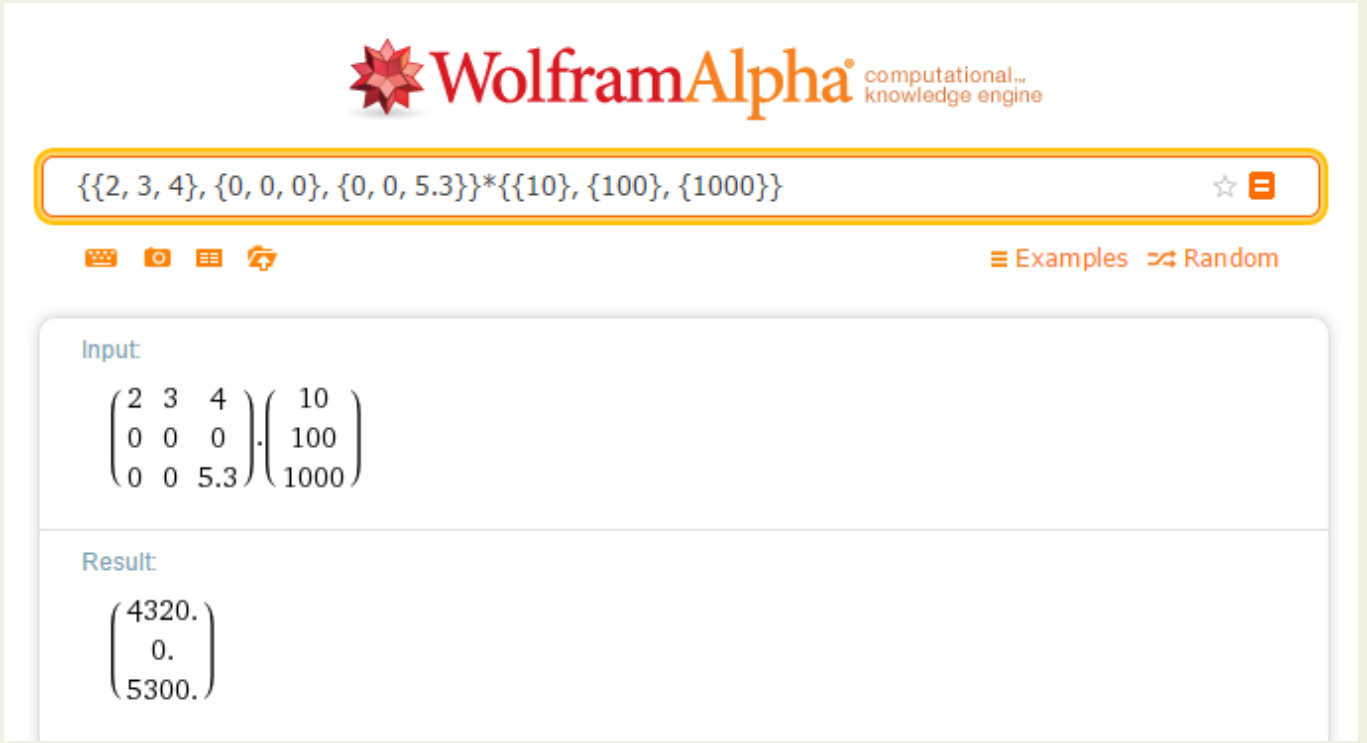


http://en.wikipedia.org/wiki/Matrix_(mathematics)

