MODEL CSAPAT

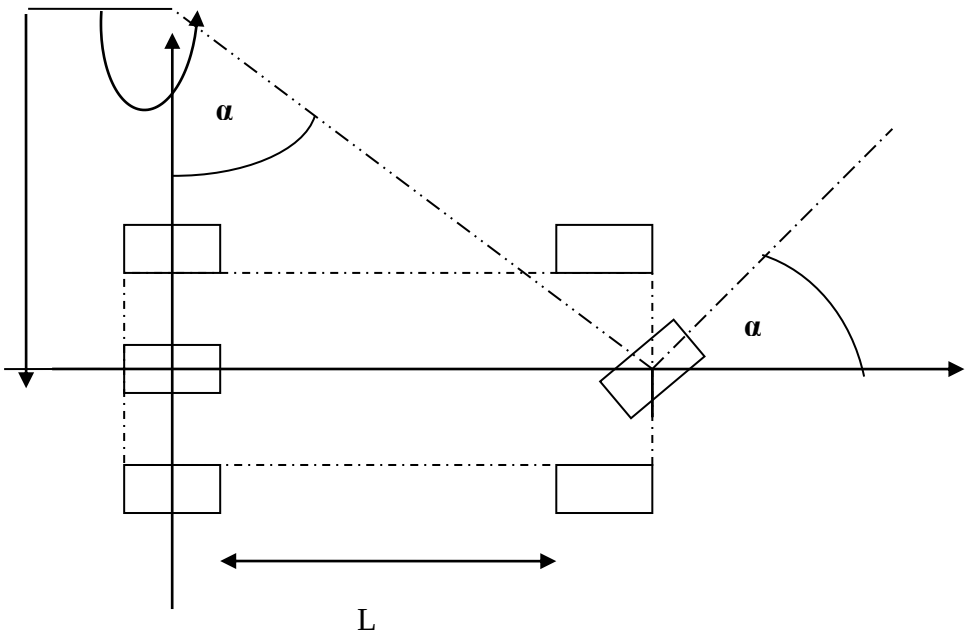
Többfajta modelltípusról beszélhetünk: taktikai, stratégia, szimulációs, leíró, diszkrét, folytonos, vegyes – egészértékű, determinisztikus, sztochasztikus modellek.

A mi feladatunk egy adott robot szimulációs modellezése, amely a vizsgált jelenséghez hasonló viselkedés mutatására képes, szimulálja az adott rendszer működését.

Egyfajta szimulációs modellt használunk, melynek alapja a kerékpár modell, más néven egynyomvonalú járműmodell. A bicikli és motorkerékpár modellek dinamikája a biciklik, motorkerékpárok és alkatrészeik mozgásainak tudománya, köszönhetően a rájuk ható erőkifejtéseknek. A dinamika a klasszikus mechanika egyik ága, ami a fizikában értelmezett. Bicikli mozgások körébe tartozik a kiegyensúlyozás, kormányzás, fékezés, gyorsítás, felfüggesztés, aktiválás és a vibráció. Ezek nagy részét mi nem használjuk a szimulációnkban, ugyanis ideális körülményeket feltételezünk. Ennek a mozgásnak a tanulmányozása a 19. század végén kezdődött, és napjainkban is tart. Modellünk lényegében az adott járműre történő paraméterillesztés után a jármű függőleges tengelye körül végzett forgómozgására (perdülés, legyezés) ad becslést.

