Gyors prototípustervezés

(IIT1)

Mérési útmutató

Rendszertervezés laboratórium 1

BMEVIMIAC03

Összeállította: Kovács Gábor

gkovacs@iit.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem

Irányítástechnika és Informatika Tanszék

2018

Tartalomjegyzék

1 Bevezetés

Az ipari gyakorlatban elterjed azon gyors prototípustervező eszközök használata, melyek lehetővé teszik a szabályozási körök megtervezését, a szakasz modelljével együtt történő szimulációját, majd az implementált irányítási algoritmus validációját és a megvalósított beágyazott irányítórendszer működésének ellenőrzését is.

A tervezési feladat első lépése mindig a modell megalkotása, akár mérések (identifikáció), akár a megfelelő fizikai (kémiai, pénzügyi stb.) egyenletek és paraméterek alapján. A modell ismeretében már megtervezhető a feladat által megkívánt komplexitású és minőségű irányítási algoritmus, a zárt szabályozási kör működése pedig a modellek alapján számítógépes környezetben ellenőrizhető (MIL-szimuláció). Ez a lépés nem csak az algoritmus helyességének validálására, hanem az abban szereplő paraméterek hangolására is lehetőséget nyújt.

A megtervezett algoritmust ezután implementálni kell a választott platformon, majd az irányítórendszer működését validálni kell. Ennek eszköze a HIL-teszt, melynek során a végleges platformon futó irányítórendszert fizikai be- és kimenetein keresztül kötjük össze egy, a szakasz modelljét futtató eszközzel. Az eljárás lehetőséget nyújt arra, hogy a költséges és esetenként veszélyes technológiát mellőzve, a modell paramétereit tetszőlegesen állítva validálhassuk az irányítórendszer helyes működését.

A mérés célja, hogy egy egyszerű feladat megoldása során bemutassa a Matlab-Simulink alapú gyors prototípustervezés egyes fázisait. A modellépítés, valamint a szabályozás megtervezése és MIL-szimulációval történő hangolása és validálása után az irányítórendszert beágyazott platformon is megvalósítjuk, a feladat által megkívánt érzékelő és beavatkozó szervek sajátosságait figyelembe véve. Ezt követően a szakasz modelljét a megfelelő fizikai interfészekkel rendelkező platformon futtatva, az irányítószoftvert futtató kontrollerrel összekapcsolva a zárt rendszer működését Hardware-In-the-Loop (HIL) szimuláció segítségével is ellenőrizzük.

A példarendszer egy elektromos jármű, melyhez egyszerű, csak gyorsításra képes sebességtartó automatikát (tempomatot) tervezünk és implementálunk.

A mérés első és második feladata (a járműmodell megvalósítása és a sebességtartó automatika tervezése) egymásra épül, ezek eredményére alapozva a harmadik és negyedik feladat (a tempomat implementálása beágyazott platformon illetve a járműmodell előkészítése HIL-szimulációhoz) két számítógép használatával párhuzamosan is elvégezhető. Természetesen a végleges HIL-teszthez mindkét feladat eredményére szükség van.

2 A járműmodell megvalósítása

Ahhoz, hogy a tempomatvezérlő algoritmusának működését ellenőrizzük, illetve a szabályozó paramétereit behangolhassuk, elsőként a jármű dinamikus modelljét kell megalkotnunk.

2.1 A jármű fizikai modellje

Egy elektromos jármű egyszerűsített, pozitív sebességekre érvényes, fékhatást nem tartalmazó longitudinális dinamikus modelljét a következők szerint írhatjuk le. A járművet az

$$F_M = F_{max}(v)T$$

erő gyorsítja, ahol $F_{max}(v)$ a sebességfüggő maximális motorteljesítményhez tartozó gyorsító erő, $T = [0,1]$ pedig a „gázpedál" állása (a kívánt gyorsító erő aránya az elérhető maximálishoz képest). Ezzel az erővel ellentétes irányú a légellenállás okozta

$$F_D = \frac{1}{2} C_d A \rho v^2$$

erő. Amennyiben a jármű nem sík úton mozog, akkor a lejtő vagy emelkedő hatását is figyelembe véve a jármű gyorsulása

$$m\dot{v} = F_M - F_D - mg\sin\alpha \Rightarrow \dot{v} = \frac{1}{m}(F_M - F_D - mg\sin\alpha)$$

alapján számítható, ahol m a gépjármű tömege, g a nehézségi gyorsulás, α pedig a lejtő vagy emelkedő szöge (emelkedő esetében pozitív). A jármű paramétereinek értelmezését és a konkrét értékeket az 1. táblázat tartalmazza.

1. táblázat- A járműmodell paraméterei

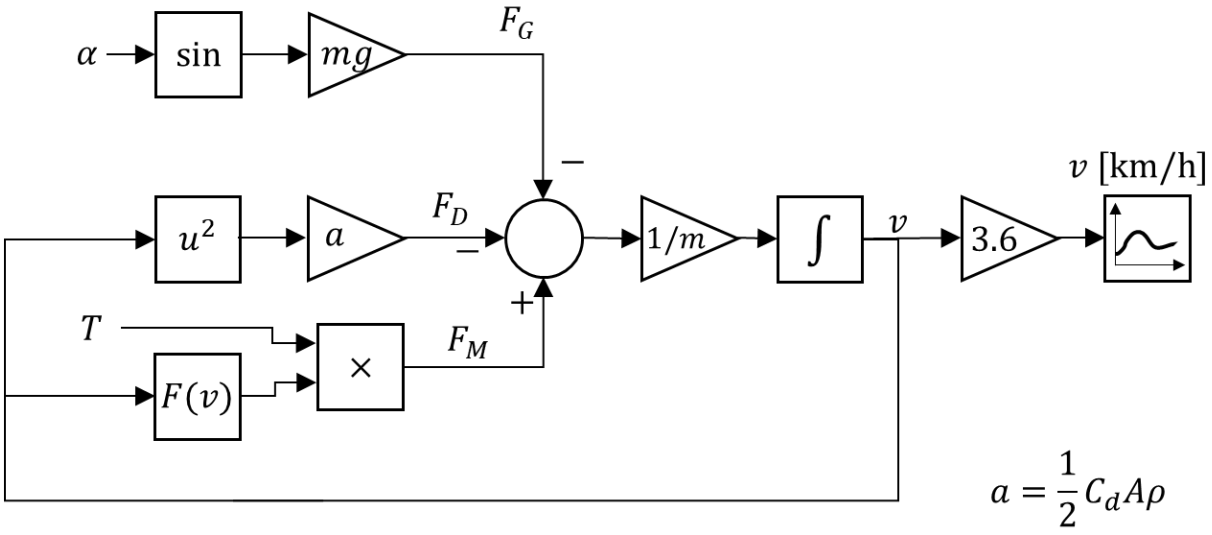
Paraméter	Értelmezés	Érték / Mértékegység
$C_d A$	A jármű homlokfelületének és légellenállási együtthatójának szorzata	$0.576\ \mathrm{m}^2$
g	Nehézségi gyorsulás	$9.81\ \mathrm{m/s}^2$
M	A jármű tömege	$2200\ \mathrm{kg}$
ρ	A levegő sűrűsége	$1.2\ \mathrm{kg/m}^2$
v	A jármű sebessége	$\mathrm{m/s}$

Elektromos meghajtású járművek esetén a maximális gyorsító erő egy adott sebességhatárig állandó, utána pedig exponenciális jelleggel csökken (ennek köszönhetően menettulajdonságaik alacsonyabb sebességeken jóval rugalmasabbak, mint magasabb sebességek esetén). A mérés során használt jármű esetén a maximális gyorsító erő $20\ \mathrm{m/s}$ ($72\ \mathrm{km/h}$) sebességig $16\,500\ \mathrm{N}$, onnan pedig a sebességgel arányosan csökken. A sebesség és a maximális gyorsító erő közti kapcsolat nemlineáris, a 2. táblázatban szereplő értékekkel adott (a mintapontok között köbös interpolációt használunk majd, 53 m/s sebesség felett pedig a gyorsító erőt állandónak feltételezzük).