Mátrixok

Mátrixszorzás bemutatása C# tesztprogramon keresztül



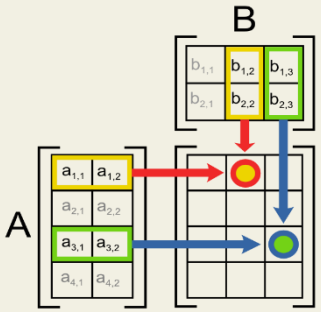


Ellenőrizzük: www.wolframalpha.com

Mátrixok

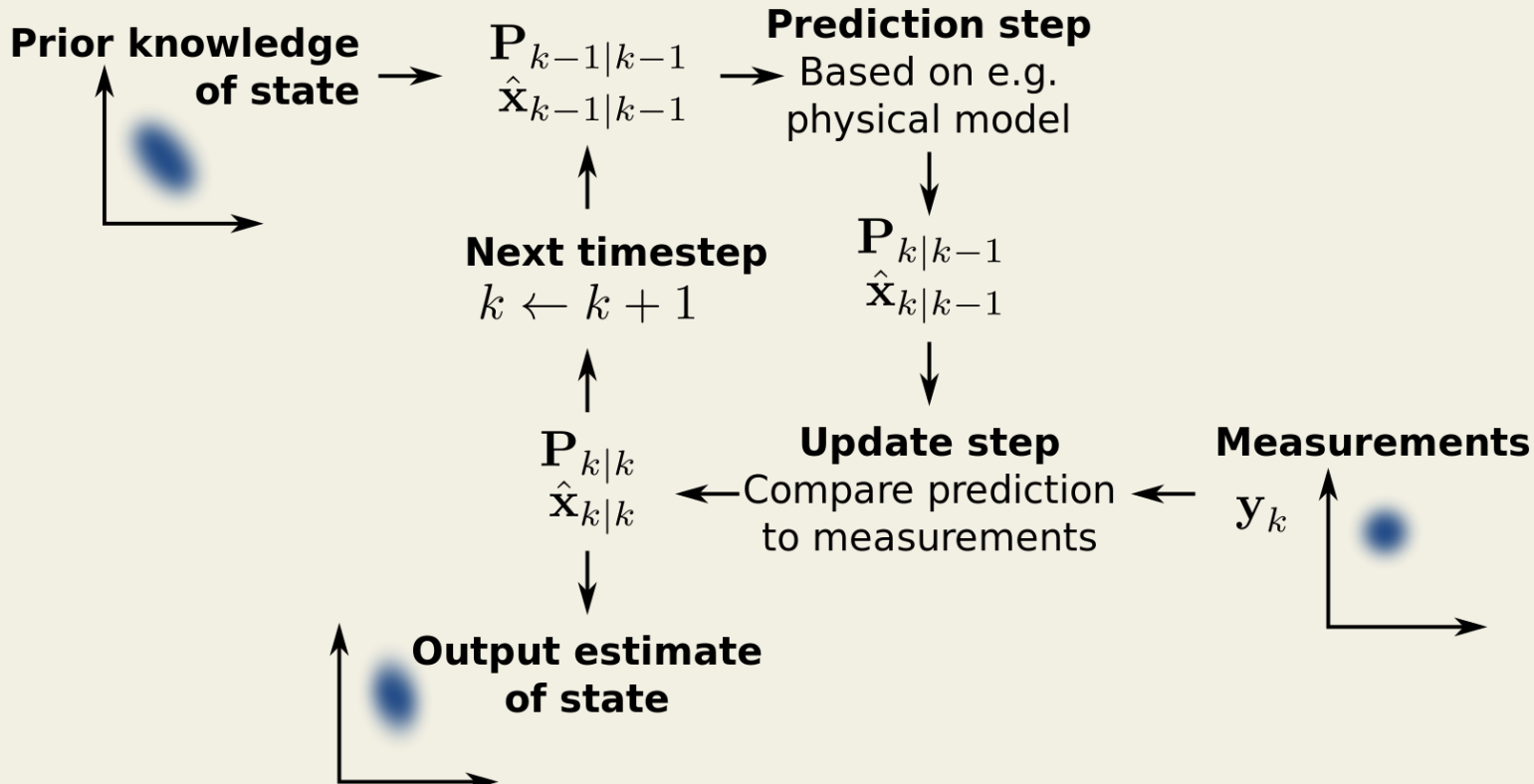
A és B mátrix szorzása, eredménye C mátrix, csak akkor lehetséges, ha A mátrix oszlopainak száma megegyezik a B mátrix sorainak számával.

```
if (a.GetLength(1) == b.GetLength(0))
{
  c = new float[a.GetLength(0), b.GetLength(1)];
  for (int i = 0; i < c.GetLength(0); i++)
   {
    for (int j = 0; j < c.GetLength(1); j++)
     {
       c[i, j] = 0;
       for (int k = 0; k < a.GetLength(1); k++) // vagy k<b.GetLength(0)
           c[i, j] = c[i, j] + a[i, k] * b[k, j];
     }
   }
}
```

