1. Előszó

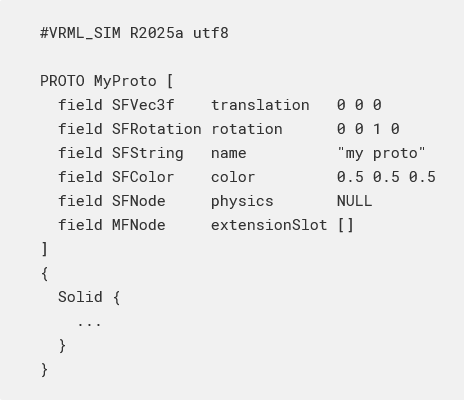
Ez a dokumentáció bemutatja a Webots szimulációs környezet három alapvető fogalmát: a protót, a controllert és a supervisort. Az egyes fejezetek részletesen ismertetik ezen elemek működését, használatát és a projektben betöltött szerepüket, kitérve a konkrét megvalósítások magyarázatára is.

# Protók áttekintése

## Általános áttekintés

A protó (angolul: PROTO) egy újrahasznosítható objektum sablon a Webots-ban. Lényegében minden, a szimulációs térben látható és kezelhető elem – például járművek, utak, épületek vagy a csomagot jelölő doboz – egy protóként van definiálva. Technikailag a protók különböző csomópontok (nodes) rendezett halmazai, amelyek egy komplex objektumot alkotnak.

A Webots egyik legnagyobb előnye a beépített protók széles könyvtára, amely jelentősen megkönnyíti a fejlesztést. Ezeket a meglévő sablonokat szabadon felhasználhatjuk, módosíthatjuk, vagy akár teljesen új, egyedi protókat is létrehozhatunk a semmiből. A program a protókat egyszerű, szöveges .proto fájlokban tárolja, amelyek könnyen szerkeszthetők.



1. ábra: példa protó fájl

Egy protó fájl általános szerkezete a következő: egy fejlécből, a protó nevéből, a paraméterezhető mezőkből (protoFields), valamint a protó testéből (protoBody) áll, amely a tényleges csomópontokat tartalmazza.



2. ábra: egy protó általános szerkezete

## A projektben használt protók

A szimulációhoz felhasználtunk egy beépített és egy egyedi készítésű protót.

### Tesla Model 3

A csomagszállítási feladatot egy, a Webots által biztosított Tesla Model 3 protó segítségével valósítottuk meg. A járművet egyedi controller vezérli, és kiegészítettük egy Display csomóponttal, amely egy kis ablakban jeleníti meg a sebességre és kormányzásra vonatkozó telemetriai adatokat.

A jármű a szimulációs világot leíró .wbt fájlban az alábbiak szerint lett definiálva:

DEF TESLA TeslaModel3 {

translation -10 5 0.5

rotation 0 0 1 -1.5707953071795862

controller "keyboard\_drive"

sensorsSlotCenter [

Display {

width 300

height 80

}

]