2. táblázat - A sebesség - gyorsító erő karakterisztika mintapontjai

Sebesség	Maximális gyorsító erő
20 m/s [72 km/h]	14 500 N
27 m/s [97 km/h]	7 200 N
36 m/s [130 km/h]	3 800 N
53 m/s [190 km/h]	1 900 N

A rendszer blokkdiagramjának felépítését az 1. ábra mutatja be.

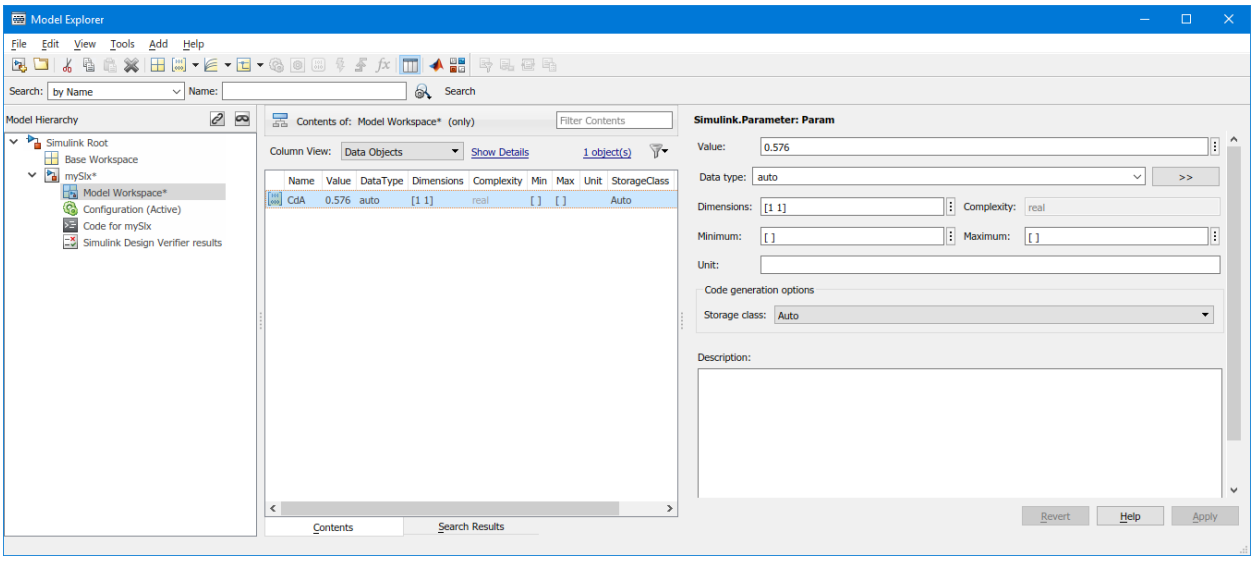


1. ábra – A járműmodell blokkdiagramja

2.2 A modell megvalósítása Simulink-környezetben

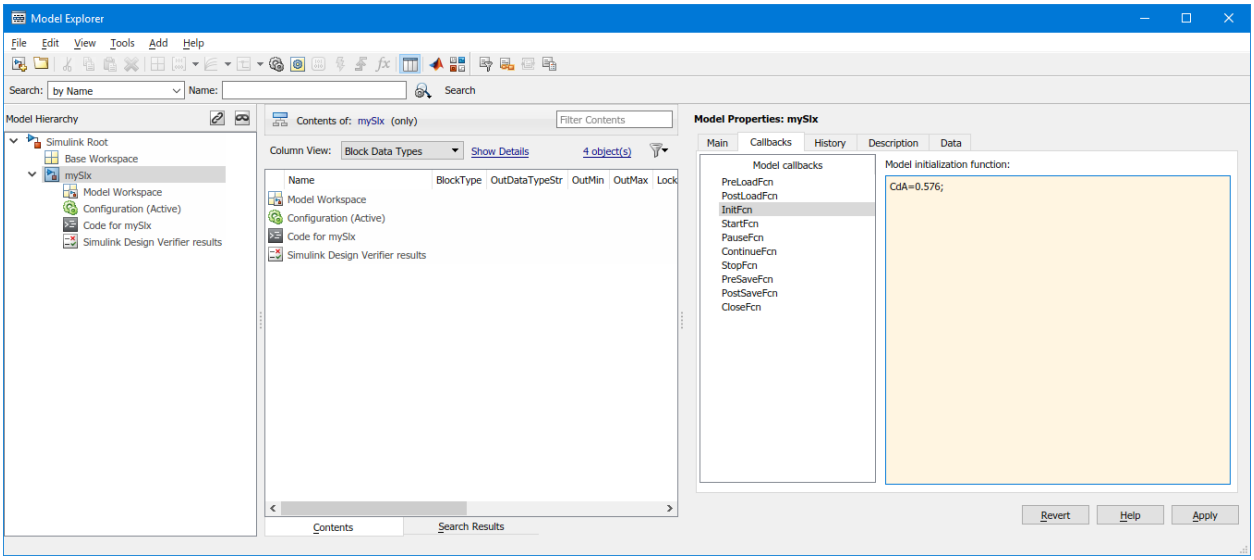
A modell megvalósítása során az átláthatóság érdekében célszerű a numerikus paramétereket Simulink-paraméterekként vagy változókként hozzáadni a modellhez, így a modellt leíró blokkok paraméterezésekor szimbolikus néven is hivatkozhatunk rájuk. Erre a Simulink *Model Explorer* eszköze (`Tools > Model Explorer` menüpont) használható, melynek segítségével több módon is rendelhetünk szimbolikus azonosítókat a modellben használt numerikus értékekhez.

Első lehetőségként a *Model Explorer* ablakának bal oldali, *Model Hierarchy* paneljén az aktuális Simulink-modellt lenyitva, majd azon belül a *Model Workspace* pontot választva rendelhetünk Simulink-paramétereket a modellhez. Egy paramétert akár az `Add > Simulink Parameter` menüparanccsal, akár a `Ctrl+P` billentyűkombinációval hozzáadhatunk, melynek hatására az új parameter megjelenik az ablak középső paneljén látható táblázatban (2. ábra). Itt a megfelelő sort kiválasztva a paraméter szimbolikus nevét a *Name*, értékét pedig a *Value* mezőben adhatjuk meg (a további beállításokra most nincsen szükség).



2. ábra – Simulink-paraméterek hozzáadása

Másik lehetőségként a modell inicializálásakor lefutó kódban is definiálhatjuk a paramétereket, egyszerű értékadás segítségével (Matlab-ban nincs szükség a scriptek változóinak explicit deklarációjára). Ehhez a *Model Explorer* ablakában a megfelelő Simulink-modellt kijelölve (magát a modellt, nem pedig a hierarchiában alá tartozó elemek valamelyikét!), majd a jobb oldali panelen a *Callbacks* fület, azon belül pedig az *InitFcn* pontot kell kiválasztanunk, majd a 3. ábrán látható módon a *Model initialization function* beviteli mezőben megadnunk a megfelelő utasításokat (pl. `CdA=0.576;`). Ezek a sorok a szimuláció indításakor, még a modell blokkjainak első kiértékelése előtt lefutnak, így azokban az itt definiált paraméterek szimbolikus formában használhatók.



3. ábra – Paraméterek definiálása a modell inicializáló függvényében

A következőkben a nemlineáris járműmodell megvalósítása, illetve a későbbi feladatok során használható blokkokat tekintjük át (a blokkok működésének leírásakor u a blokk bemenetét, y pedig a blokk kimenetét jelöli).

Integrator

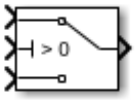
Egy jel integrálására (példánkban a jármű gyorsulásának integrálására, és így a sebesség meghatározására) az `Integrator` blokk (`Simulink > Continuous > Integrator`) használható, melynek működése a következő egyenlettel írható le: $y(t) = \int_0^t u(\tau)d\tau + \mathrm{IC}$.