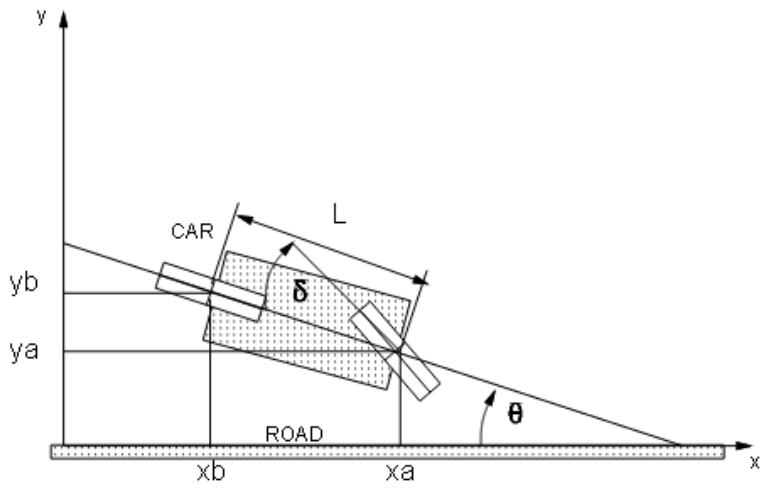
„A Bicikli-modell állapotváltozói: a perdülési szögsebesség és az úszási szög, melyek szabályozási célú felhasználásával a robot menetdinamikai tulajdonságai jelentősen javíthatók; például túlkormányzás (belső ív felé sodródás) vagy alulkormányzás (külső ív felé sodródás) esetén a robot kanyarmenetben az ideális íven tartható. E folyamatot a magyar szakirodalom Aktív Perdület Szabályozásnak (angol meg-felelője: AYC – Active Yaw Control) hívja.”[1]

[1] http://jovojarmuve.hu/content/imageup/cikk_pdf/8c6cb3f3bdd05a0455f815e86474dbd5.pdf
http://en.wikipedia.org/wiki/Bicycle_and_motorcycle_dynamics



L:két kerék közötti távolság

α: kormány elfordulási szöge



[2]

Θ:valós elfordulás szöge

[2] http://inst.eecs.berkeley.edu/~ee192/sp13/pdf/steer-control.pdf

Bicikli modell kinematikája

„A **kinematika** (mozgástan) a fizika azon részterülete, amelynek feladata a mozgások leírása. A mozgástant hagyományosan a mechanika tudományágába soroljuk, de feladata alapvetően matematikai jellegű. A mozgások leírása alatt azt értjük, hogy tetszőleges időpontban meghatározzuk egy test helyét, illetve helyzetét egy másik testhez képest.”[3]

A következő egyenletek adottak a mozgások felírására:

$\int xb = v*\cos\Theta$

$\int yb = v*\sin\Theta$

$\int \Theta = v*(1/L)*tg\boldsymbol{\alpha}$,

ahol v jelöli a robot sebességét.

Az egyszerűsítés miatt feltételezzük, hogy a jármű sebessége állandó. Ekkor csak egy vezérlő bemenete van a rendszernek, a kormányzási szög, melyet a hármas csapattól kapunk meg egységes formátumban (radián).

Inputként megkapjuk még az x, y koordinátákat, valamint az α (kormányzási) szöget. Ezek mindig változni fognak időben, paraméterként megadva hogy milyen időközönként. Mivel időben változik, a modell adott egyenleteit idő szerint kell integrálnunk. Minden időpillanatban ki kell számítanunk a soron következő trajektóriát (x, y koordináta, valamint a ϑ szög), ami megadja az útvonal koordinátáit folyamatosan, minden időpillanatában a robot mozgásának.

[3] http://hu.wikipedia.org/wiki/Kinematika

A matematikai modell, amit használunk a programunkban, így néz ki:

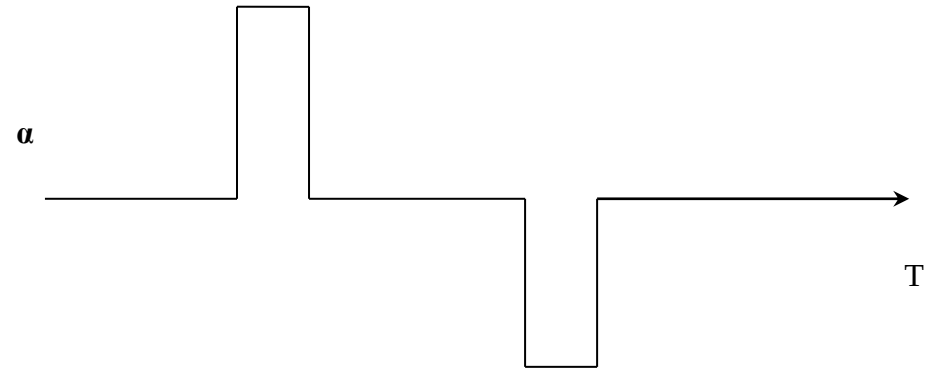
$\vartheta(i) = \vartheta(i-1) + (V*(1/L)*\text{tangens}(\boldsymbol{\alpha}(i)))$