}

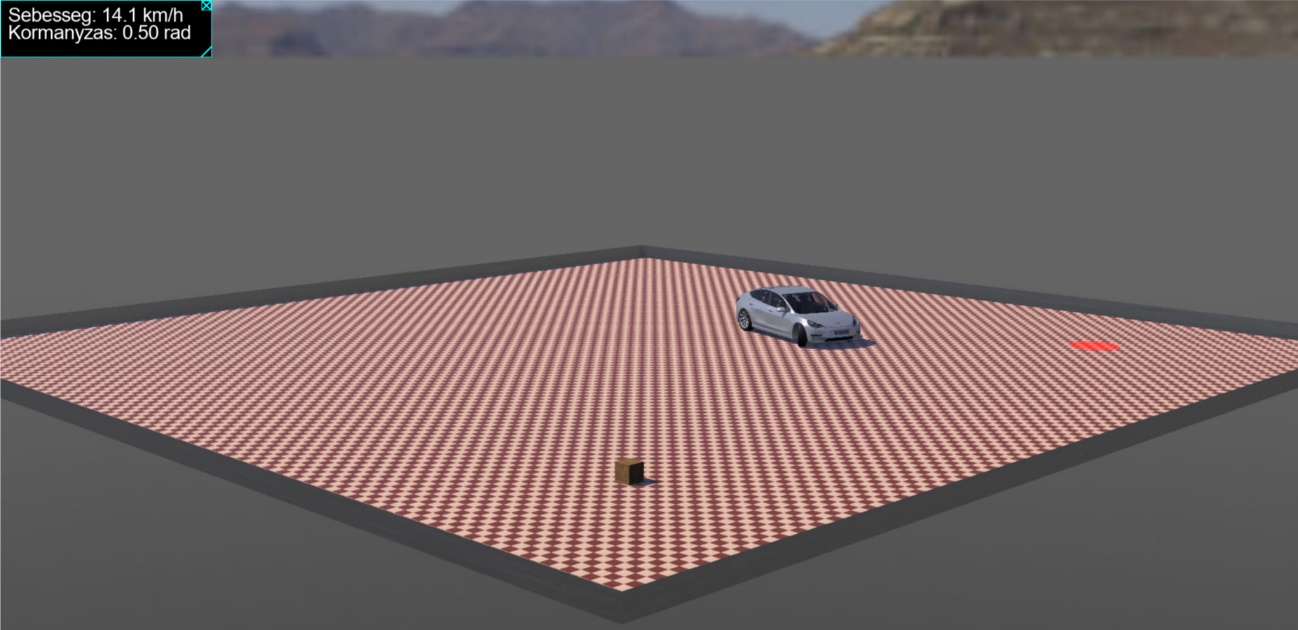
1. kódrészlet: Tesla protó kódja

* DEF TESLA: Egyedi nevet ad a protó példánynak, amellyel később hivatkozni lehet rá.
* translation: Az objektum pozíciója a 3D térben (x, y, z) méterben.
* rotation: Az objektum elforgatása egy tengely körül (x, y, z, szög) radiánban.
* controller: Meghatározza, hogy melyik vezérlő szkript felelős a jármű irányításáért.
* sensorSlotCenter: A jármű központi szenzorhelye, ahova a Display ablakot illesztettük.

A protó használatához szükséges a .wbt fájl elején importálni azt a Webots hivatalos GitHub repositoryjából.

EXTERNPROTO "https://raw.githubusercontent.com/cyberbotics/webots/R2025a/projects/vehicles/protos/tesla/TeslaModel3.proto"

2. kódrészlet: Szükséges import a Tesla működéséhez



3. ábra: Tesla működés közben

### DeliveryMarker (Saját protó)

A projekt tartalmaz egy teljesen egyedi készítésű protót is, amely a csomag kiszállítási helyét jelöli a pályán. Ez egy áttetsző, piros vagy zöld színű henger, amelynek tulajdonságai (pozíció, szín) paraméterezhetők.

#VRML\_SIM R2025a utf8

PROTO DeliveryMarker [

field SFVec3f translation 0 0.01 0

field SFColor color 1 0 0

]

{

Transform {

translation IS translation

children [

Shape {

appearance Appearance {

material Material {

diffuseColor IS color

emissiveColor IS color

transparency 0.3

}

}

geometry Cylinder {

height 0.02

radius 0.7

}

}

]

}

}

3. kódrészlet: Marker-t megvalósító protó

* PROTO DeliveryMarker [...]: Definiálja az új protó nevét és annak mezőit (translation és color).
* Transform: Egy csomópont, amely a gyermekeire (itt a Shape-re) alkalmaz geometriai transzformációkat, mint például az eltolást. Az IS translation összeköti a Transform csomópont translation mezőjét a protó translation paraméterével.
* Shape: Egy látható geometriai alakzatot definiál.
* Appearance és Material: Az objektum vizuális tulajdonságait határozzák meg, mint a szín (diffuseColor), az önálló világítás színe (emissiveColor) és az átlátszóság (transparency).
* Cylinder: A geometry csomópont, amely egy egyszerű hengert hoz létre megadott magassággal és sugárral.

# Controllerek áttekintése

## Általános áttekintés

A controllerek azok a szkriptek, amelyek a szimulációban szereplő robotok és járművek "agyát" képezik. Ezek felelősek a szenzoradatok feldolgozásáért és a végrehajtó szervek (motorok, kerekek) vezérléséért. A Webots támogatja a C, C++, Java, Python és MATLAB programozási nyelveket, így a fejlesztők a számukra leginkább ismert környezetben dolgozhatnak.

A projektben a Python nyelvet választottuk, mivel magas szintű nyelvi eszközei és a mobil robotikában elterjedt csomagjai egyszerűsítik a fejlesztést.