4. ábra – Integrator blokk

A blokkra duplán kattintva megnyíló párbeszédablakban megadható az integrátor kezdeti állapota (*Initial Condition*, IC, ez a kimenet értéke is a $t = 0$ időpillanatban), valamint beállítható többek között az integrátor értékének alsó és felső korlátja. Mivel a modell csak pozitív sebességek esetén írja le a rendszer működését, érdemes az integrátor értékét (a jármű sebességét) 0 és 100 [m/s] közé korlátozni.

Switch

Amennyiben egy feltételtől függően különböző jeleket szükséges továbbítani, az esetválasztást lehetővé tevő (a megszokott `If-Then-Else`, illetve inkább a C nyelv `?:` operátorához hasonló) `Switch` blokk (`Simulink > Signal Routing > Switch`) használható. A blokk az első (legfelső) vagy harmadik (legalsó) bemenetére kötött értéket továbbítja a kimenetre a második (középső) bemenetre kötött érték és a beállított feltétel függvényében.



5. ábra - Switch blokk

A blokk konfiguráció párbeszédablakában beállítható feltétel az alábbiak közül választható:

- az első bemenet továbbítása, ha a második bemenet értéke a blokk párbeszédablakában beállított küszöbértéknél (*threshold*) szigorúan nagyobb
- az első bemenet továbbítása, ha a második bemenet értéke a küszöbértéknél nagyobb vagy egyenlő
- az első bemenet továbbítása, ha a második bemenet értéke nem nulla

Ha a feltétel nem teljesül, a harmadik bemenetre kötött érték kerül továbbításra.

Gain

Egy jel állandó értékkel való szorzására az erősítést megvalósító `Gain` blokk (`Simulink > Math Operations > Gain`) használható. Amennyiben a blokk erősítése K, akkor a működés az $y = Ku$ egyenlettel írható le.

6. ábra – Gain blokk

A blokk erősítése az arra kétszer kattintva megnyíló párbeszédablak *Gain* paramétereként állítható be. Az erősítés lehet literális érték, a modellben definiált paraméter, Matlab-változó, illetve ezeket tartalmazó kifejezés (pl. `0.5*GainVal`). Szintén itt állítható be a szorzási mód (elemenkénti, balról illetve jobbról történő szorzás), mely vektor jelek és vektor- vagy mátrix erősítések esetén meghatározza az alkalmazandó szorzási szabályt (a modellünkben előforduló skalár jelek és erősítések esetén ennek értéke indifferens).

Add

Jelek összeadására illetve kivonására legkényelmesebben az `Add` blokk (`Simulink > Math Operations > Add`) használható. A blokk alapértelmezésben két bemenettel rendelkezik, melyek összegét továbbítja a kimenetre. Amennyiben több bemenetre van szükségünk, illetve jeleket szeretnénk kivonni egymásból, a blokk párbeszédablakának *List of Signs* paraméterét megváltoztatva definiálhatunk további, megfelelő előjelű bemeneteket. Ha ezt például `+-+` értékre állítjuk, akkor a blokk három bemenetet kap, egyenlete pedig a következőképpen alakul: $y = u_1 - u_2 + u_3$.

7. ábra – Add blokk

Math Function

Alapvető matematikai függvények használatát a `Math Function` blokk (`Simulink > Math Operations > Math Function`) teszi lehetővé. Ezen függvények közé tartozik többek között a négyzetre emelés (*square*), gyökvonás (*sqrt*), hatványozás, az exponenciális (*exp*) és logaritmikus függvény (*log*). A használni kívánt függvény a blokk párbeszédablakában egy legördülő listából választható ki (amennyiben szükséges, például hatványozás esetén, a blokk bal oldalán egy további bemenet is megjelenik).

8. ábra – Math Function blokk

Trigonometric function

Trigonometrikus függvények (*sin*, *cos*, *tan* stb.) a `Trigonometric Function` blokk (`Simulink > Math Operations > Trigonometric Function`) segítségével értékelhetők ki a blokkdiagramban. A használni kívánt függvény a matematikai függvény blokkhoz hasonlóan a párbeszédablak legördülő listájából választható ki. Fontos megjegyezni, hogy a blokk a bemenetét radiánként értelmezi – amennyieb szükséges, egy `Degrees to Radians` blokk (`Simulink Extras > Transformations > Degrees to Radians`) segítségével a fok-radián konverzió egyszerűen elvégezhető.

9. ábra – Trigonometric function blokk

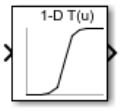
Saturation

Egy jel telítéses karakterisztikája legegyszerűbben egy `Saturation` blokk (`Simulink > Discontinuities > Saturation`) segítségével biztosítható. A blokk párbeszédablakában beállítható a jel megengedett alsó határa (`Lower Limit`, L) és felső határa (`Upper Limit`, U) is, a kimenet a következő módon áll elő:

$$y = \begin{cases} U & ha\ u \geq U \\ u & ha\ L < u < U \\ L & ha\ u \leq L \end{cases}$$

1-D Lookup Table

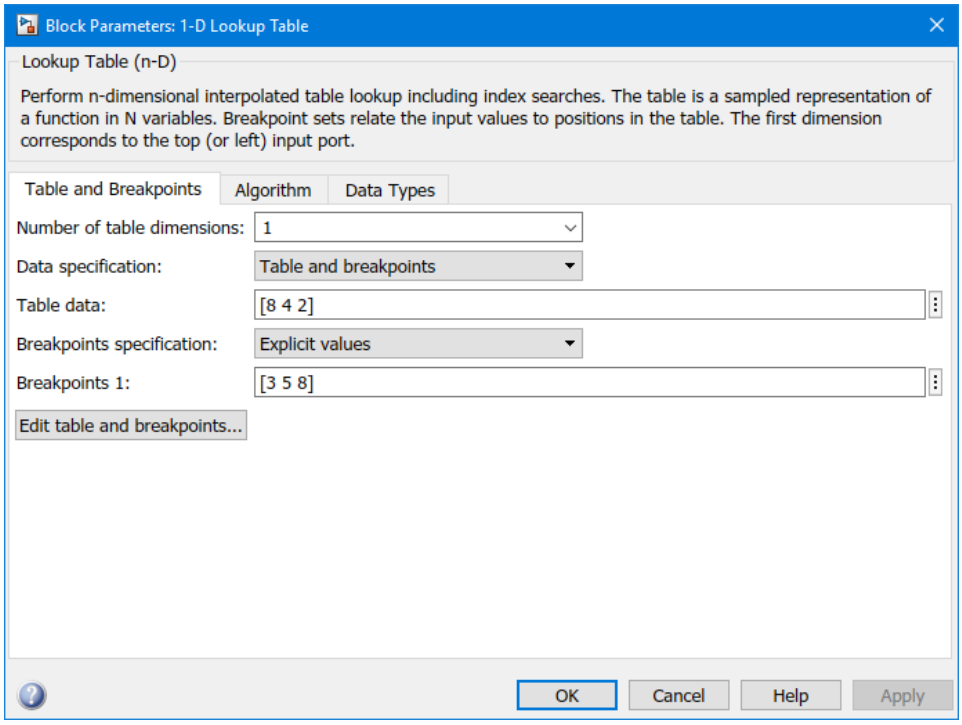
Amennyiben egy $y = f(u)$ függvény analitikusan nem, vagy csak nehezen írható le, használhatunk lookup-táblát is. Ebben az esetben a függvénykapcsolatot (u_i, y_i) mintapárok formájában adhatjuk meg, a `1-D Lookup Table` blokk lineárisan vagy köbösen interpolál illetve extrapolál a mintapontok között illetve azokon kívül.