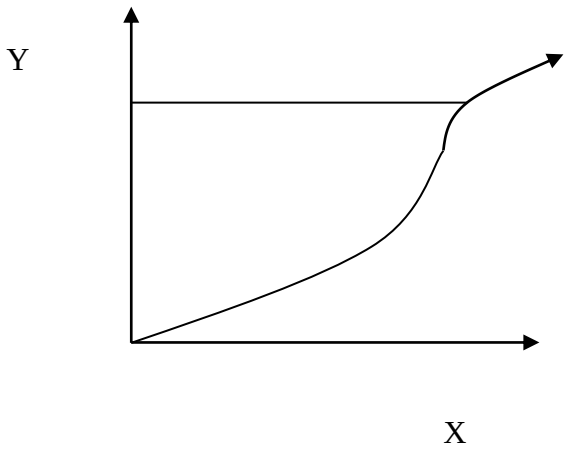
$x(i) = x(i-1) + (V*\cos(\vartheta(i-1))* \Delta T)$

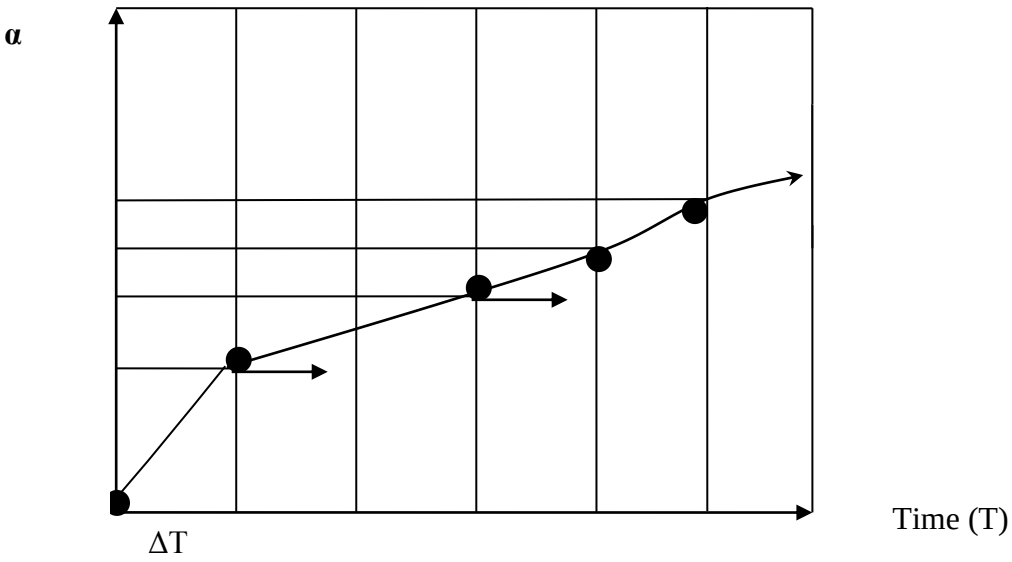
$y(i) = y(i-1) + (V*\sin(\vartheta((i-1))* \Delta T)$

x(i) az aktuális objektum x koordinátája,
y(i) az aktuális objektum x koordinátája,
Θ az aktuális objektum szöge

A program vezérlő ciklusába van ágyazva a modell, egy adott T intervallumig játszódik le a ciklus, mely az egész szimuláció időtartamát jelöli.
Az idő „finomítása”, egy paraméterként megadott adat, (ΔT), azt jelzi, hogy milyen időközönként van beosztva a robot teljes szimulációs ideje. Minél kisebb, annál pontosabb lesz a szimulációnk.