## A Teslát vezérlő controller

A projektben a Tesla manuális irányítását egy keyboard\_drive.py nevű Python controller valósítja meg. Ez a szkript a billentyűzetről beolvasott jelek alapján vezérli a járművet.

# --- Importok ---

from vehicle import Driver

from controller import Keyboard

# --- Konstansok ---

TIME\_STEP = 16

THROTTLE\_STEP = 0.05 # 0 es 1 kozotti ertek

BRAKE\_STEP = 0.1 # 0 es 1 kozotti ertek

STEERING\_STEP = 0.03 # Radian

MAX\_STEERING\_ANGLE = 0.5 # Radian

# --- Inicializalas ---

driver = Driver()

keyboard = Keyboard()

keyboard.enable(TIME\_STEP)

# --- HUD beallitasa ---

try:

display = driver.getDevice("display")

except:

display = None

print("ERROR: 'display' nem talalhato")

font = "Arial"

# --- Alap ertekek beallitasa ---

throttle = 0.0

brake = 0.0

steering = 0.0

gear = 1 # Alapertelmezett fokozat: elore menet

is\_gear\_key\_down = False

driver.setGear(gear)

print("↑↓ - gaz/fek, ←→ - kanyarodas, G - sebessegvalto, Space - kezifek")

# --- Esemenyek kezelese ---

while driver.step() != -1:

key = keyboard.getKey()

current\_key\_is\_G = (key in (ord('G'), ord('g')))

is\_throttle\_active = (key == Keyboard.UP)

is\_brake\_active = (key == Keyboard.DOWN or key == ord(' '))

is\_steering\_active = (key == Keyboard.LEFT or key == Keyboard.RIGHT)

# --- Gazadas ---

if is\_throttle\_active:

throttle = min(1.0, throttle + THROTTLE\_STEP)

brake = 0.0

# --- Fekezes ---

elif is\_brake\_active:

if key == Keyboard.DOWN:

brake = min(1.0, brake + BRAKE\_STEP)

elif key == ord(' '):

brake = 1.0

throttle = 0.0

# --- Kanyarodas ---

if key == Keyboard.LEFT:

steering = max(-MAX\_STEERING\_ANGLE, steering - STEERING\_STEP)

elif key == Keyboard.RIGHT:

steering = min(MAX\_STEERING\_ANGLE, steering + STEERING\_STEP)

# --- Valto ---

if current\_key\_is\_G and not is\_gear\_key\_down:

if throttle == 0.0: # Csak akkor, ha allo helyzetben van

gear = -gear

driver.setGear(gear)

brake = 0.0

print("Fokozat:", "Eloremenet" if gear == 1 else "Hatramenet")

is\_gear\_key\_down = True

elif not current\_key\_is\_G:

is\_gear\_key\_down = False

# --- Kormany visszaallitasa ---

if not is\_steering\_active:

steering \*= 0.90

# --- "Motorfek" szimulacioja ---

if not is\_throttle\_active and not is\_brake\_active:

throttle \*= 0.98

brake \*= 0.9

# --- Ertekek beallitasa ---

driver.setSteeringAngle(steering)

driver.setThrottle(throttle)

driver.setBrakeIntensity(brake)

# --- HUD ---

if display:

display.setColor(0x000000)

display.fillRectangle(0, 0, display.getWidth(), 60)

display.setColor(0xFFFFFF)

display.setFont(font, 20, True)

speed = driver.getCurrentSpeed()

display.drawText(f"Sebesseg: {speed:.1f} km/h", 10, 10)

display.drawText(f"Kormanyzas: {steering:.2f} rad", 10, 35)

4. kódrészlet: Tesla irányítását megvalósító controller

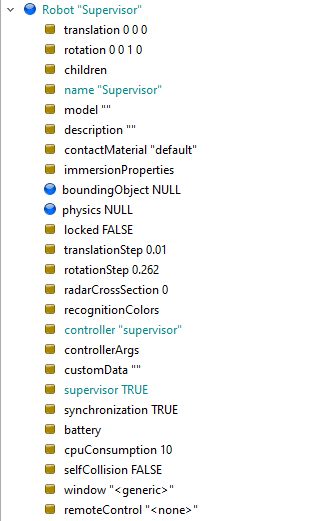
A controller kódjának főbb funkciói:

1. Inicializálás: Létrehozza a Driver és Keyboard objektumokat, amelyek a jármű vezérléséhez és a billentyűzet figyeléséhez szükségesek. Emellett beállítja a Display eszközt a HUD (Head-Up Display) megjelenítéséhez.
2. Fő ciklus: A while driver.step() != -1: ciklus a szimuláció minden egyes időlépésében lefut.
3. Billentyűleütések kezelése: A ciklus minden iterációjában lekérdezi a lenyomott billentyűt (keyboard.getKey()).
   1. Fel/Le nyilak: A gázadást (throttle) vagy a fékezést (brake) szabályozzák.
   2. Jobb/Bal nyilak: A kormányzást (steering) állítják.
   3. 'G' billentyű: Vált a hátra- és előremeneti sebességfokozat között, de csak álló helyzetben.
   4. Space: Vészfékezést (maximális fékintenzitást) aktivál.
4. Szimulált fizika: A kód finomítja a vezérlést:
   1. Ha a kormányzás nem aktív, a kerekek lassan visszaállnak egyenesbe (steering \*= 0.90).
   2. Ha sem a gáz, sem a fék nincs lenyomva, a kód "motorféket" szimulál a sebesség lassú csökkentésével (throttle \*= 0.98).
5. Parancsok kiadása: A kiszámított értékeket a driver objektum megfelelő függvényeivel (setSteeringAngle, setThrottle, setBrakeIntensity) továbbítja a járműnek.
6. HUD frissítése: A Display ablakra kirajzolja az aktuális sebességet és a kormányszöget.

# Supervisorok áttekintése

## Supervisor vs. Controller

A supervisor egy speciális jogosultságokkal rendelkező controller. Míg egy hagyományos controller csak a hozzárendelt robotot vagy járművet képes irányítani, a supervisor a szimuláció teljes egészét manipulálhatja: csomópontokat (objektumokat) hozhat létre, törölhet, vagy módosíthatja azok bármely tulajdonságát (pl. pozíció, szín) futás közben. A használatához egy Robot csomópontot kell létrehozni, és annak supervisor mezőjét TRUE-ra kell állítani.



4. ábra: Supervisor csomópontok

## A csomagokkezelő supervisor

A projektünkben a supervisor felel a csomagszállítási logika megvalósításáért: figyeli a jármű pozícióját, és kezeli a csomag felvételét és leadását.

# --- Importok ---

from controller import Supervisor

import math

# --- Konstansok ---

TIME\_STEP = 16

PICKUP\_DISTANCE = 4.5 # Meter

DROPOFF\_DISTANCE = 4.5 # Meter

TESLA\_DEF = "TESLA"

PACKAGE\_DEF = "PACKAGE"

MARKER\_DEF = "MARKER"

# --- Inicializalas ---

supervisor = Supervisor()

tesla = supervisor.getFromDef(TESLA\_DEF)

package = supervisor.getFromDef(PACKAGE\_DEF)

marker = supervisor.getFromDef(MARKER\_DEF)

if tesla is None or package is None or marker is None:

print("ERROR: Hianyzo NODE vagy DEF nev")

exit(1)

has\_package = False

delivered = False

# --- Jelolo szin reset ---

marker\_color\_field = marker.getField("color")

marker\_color\_field.setSFColor([1, 0, 0]) # Piros

# --- Tavolsag meghatarozasa ---

def distance(a, b):

return math.sqrt((a[0]-b[0])\*\*2 + (a[1]-b[1])\*\*2)

# --- Esemenyek kezelese ---

while supervisor.step(TIME\_STEP) != -1:

# --- Poziciok lekerese ---

tesla\_pos = tesla.getPosition()

package\_pos = package.getPosition()

marker\_pos = marker.getPosition()

if delivered: # Kiszallitas utan

continue

if not has\_package and distance(tesla\_pos, package\_pos) < PICKUP\_DISTANCE: # Csomag felvetele

print("Csomag felveve!")

has\_package = True

package.getField("translation").setSFVec3f([0, 0, -10]) # Kikuldjuk a vilagbol

elif has\_package and distance(tesla\_pos, marker\_pos) < DROPOFF\_DISTANCE: # Csomag leszallitasa

print("Csomag kiszallitva!")