10. ábra - 1-D Lookup Table blokk

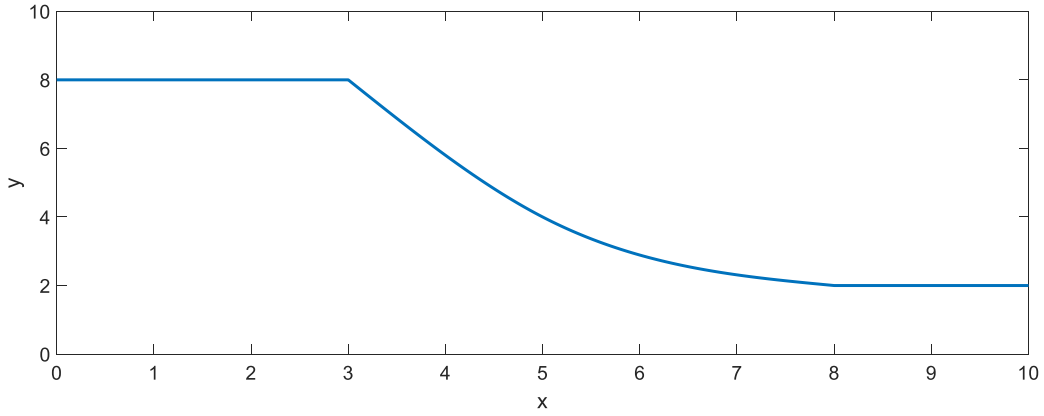
Az u_i pontok a konfigurációs párbeszédablak `Breakpoints 1` beviteli mezőjében adhatók meg (vektor formában), míg az y_i pontok a `Table data` mezőben adhatók meg (szintén vektorként), ld. 10. ábra. Az összetartozó $u_i - y_i$ pontokat a vektorok azonos indexű elemeiben kell szerepeltetni, opcionálisan az `Edit table and breakpoints...` gombra kattintva a párok táblázatos formában is megadhatók. Az interpoláció illetve extrapoláció módját az `Algorithm` pontban állíthatjuk be. A megadott mintapontok

között lineáris (`linear`) vagy köbös (`cubic`) interpolációt választhatunk, a tartományon kívüli extrapoláció esetén pedig ezeken kívül a `clip` opciót is használhatjuk. Ez utóbbi azt biztosítja, hogy a tartományon kívüli $u < u_1$ értékekhez az y_1, míg az $u > u_N$ értékekhez az y_N értékek kerüljenek hozzárendelésre.



11. ábra - A lookup-tábla beállításai

Beágyazott targetek esetén a lookup-tábla további előnye, hogy főként komplex, többváltozós függvénykapcsolat esetén a keresés a számításoknál jóval kisebb erőforrás-igényű.



12. ábra - Az interpoláció eredménye a 11. ábrán szereplő mintapont-párok, köbös interpoláció és clip extrapoláció használata mellett

Constant

A diagramba egy állandó értékű jelforrást a `Constant` blokk (`Simulink > Sources > Constant`) használatával illeszthetünk. A forrás értékét, mely lehet literális, a Simulink-modellben vagy a Matlab munkaterében definiált paraméter vagy változó, illetve ezeket tartalmazó kifejezés, a blokk párbeszédablakában állíthatjuk be.

13. ábra – Constant blokk

Signal Builder

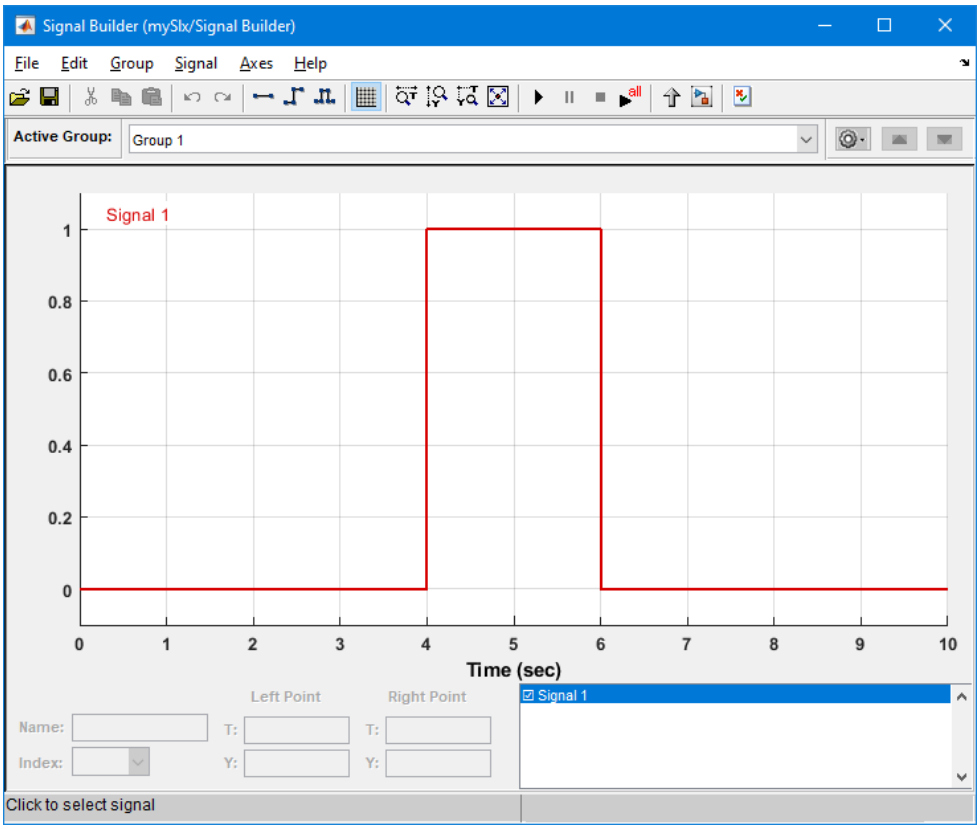
A Signal Builder blokk (Simulink > Sources > Signal Builder) olyan jelforrás használatát teszi lehetővé, melynek időfüggvényét lineáris szakaszokból egy grafikus felület segítségével adhatjuk meg.

14. ábra – Signal builder blokk

A blokkra duplán kattintva megjelenő ablakban alapértelmezésben a 15. ábrán látható időfüggvény látható, ez jelenik meg a szimuláció során a `Signal Builder` blokk kimenetén. A jelforma módosítására a görbére kattintás után következő lehetőségek állnak rendelkezésre:

- a jelalak vízszintes szakaszait a függőleges értéktengely mentén, függőleges szakaszait pedig a vízszintes időtengely mentén az egér segítségével mozgathatjuk
- egy vízszintes vagy függőleges szakaszt kiválasztva az ablak bal alsó részén látható beviteli mezők közül a bal oldali pontra vonatkozó *Y* illetve *T* paramétert módosítva a szakasz helyzetét pontosan beállíthatjuk
- egy sarokpontot az egérrel az értéktengellyel párhuzamosan mozgathatunk
- a Shift gomb lenyomása mellett a görbére kattintva ahhoz új sarokpontot adhatunk
- az időtengely tartományát az `Axes > Change Time Range…` menüpontban módosíthatjuk

A szerkesztőablak bezárásával a görbe mentésre kerül, a szimuláció indítása után annak kimenetére az általunk megszerkesztett jelalak jut.



15. ábra – A Signal builder szerkesztőablaka

Display

Egy jel numerikus értékét a szimuláció illetve külső módban történő futtatás során egy `Display` blokk (`Simulink > Sinks > Display`) segítségével jeleníthetjük meg. A Simulink a kijelzőn folyamatosan frissülve jeleníti meg az értéket, így a valós időnél gyorsabb szimuláció során ez a blokk gyorsan változó jelek esetén csak korlátozottan alkalmazható, külső módban történő futtatás esetén azonban gyakran jól használható.