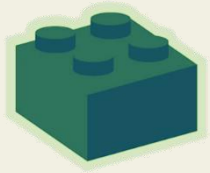


Kalman filter

- Kálmán Rudolf Emil (1930 -)
- Zajos bemenő adatok rekurzív mérésével egy pontosabb becslést ad a mérés tárgyának állapotáról.
  http://en.wikipedia.org/wiki/Kalman_filter



Bayes-tétel



Adott A és B események valószínűsége ($P(A)$ és $P(B)$), és a $P(B|A)$ feltételes valószínűség esetén fennáll a

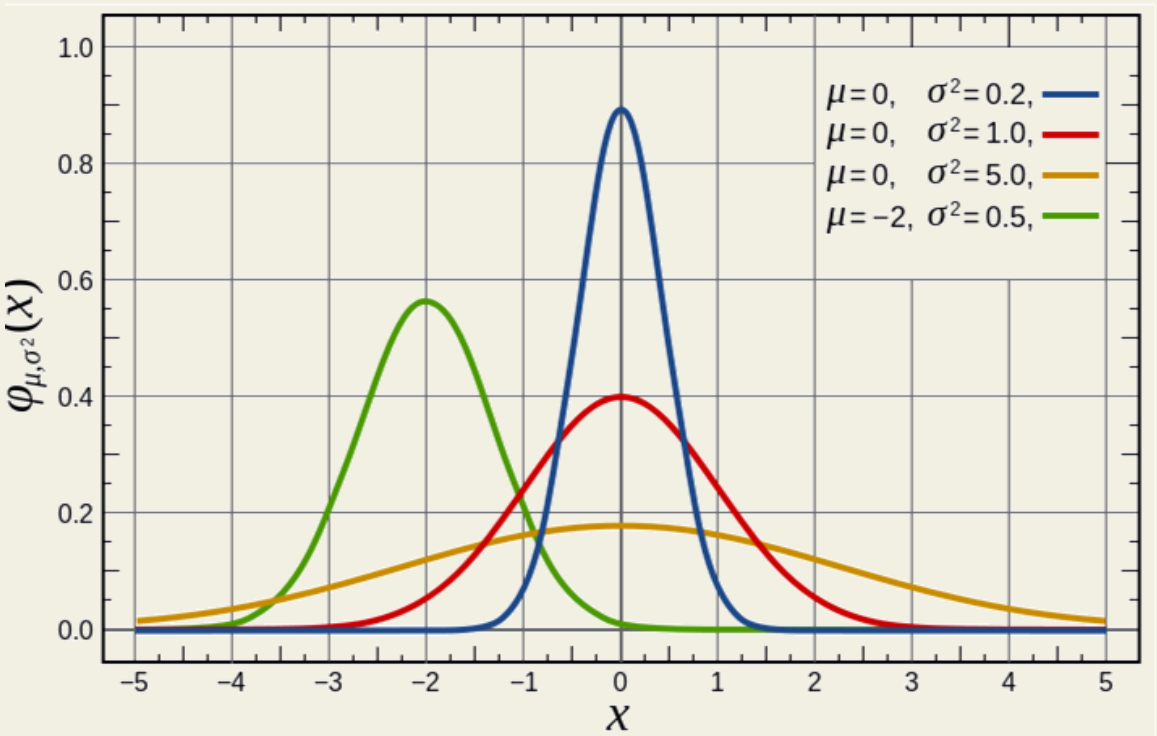
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