has\_package = False

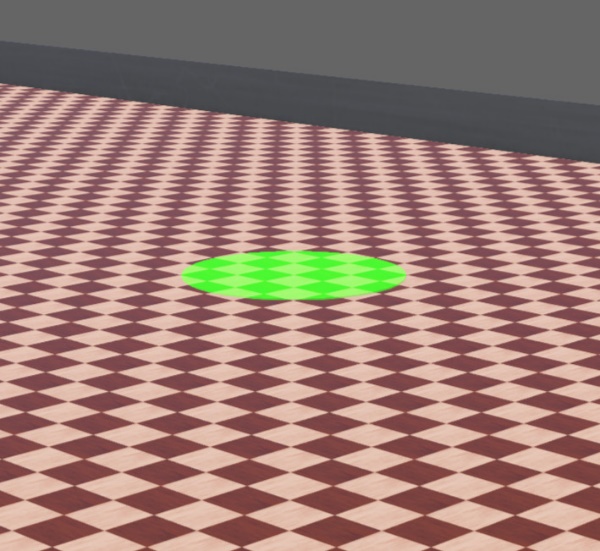
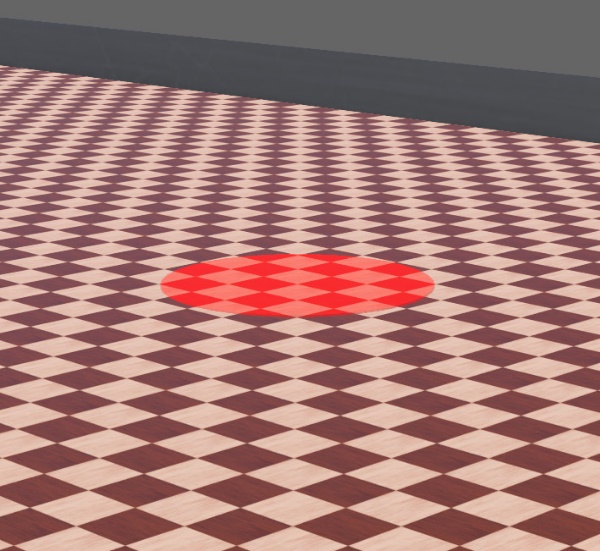
delivered = True

marker\_color\_field.setSFColor([0, 1, 0]) # Zold

5. kódrészlet: Csomagok kezelését végző supervisor

A supervisor.py szkript működése:

1. Inicializálás: A supervisor lekéri a szimulációban szereplő kulcsfontosságú objektumokat (a TESLA járművet, a PACKAGE dobozt és a MARKER jelölőt) a DEF nevük alapján. Emellett alaphelyzetbe állítja a jelölő színét pirosra.
2. Fő ciklus: A szimuláció minden időlépésében lefut.
3. Pozíciók lekérdezése: Folyamatosan lekéri a Tesla, a csomag és a céljelölő aktuális pozícióját.
4. Távolságszámítás: Egy segédfüggvény segítségével kiszámolja a Tesla távolságát a csomagtól és a célterülettől.
5. Csomagfelvétel: Ha a jármű a csomag közelébe ér (a PICKUP\_DISTANCE-nél közelebb) és még nem vette fel azt, a supervisor a csomagot "eltünteti" a pályáról azáltal, hogy egy láthatatlan helyre teleportálja ([0, 0, -10]).
6. Csomag leadása: Ha a jármű már felvette a csomagot és a céljelölő közelébe ér (a DROPOFF\_DISTANCE-nél közelebb), a supervisor a kézbesítést sikeresnek minősíti, és a jelölő színét zöldre változtatja.



5. ábra: Piros és zöld jelölő

# Összefoglalás

Ez a dokumentáció bemutatta a Webots szimulációs szoftver három kulcsfontosságú elemét: a protókat, mint újrahasznosítható objektumokat, a controllereket, mint a robotok viselkedését meghatározó szkripteket, és a supervisorokat, amelyek a szimulációs környezet egészét képesek dinamikusan manipulálni.

A bemutatott projektben ezen eszközök segítségével egy manuálisan, billentyűzettel irányítható járművel megvalósított csomagszállítási feladatot hoztunk létre. Ez a rendszer stabil alapot biztosít a további fejlesztésekhez, amelyek végső célja egy teljesen önvezető, szenzorokkal felszerelt csomagkihordó jármű létrehozása.