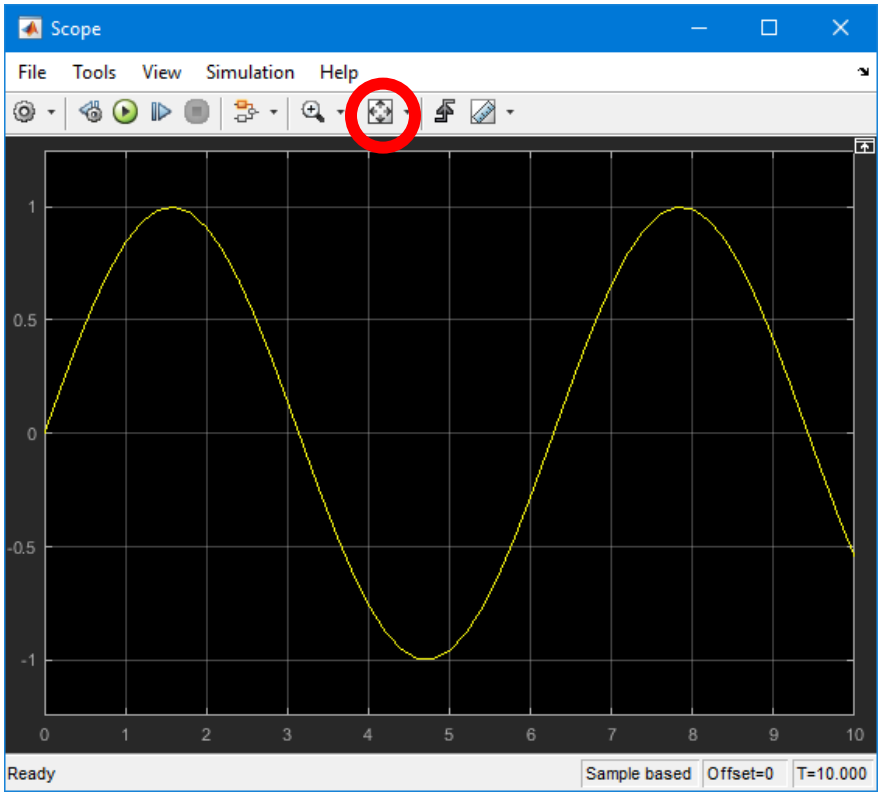
16. ábra – Display blokk

Scope

A szimuláció vagy külső módban történő futás során a jelek időfüggvénye is megjeleníthető egy `Scope` blokk (`Simulink > Sinks > Scope`) segítségével. Ezek a grafikus megjelenítők memóriával is rendelkeznek, így az értékek a szimuláció leállítása után is megtekinthetők.

17. ábra – Scope blokk

A blokkra duplán kattintva az 18. ábrán láthatóhoz hasonló ablak jelenik meg. A fejléc gombjainak segítségével a diagram nagyítható, a teljes ablakot kitöltő skálázásra (autoscale) a pirossal jelölt ikon szolgál. A görbe a `File > Copy To Clipboard` menüparanccsal fehér háttérrel a vágólapra másolható.

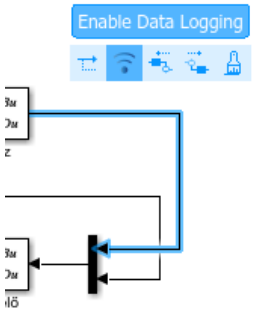


18. ábra – Scope blokk által megjelenített görbe

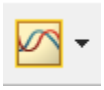
Jelek rögzítése

A szimuláció illetve egyes targetek esetén (sajnos a mérés során használt eszköz nem tartozik ezek közé) a külső módban történő futtatás során a jelek alakulását a *Simulation Data Inspector* segítségével is megjeleníthetjük. Ennek használatához kattintsunk a megfigyelni kívánt jeleket továbbító vezetékekre, majd vigyük a kurzort vezeték fölött megjelenő, három pontot ábrázoló ikon fölé. Az ennek hatására láthatóvá váló ikonsor (19. ábra) második elemét kiválasztva adhatjuk hozzá a választott jelet a rögzítettek közé. Érdemes a megfigyelt jeleket elnevezni (a vezetéken kétszer kattintva), így a *Simulation Data Inspector* ablakában megfelelően címkézve láthatjuk a jelalakokat.

A szimuláció végén a rögzített jeleket a *Simulation Data Inspector* segítségével jeleníthetjük meg a modell ikonsorának 20. ábrán látható elemére kattintva. A *Simulation Data Inspector* ablakának (21. ábra) bal oldali panelján választható ki, hogy mely futás eredményei jelenjenek meg: alapértelmezésben minden szimulációs futás egy új adatsort hoz létre, így a különböző paraméterek mellett kapott eredmények kényelmesen összehasonlíthatók. Amennyiben egy futásra jobb gombbal kattintunk és kijelöljük az `Overwrite Run` pontot, az előző eredmények nem őrződnek meg, az újabb futások jelei felülírják az előzőeket.



19. ábra – Jelek rögzítésének beállítása

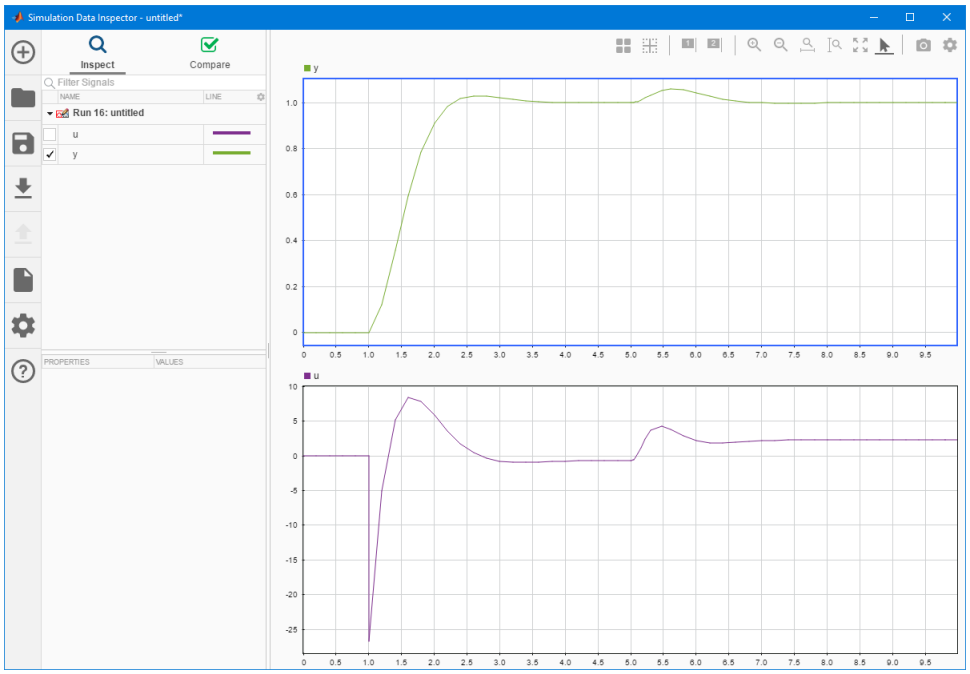


20. ábra - A Simulation Data Inspector ikonja

Egy futás jelei közül a jelölőnégyzetekkel kiválasztottak kerülnek ábrázolásra. A fejléc ikonjai segítségével a grafikus ablakot több részre is oszthatjuk, így az egyes jelek más-más ábrákon is megjeleníthetők (ez különösen hasznos, ha értékük nagyságrendekben eltér).

Külső módban történő futtatás esetén a *Simulation Data Inspector*-ban megjelenített görbék folyamatosan frissülnek, a `Scope` blokkokkal ellentétben memóriájuk sem korlátozott 10 másodpercre.

A görbék a *Simulation Data Inspector* ikonsorának fényképezőgép-ikonjával a vágólapra vagy képfájlba menthetők.



21. ábra – A Simulation Data Inspector ablaka

2.3 Mérési feladatok

Hozzon létre egy új Simulink-modellt és abban készítse el a járműmodellt megvalósító blokkdiagramot!