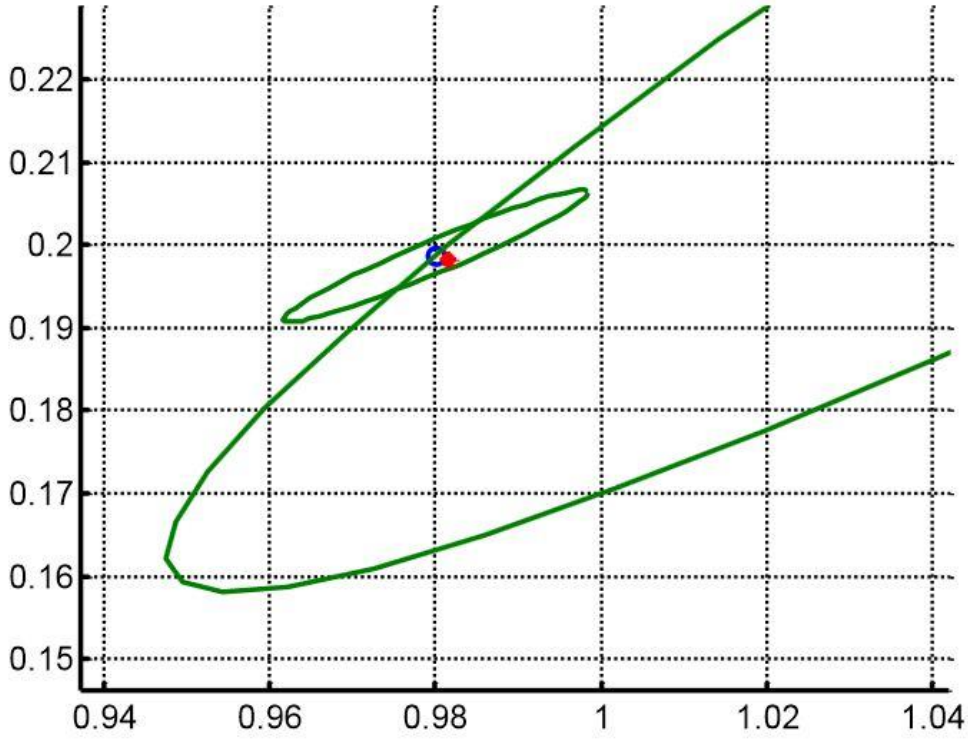


(Minél közelebb állnak egymáshoz a függőleges rácsvonalak, annál kisebb a ΔT, annál pontosabb lesz a számításunk.)

[4]Zaj kényszeres felhasználása: Approximáció, ellipszisek

Ideális esetben, számításaink arra szolgálnak, hogy meghatározzák a robot pontos lokalizációját. A valós világban azonban ez nem így van. A tervezett útvonal, és a szenzoros korrigálás után sem határozható meg a robot pontos helye, a számításba nem vett tényezők miatt, például súrlódás, vagy éppen a kerék csúszása. Csak a kerék forgásából számított út, így nem pontos. Mivel a robot nem pont azt az utat írná le, amit a szimuláció ad vissza, így szükség van mesterséges zajra. A meghatározott pontok köré egy ellipszist rajzolva kaphatjuk meg azon pontokat, ahol a robot valójában tartózkodik. Csak azt tudjuk, hogy valahol az ellipszisben található.

[4] http://hu.wikipedia.org/wiki/Approxim%C3%A1ci%C3%B3



A zaj úgy néz ki a programban, hogy egy 100-as nagyságrenddel kisebb, véletlen számmal torzítjuk el a valós értéket. Így a szimuláció, rendelkezik némi zajjal. Az x, y koordináták, és a szög generált eltérését, a 3-mas csapattól kapjuk paraméterben.
A robot folyamatosan közelíti a valós útvonalat, ez az approximációval lehetséges.

