SZÉCHENYI ISTVÁN EGYETEM
Műszaki Tudományi Kar
Informatikai tanszék

# Szimulációs technikák

( NGB_IN040_1)

## 2. csapat

## Comparator - Dokumentáció

**Mérnök informatikus BSc szak, nappali tagozat**

**2012/2013 II. félév**

# Tartalomjegyzék

# Bevezetés

A félév folyamán általunk elkészített alkalmazás egy önjáró robot működésének szimulációját valósítja meg. A programozáshoz a java nyelvet választottuk.

**A feladat teljesítéséhez 6 csapatra osztottuk szét a feladatokat:**

- Trajectory Planning
- Comparator
- PID
- Model
- Plot
- Sensor

Az alábbi dokumentáció a Comparator megvalósításának folyamatát mutatja be.

## Csapattagok:

- Albano Péter
- Bella Ádám
- Gregor Zoltán (csoport-vezető)
- Kelemen Balázs
- Molnár József

# A Comparator feladata

A Comparator modul feladata az inputként kapott pontokból, amelyek a robot követendő pályáját határozzák meg, és a robot aktuális pozíciójából, előállítani a "d" értéket minden egyes hívásnál, amely a pálya aktuális szakaszától való távolságot adja meg kimenetként. (az aktuális szakasz az előzetesen megtervezett útvonal egy része). Erre azért van szükség, mert az előzetesen megtervezett útvonalhoz képest eltérések adódhatnak, a robot a valóságban nem egy teljesen egyenes pályán fog haladni, így korrigálásra van szükség futása közben.

# Megvalósítás

A feladat megvalósításához készítettünk egy comparator.java osztályt.

A comparator.java hívásait a helper csomag osztályai szolgálják ki, melyeket a csapatok közösen hoztak létre: Line.java, Position.java, Pose.java. Ezeket használathoz beimportáljuk a helper-ből. Segítségükkel globálisan, a többi csapattal megegyezően azonos adattípusokat és adatszerkezeteket használtunk a különböző pontok, egyenesek, és közöttük lévő viszonyok számításához.

Továbbá az Icomparator.java-t amely a comparator elkészítéséhez szükséges interfész osztályt tartalmazza. Ennek segítségével, a többi csapat munkájától függetlenül, szinte kész állapotba fejleszthettük az osztályt.

A megoldáshoz elengedhetetlen a matematikai háttér, ezen belül is a koordináta geometria. Természetesen csak síkban lévő számításokra volt igény a szimuláció 2 dimenziós lefolyása miatt.

**Olyan műveletek alkalmazására volt szükségünk, mint például**

- két pont által meghatározott szakasz
- egy adott szakasz hossza
- két szakasz skaláris szorzata
- a robot elhelyezkedését meghatározó pont és az aktuális szakasz távolságának meghatározása.

$$\left|\overrightarrow{AB}\right| = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$$