- a jármű sebességét a **gyorsulás integrálásával** képezze: $\dot{v} = \frac{1}{m}(F_M - F_D - mg\sin\alpha)$
- a sebességfüggő maximális gyorsító erő leírásához **használjon lookup-táblát** (`1-D lookup table`), a mintapontok között köbös interpolációt (*cubic*), azokon kívül pedig vágás típusú extrapolációt (*clip*) alkalmazva

Ellenőrizze a modell működését konstans 1 pedálállás és 0 lejtőszög mellett!

- növelje meg az alapértelmezett 10 másodperces szimulációs időtartamot 100 másodpercre
- rögzítse (a *Simulation Data Inspector* vagy egy `Scope` blokk segítségével) a jármű sebességének alakulását

3 Sebességtartó automatika tervezése és hangolása

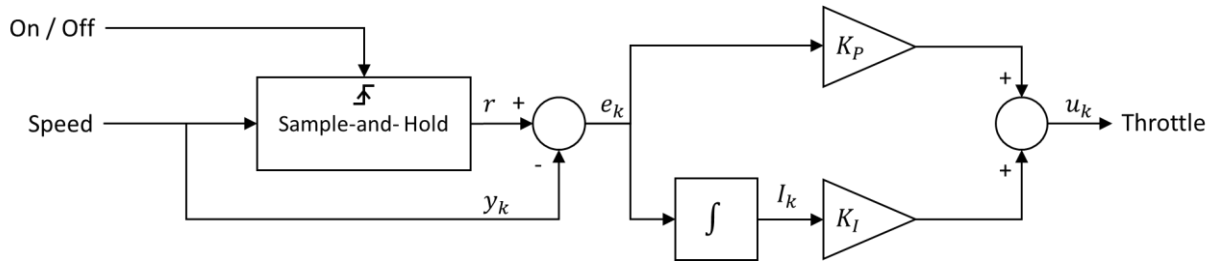
A sebességtartó automatika (tempomat) feladata, hogy a zavaró hatások (lejtő, megváltozott tömeg vagy légellenállás) ellenére is tartsa azt a sebességet, mellyel a jármű a tempomat bekapcsolásának pillanatában haladt.

A tempomat bemenetei tehát a befolyásolni kívánt szabályozott jellemző (az aktuális sebesség, y_k), illetve a sebességtartás be- illetve kikapcsolt állapota (o_k, melynek értéke 1 bekapcsolt, 0 pedig kikapcsolt tempomat esetén). A sebességtartó automatika kimenete a beavatkozó jel (u_k), ami esetünkben a „pedálállás" 0 és 1 közti értéke.

A megvalósítandó algoritmusnak kikapcsolt állapotban ($o_k = 0$) zérus beavatkozó jelet kell kiadnia ($u_k = 0$). Bekapcsolás esetén (az állapot $0 \rightarrow 1$ váltása, $o_k = 1$ és $o_{k-1} = 0$) az aktuális sebességet el kell menteni ($r = y_k$), és a működés során azt kell alapjelként használni. A sebességtartásért egy mintavételes szabályozó felelős, mely az u_k beavatkozó jelet az $e_k = r - y_k$ hibajel alapján állítja elő.

3.1 A szabályozó algoritmusa

Szabályozóként PI szabályozót használunk, mely az integráló hatásnak köszönhetően képes konstans alapjel maradó hiba nélküli követését biztosítani, még zavarás jelenléte esetén is. Ennek, valamint egyszerű felépítésének köszönhetően számos esetben használhatjuk akár nemlineáris rendszerek irányítására is – természetesen szem előtt tartva az egyszerűséggel járó korlátokat is. A sebességtartó automatika blokkvázlatát a 22. ábra mutatja be.



22. ábra - A sebességtartó automatika blokkvázlata

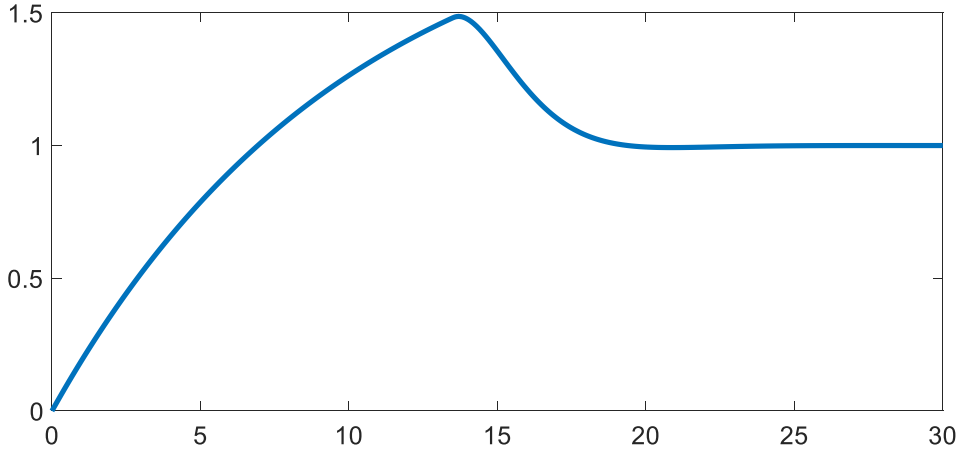
A szabályozót beágyazott környezetben természetesen mintavételesen valósítjuk meg. A PI szabályozó differenciaegyenlete

$$u_k = K_P e_k + K_I I_k,$$

ahol a hiba I_k integrálját téglalapszabály szerint számíthatjuk (T_s a mintavételi idő):

$$I_k = I_{k-1} + T_s e_k.$$

Ha azonban a beavatkozó szerv telítéses karakterisztikával rendelkezik (esetünkben is ez a helyzet, a motor teljesítménye nem lehet tetszőlegesen nagy, a pedálállás értéke 0 és 1 közé eshet), az integrátor használata problémát okozhat. Ilyen esetekben ugyanis, különösen zavarás jelenlétében, a hiba a zárt körben lassan csökken: ekkor az integrátor értéke gyorsan növekszik, aminek hatására a beavatkozó jel egy ideig még negatív hibajel esetén is pozitív lesz, ami jelentős túllövést is okozhat. Ezt a jelenséget nevezzük elintegrálódásnak (23. ábra).



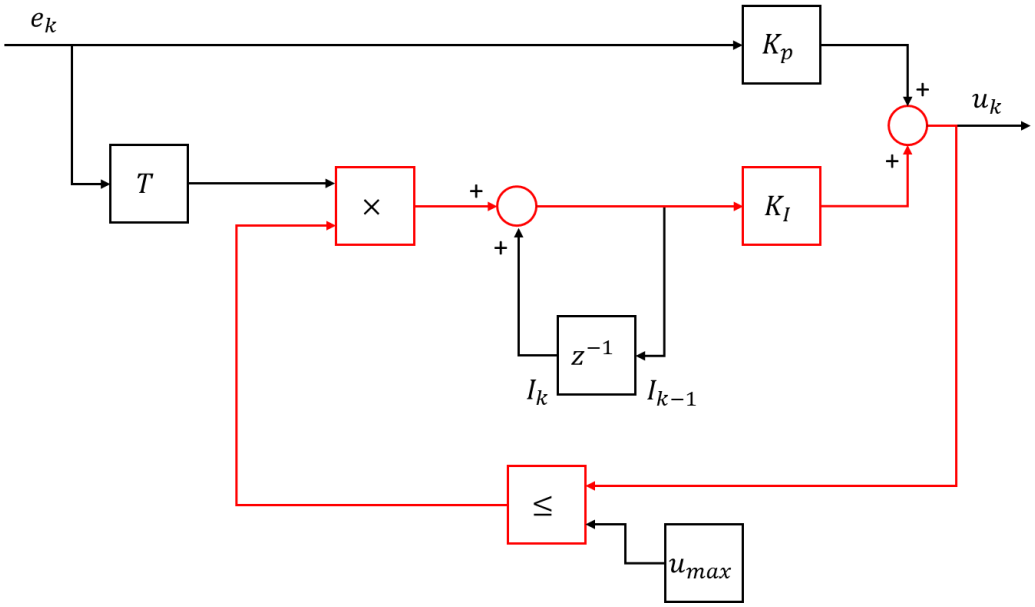
23. ábra – Zárt kör ugrásválasza elintegrálódás esetén