Approximáció: A matematikában approximációnak nevezzük azt az eljárást, mellyel függvények értékét próbáljuk közelíteni egyszerűbb függvényekkel, közben számosítva a lehetséges hibakorlátot. Szorosan kapcsolódik a tematikához a függvények Fourier-sorok általi közelítése, azaz ortogonális polinomokon alapuló sorozatok szummázásával való közelítés. Egy különlegesen érdekes probléma egy függvény közelítése számítógépes matematikai közegben, oly módon alkalmazva számítógépes operációkat (pl. összeadás, szorzás), hogy a megoldás az adott függvényt a lehető legjobban közelítse. Ez esetben tipikusan polinomiális vagy törtfüggvényekkel közelítünk.

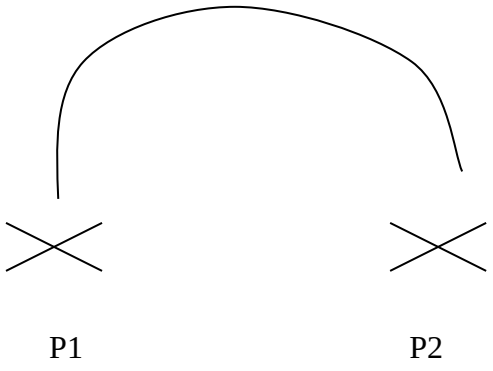
A cél a függvény közelítése oly módon, hogy az megközelítse a számítógép által szabott lebegőpontos aritmetikát. Ezt egy magas fokú polinom használatával érhetjük el és/vagy leszűkítve az értéktartományt, melyen a polinomnak közelítenie kell a függvényt. Napjainkban az értékkészletet gyakran kicsiny részekre bontjuk, külön-külön közelítve minden részét alacsony fokú polinomokkal.

- Optimális polinomok
- Csebisev approximáció
- Remez algoritmusa

Az approximáció segítségével, a robot folyamatosan korrigál a tervezett út felé, de nem tudja pontosan követni.

A program működéséről

Az egyszerűség kedvéért minden csoport JAVA nyelven írta a rész programját. A mi feladatunk egy olyan modul létrehozása volt, ami a konstruktorában paraméterként vár egy kezdeti pozíciót. A kezdeti pozíciót eltárolja egy **update** metódus segítségével az *utolso* objektumban, érdemes megjegyezni, hogy ez osztályszinten van megvalósítva, azért mert ezzel a módszerrel amikor meghívjuk újra és újra az objektumunkat akkor mindig az aktuálisan vett utolsó pozíció lesz benne. A funkció kibővítése érdekében a modul rendelkezik egy felültöltött konstruktorral is, ami már az alapon kívül zajt is tud kezelni. Ezen adatokból kellett nekünk meghatározni, hogy a robot várhatóan hova érkezik és hova fog „nézni” (mi lesz a szögállása), tehát ebből következően az osztályunk példányosítás után képes olyan üzenetet biztosítani a többi classifier-felé amit ők felhasználhatnak. Működés szemléltetése:



P1 az a „pont”, ahol a robot aktuálisan áll, a P2 pedig az, ahova érkeznie kellene számítások alapján. Minden állapotban van egy jelenlegi szöge, P1 állapotban lévő szög az, amit mi meg kapunk ha első futás van, különben pedig az előző pozíció szöge és P2 pozíció az, amit mi tovább adunk és természetesen a megváltozott szögállást is tovább adjuk. Ezen feladatok

megoldásához létrehoztunk egy **getNewPos** metódust, ami paraméterül kapja az alfát és képes a fent leírt matematikai összefüggések alapján kiszámolni azt a pontot és szöget ahova érkezünk és eltárolja az *utolso* objektumba. A metódus rendelkezik az aktuális pont felüldefiniálásával, ez az **addZaj** metódus, olyan módon, hogy ha van zaj, akkor felültölti (update), tehát a zaj koordinátáit hozzá adjuk a jelenlegi pont koordinátáihoz, így kapjuk meg a tényleges aktuális pontot**.** Ezeken kívül rendelkezik még a program getter és setter metódusokkal is. A többi funkciót a sajnos nem lehetett megvalósítani néhány csapat közreműködése nélkül.