egyenlőség.
http://en.wikipedia.org/wiki/Bayes'_theorem

Normális eloszlás

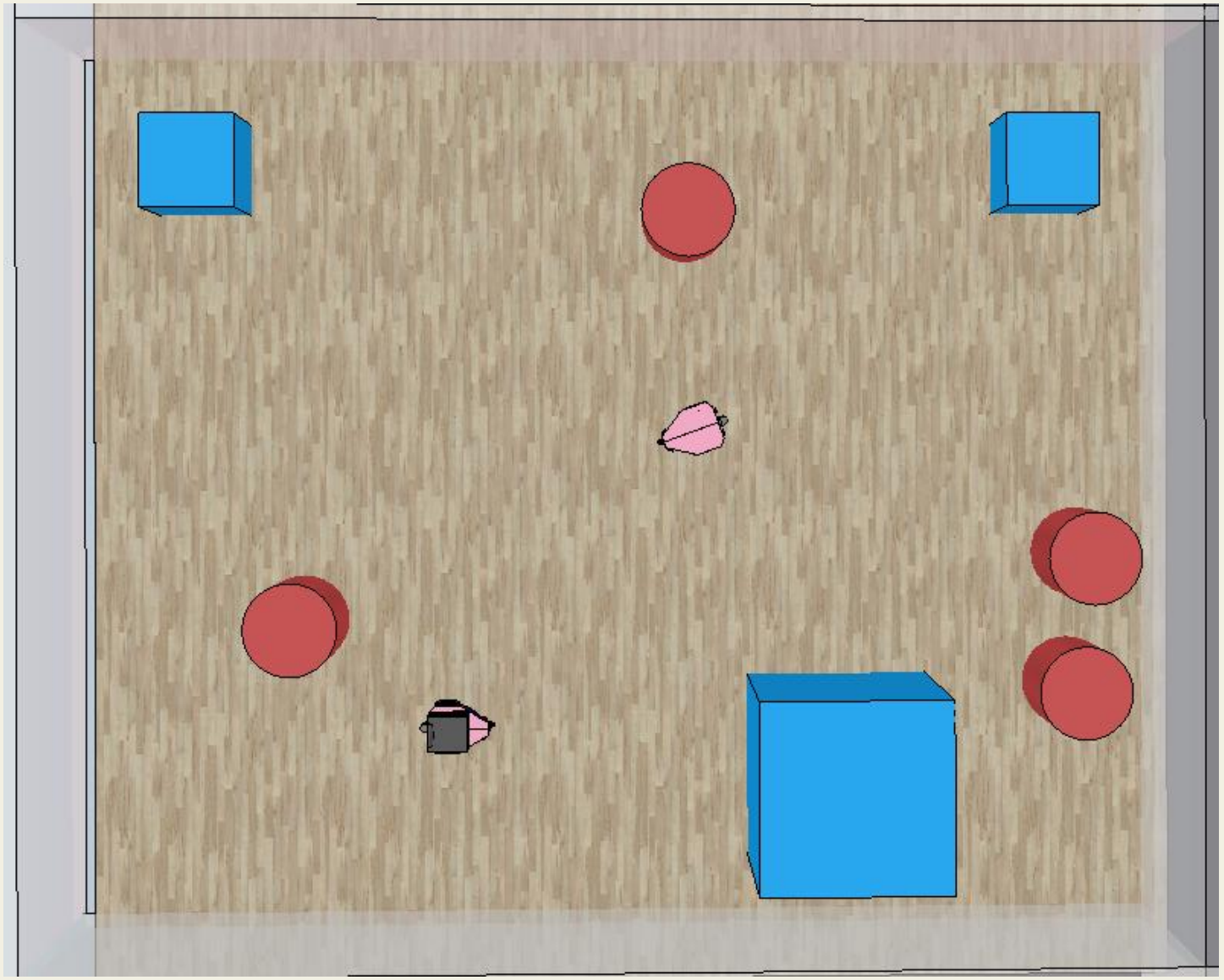
Normal (Gaussian) distribution
Sűrűségfüggvénye:

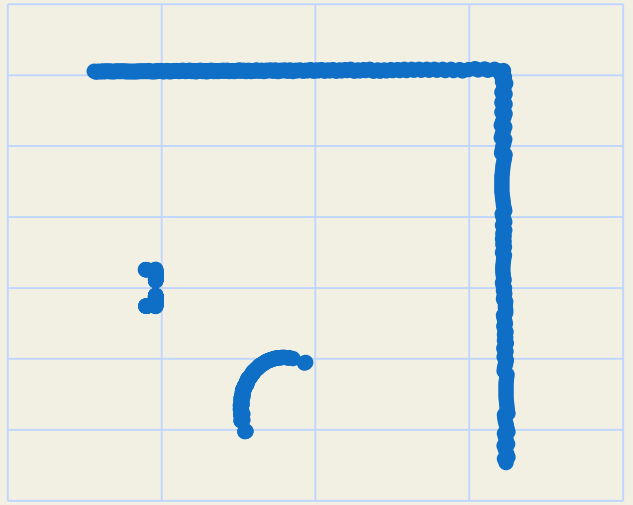
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