Az elintegrálódás kiküszöbölésének legegyszerűbb, a gyakorlatban előszeretettel alkalmazott módja, ha az integrálást „kikapcsoljuk”, amennyiben a beavatkozó jel eléri a telítés határát:

$$u_k = K_P e_k + K_I I_k$$

$$I_k = I_{k-1} + \begin{cases} T_s e_k & \text{ha } |u_k| \geq u_{min} \text{ és } |u_k| \leq u_{max} \\ 0 & \text{egyébként} \end{cases}$$

Könnyen belátható azonban, hogy a fenti kifejezés közvetlenül nem valósítható meg, hiszen I_k számításához ismerni kell u_k-t, amihez viszont I_k ismeretére is szükség van, azaz algebrai hurokhoz jutunk, ahogy az a fenti kifejezésnek megfelelő blokkdiagramon (24. ábra, a blokkdiagramon csak felső korlát szerepel) is látható.



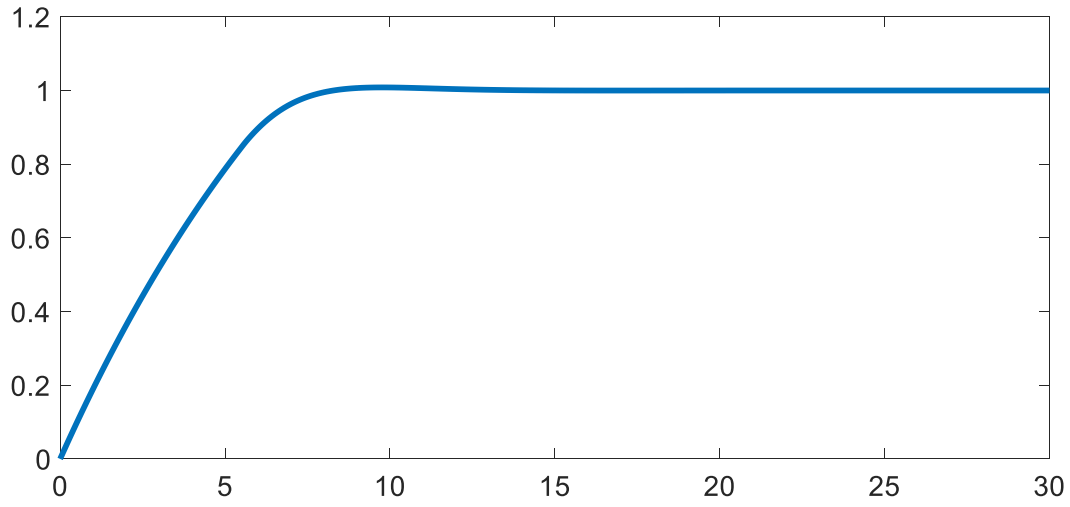
24. ábra - Az elintegrálódás kezelése során adódó algebrai hurok

Szerencsére ezt a hurkot könnyen feloldhatjuk, ha a beavatkozó jel számítását a következők szerint végezzük:

$$u_k = K_p e_k + K_I I_{k-1} + K_I T_s e_k$$

$$I_k = I_{k-1} + \begin{cases} T_s e_k & \text{ha } |u_k| \geq u_{min} \text{ és } |u_k| \leq u_{max} \\ 0 & \text{egyébként} \end{cases}$$

Ekkor a beavatkozó jel értéke már nem függ az integrator aktuális értékétől, így algebrai hurok nem alakul ki. Az elintegrálódást a fentiek szerint kezelve az ugrásválasz már a 25. ábrán látható módon alakul.



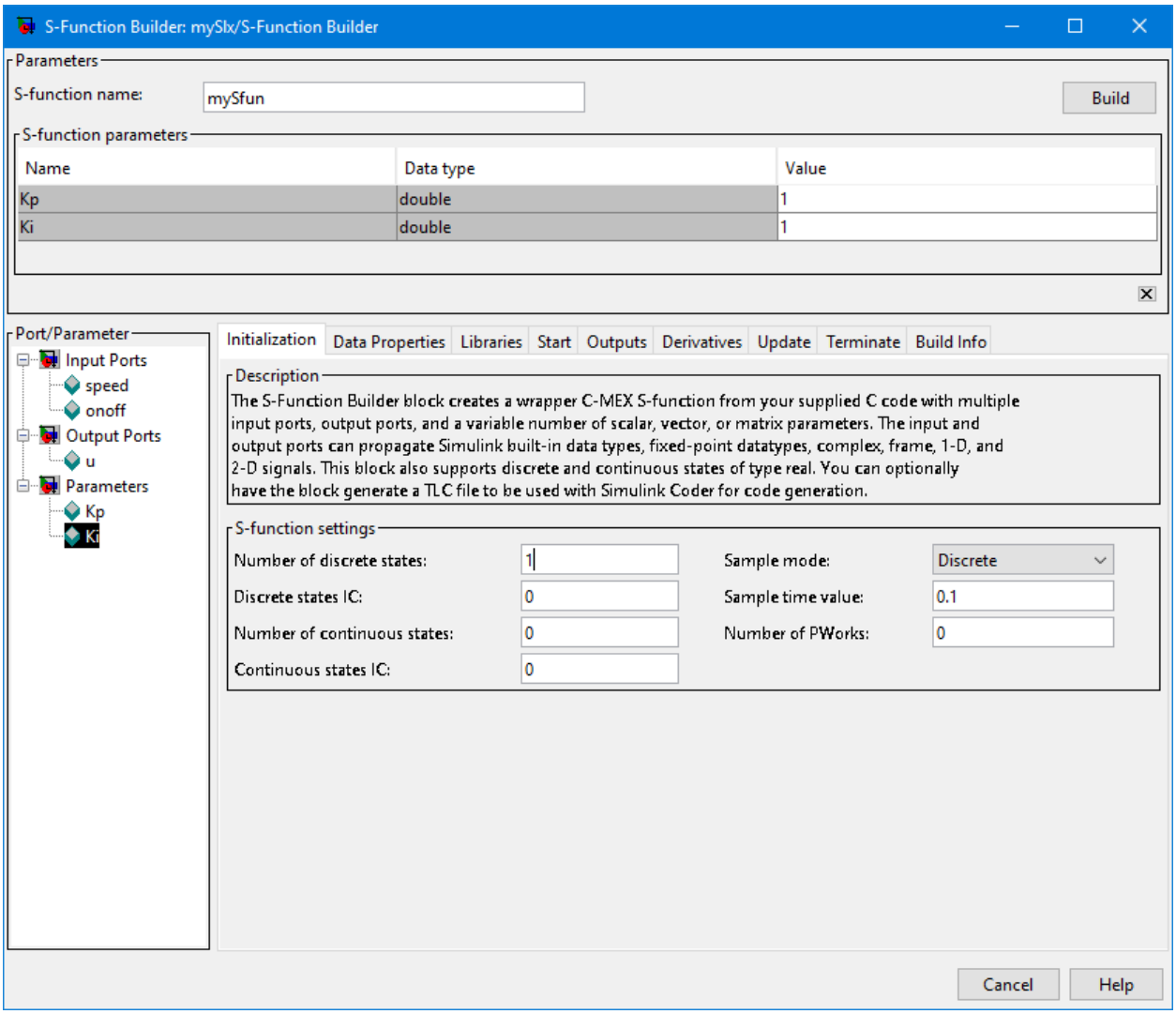
25. ábra - A zárt kör ugrásválasza az elintegrálódás egyszerű kezelésével

Szintén problémát okozhat, ha a szabályozó nem folyamatosan üzemel, hanem szakaszosan kapcsoljuk be illetve ki, mint ahogy ez egy tempomatra is jellemző. Nem kellően gondos megvalósítás esetén előfordulhat, hogy az integrátor kikapcsolt szabályozó mellett is integrálja a hibát, vagy éppen megőrzi a bekapcsolt állapotban felvett utolsó értékét. Ilyenkor a szabályozó újbóli bekapcsolásakor az integrátor értéke jelentős ugrást, sőt, rövid ideig a megkívánttal ellentétes előjelet eredményez a beavatkozó jelben. Mindenképpen szükséges tehát biztosítani azt, hogy az integrátor értéke a szabályozó kikapcsolt állapotában zérus értéket vegyen fel (alternatív megoldásként az integrátor értékét nullára állíthatjuk a szabályozó bekapcsolásakor is).