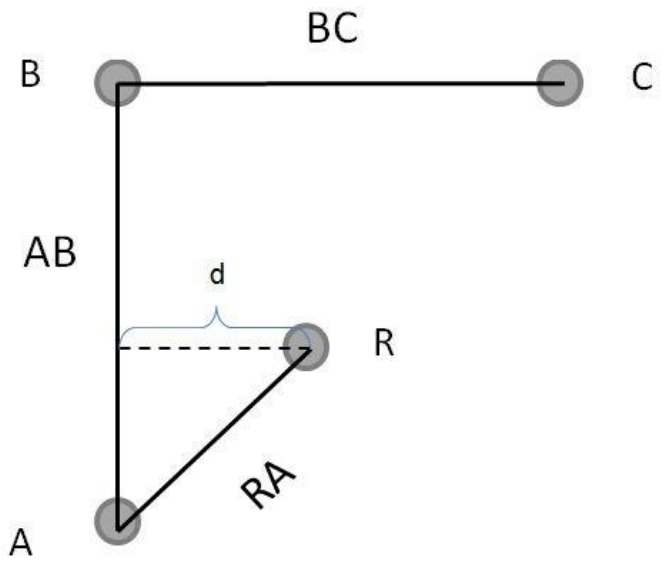
Két pont távolsága

$$(x_1, y_1) = (-y_1, x_1)$$

Merőleges egyenlete

$$skalar = x_1 * x_2 + y_1 * y_2$$

Két pont skaláris szorzata




1. ábra

**R:** a robot aktuális pozíciója

**A, B, C**: pedig a térkép egymást követő pontjai

**d:** pedig a távolság az aktuális szakasztól

Példában ez a szakasz az AB. Ha A-t és R-t összekötő szakasz skaláris szorzata A-t és B-t összekötő szakasszal nagyobb mint az A és B pontot összekötő szakasz hossza akkor R-t, a B-t és C-t összekötő szakaszhoz mérjük és töröljük az A pontot és ezzel az A-t és B-t összekötő szakaszt. Viszont amíg ez nem teljesül az A-t és B-t összekötő szakaszhoz viszonyítunk. Az összehasonlítás képlete: AB × AR > AB.

## Fejlesztési lehetőség

Nem kell megvárni míg nagyobb lesz a skaláris szorzat mint az aktuális szakasz, hanem egy előre megadott értékkel számítva, ha elégségesen megközelíti azt, akkor továbbléphetünk a következő szakaszra. Tehát: (AB*AR)> AB * 95%

# Comparator.java osztály működése

## Változók

| | |
|---|---|
| d | számított távolság értéke |
| act_rob_pos | a robot aktuális pozíciójának tárolása |
| smoothtraj_points | a követendő pálya pontjainak tárolására |

## Metódusok

| | |
|---|---|
| setSmoothTrajectory() | az aktuálisan követendő pálya pontjainak megadása a paraméterben megadott „List<Position>” típusú adatszerkezettel, a amelyet a „ smoothtraj_points”-ban tárolunk el. |
| setCurrentPose() | a robot aktuális pozíciójának megadása a paraméterben megadott „Pose p” típusú adattal, amelyet a „act_rob_pos”-ban tárolunk el. |
| getDistance() | visszaadja az aktuális szakasz és aktuális robotpozíció között aktuálisan meglévő távolságot, azaz a „d”–t, a paraméterben megadott getActLine() függvénnyel visszatérési értéke az aktuális szakasz ) és act_rob_pos adattal. **Ezt a függvényt fogja hívni PID.** |
| getSkalarHossz() | visszaadja a paraméterben megadott, 3 pont ( robot aktuális pozíciója és az aktuális szakasz kezdő és végpontja) által meghatározott szakaszok skaláris szorzatát egy „double„ típusú értékben. |

| | |
|---|---|
| getActLine() | visszaadja az aktuális szakaszt egy „Line” típusú adatszerkezetben, amelyet a robot aktuális pozíciója és az aktuális ponthalmaz első két pontja alapján, az előre meghatározott összefüggések segítségével, eldönti hogy a 3 pont által meghatározott szakaszok skaláris szorzata nagyobb e mint az aktuális ponthalmaz első két pontja által meghatározott szakasz.<br><br>Ha rövidebb, akkor az aktuális első két pont által meghatározott szakasszal tér vissza<br><br>Ha hosszabb, akkor az első pontot törli, majd a ponthalmaz „új” első és második pontjai által meghatározott szakasszal tér vissza. |

## Külső osztályok

- import helper.Pose;
- import helper.Position;
- import helper.Line;
- import interfaces.Icomparator;

## Futás közben

Az osztálypéldány létrehozása után, a „Trajectory Planning” megadja az aktuális követendő pálya ponthalmazát a „setSmoothTrajectory()” metódus paraméterében, továbbá megadja a robot kezdő pozícióját a „setCurrentPose()” metódus paraméterében. Ezek után a „getDistance()” függvény hívásával kérhető le a keresett „d” értéke, melyet a „PID” fog kérni.

A szimuláció futása közben, a „Sensor” folyamatosan frissíti a robot pozícióját a „setCurrentPose()” metódus paraméterbeni átadásával.