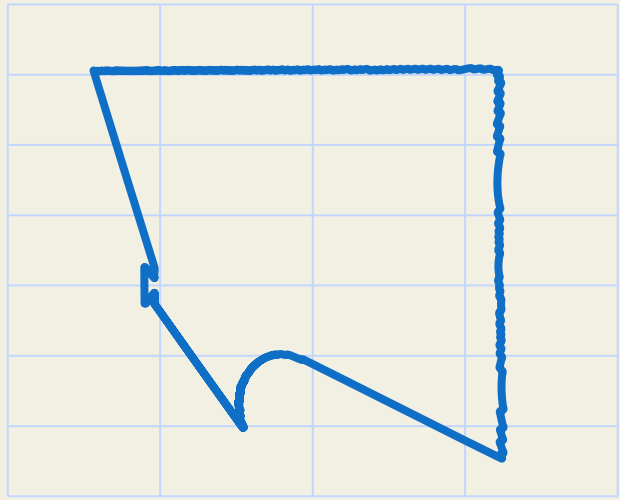


http://en.wikipedia.org/wiki/Normal_distribution

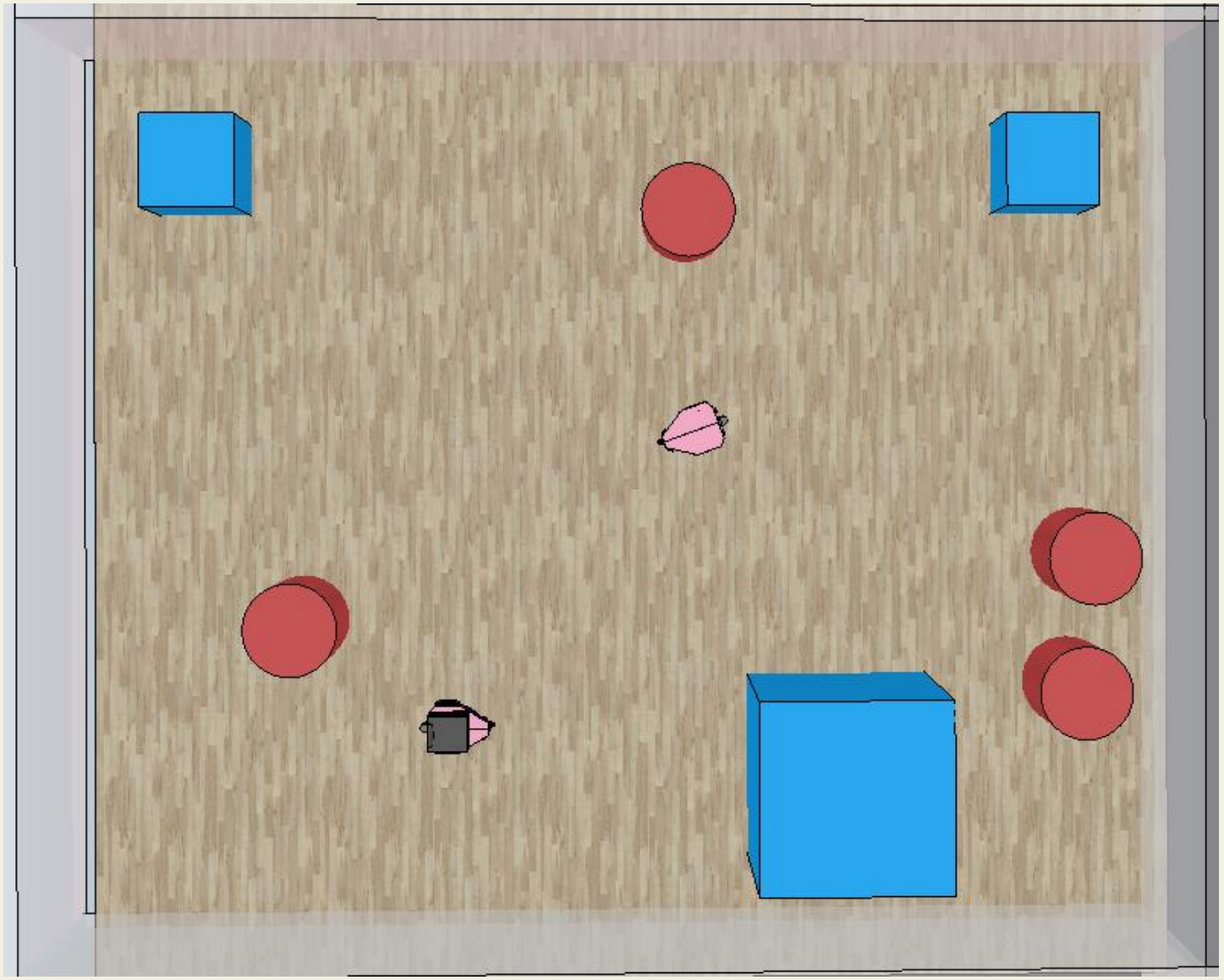
Lokalizáció

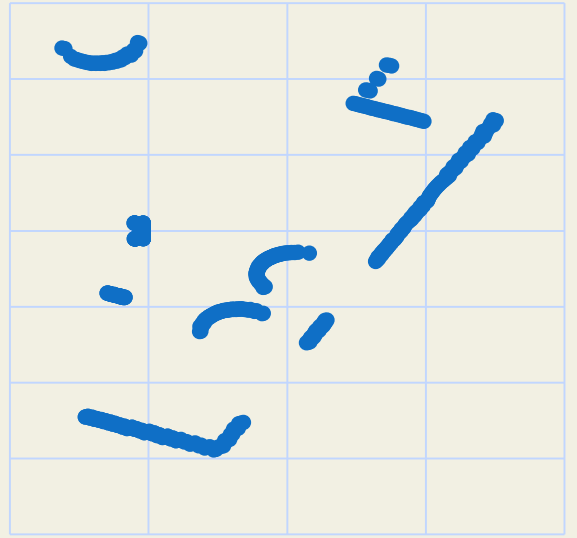


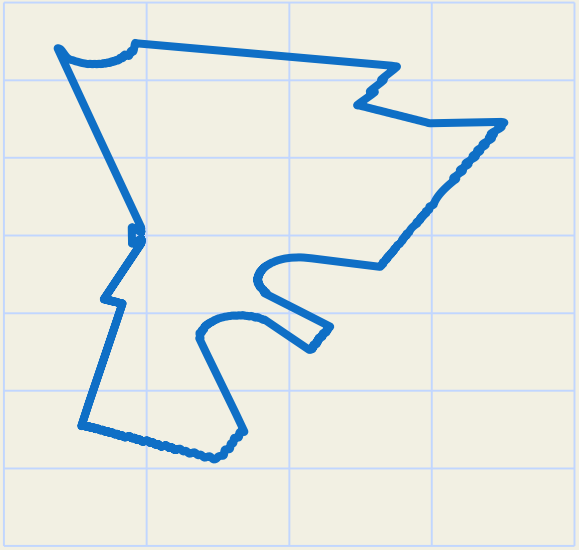




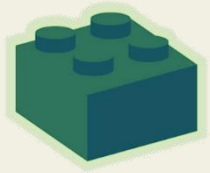
Lokalizáció







Felhasznált irodalom



- **A C programozási nyelv** - Az ANSI szerinti változat. **B. W.** Kernighan - **D. M. Ritchie**; Műszaki Könyvkiadó, 1995
- http://channel9.msdn.com/
- http://en.wikipedia.org/wiki/Robotics
- http://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping
- http://www.probabilistic-robotics.org/
- Nagy Gergely: C# szemelvények
- http://www.coppeliarobotics.com/helpFiles/