3.2 A szabályozó megvalósítása S-függvényként

A szabályozó a fenti kiegészítésekkel együtt is egy mintavételes rendszer, melynek egyetlen állapotváltozója az integrátor értéke, bemenetei pedig az aktuális sebesség (y_k) és a tempomat be- vagy kikapcsolt állapota (o_k), kimenete pedig a 0 és 1 közé korlátozott beavatkozó jel ($u_k \in [0,1]$).

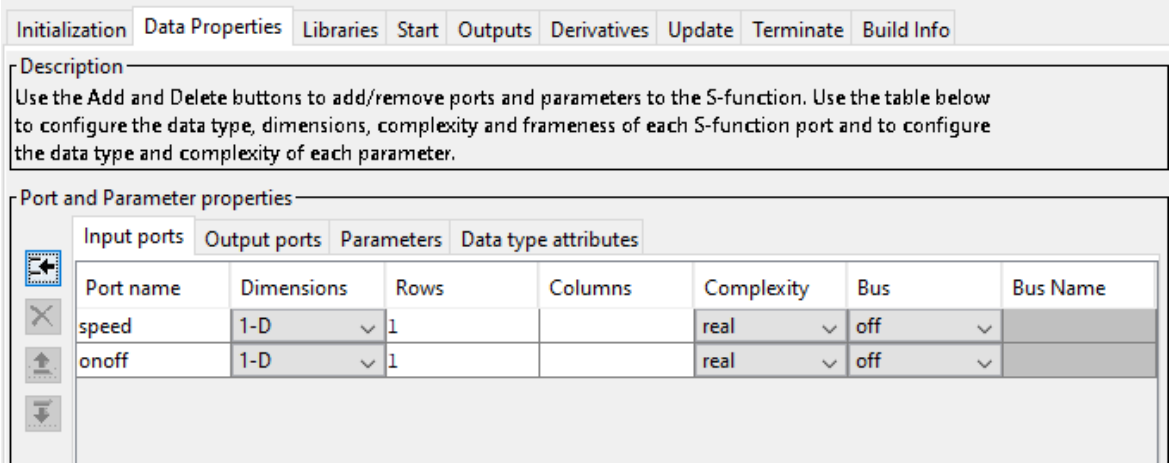
A szabályozót C nyelvű S-függvényként valósítjuk meg az S-Function Builder segítségével. Ehhez egy `S-Function Builder` blokkot (`Simulink > User-Defined Subsystems > S-Function Builder`) kell elhelyeznünk a blokkdiagramban. A blokkra kétszer kattintva megjelenik annak kofigurációs ablaka (26. ábra). Ennek legfelső, *S-function name* beviteli mezőjében kell megadnunk az s-függvény nevét, ennek hiányában azt nem fordíthatjuk le.



26. ábra - Az S-Function Builder ablaka

A blokk alapvető paramétereit (mintavételezés, be- és kimenetek, valamint állapotok száma) az *Initialization* fülön állíthatjuk be. Mivel diszkrét szabályozót szeretnénk megvalósítsani, a *Sample mode* paramétert *Discrete*-re, míg a *Sample time* értékét `0.1`-re (100ms) állítsuk be. A szabályozó egyetlen (diszkrétidejű) állapotváltozója az integrátor állapota, így a *Number of Discrete States* paraméternek `1` értékét adunk. Az integrátor kezdeti állapotát (*Discrete states IC*) hagyjuk zérus értéken, és mivel nincsenek folytonos idejű állapotaink, a *Number of continuous states* és *Continuous states IC* paraméterekkel is tegyünk így.

A blokk be- és kimeneteinek számát, típusát és elnevezését, valamint a konstans paraméterek elnevezését és értékét a *Data Properties* fülön állíthatjuk be (27. ábra). Ezen belül is találunk további alfüleket: az *Input*, *Output* és *Parameters* alfülek értelemszerűen a be- és kimenetekre, valamint a paraméterekre vonatkozó beállítások, a *Data type attributes* alfülön pedig az ezek adattípusainak részletes konfigurációjára van lehetőség. A be- és kimeneteknél a táblázat melletti négy ikonnal sorban új be- vagy kimenetet adhatunk hozzá, a másodikkal törölhetjük a kijelöltet, a harmadik és negyedik gombbal pedig átrendezhetjük a portok sorrendjét.



27. ábra - Az S-Function Builder Data Properties füle

A be-és kimeneteknek a *Port name* mezőben tetszőleges, a C nyelv azonosítóformátumának megfelelő elnevezést adhatunk. A be- és kimenetek mindegyike lehet vektor is, így adattípusuk tömb (alapértelmezésben egydimenziós, mérete a *Rows* illetve *Columns* mezőkben adható meg), még skalár értékű be- és kimenetek esetén is. Ennek megfelelően a megvalósítandó függvényben ezekre minden esetben indexelten, pl. a skalárértékű `in1` bemenetre `in1[0]` módon kell hivatkozni.

A *Parameters* fülön beállítható paraméterek valójában konstansok, melyek elnevezése a *Parameter name*, adattípusa pedig a legördülő *Data type* mezőben adható meg. Az itt definiált paraméterekhez értékek az *S-Function Builder* főablakában, az s-függvény neve alatt megjelenő táblázatban rendelhetők. A megvalósított szabályozási algoritmus K_p és K_i paramétereit érdemes az s-függvényben is paraméterként kezelni.

Mint ismeretes, a szimuláció során a Simulink az s-függvények meghatározott metódusait hívja, melyeket a megfelelő füleken implementálhatunk. Ezek a metódusok a következők:

- *Outputs* – a blokk kimenetének számításakor hívott metódus
- *Update* / *Discrete Update* – diszkrétidejű blokk esetén a mintavételi időpontokban hívott metódus (pontos megnevezése verziófüggő)
- *Derivatives* – folytonosidejű blokk esetén az integrálási lépésenként hívott metódus

Az egyszerű mintavételes PI-szabályozó megvalósítása esetén elegendő az *Update (Discrete Update)* metódust implementálnunk, melyet a Simulink a beállított mintavételi időnek megfelelő ütemezéssel hív meg.

Az egyes metódusokon belül tetszőleges változókat deklarálhatunk, azonban ezek lokálisak, más metódusokból nem elérhetőek. Amennyiben szeretnénk, hogy egy változó értéke a metódus hívásai között megőrződjön (pl. alapjel értéke), akkor azt a `static` kulcsszó segítségével kell deklarálnunk. A be- és kimenetek, valamint a rendszer egyetlen diszkrétidejű állapota (melyre `xD[0]`-ként hivatkozhatunk) globális változók, így például az *Updates* metódusban is adható érték a rendszer kimeneteknek.

Az s-függvény az *S-Function Builder* ablakának jobb felső sarkában elhelyezkedő *Build* gombra kattintva fordítható le, a fordítás folyamata és eredménye (az esetleges hibaüzenetekkel együtt) a *Build Info* fülön jelenik meg.

3.3 Mérési feladatok

Implementáljon C nyelvű s-függvényként egy mintavételes PI-szabályozót, mely biztosítja a bekapcsoláskor érvényes sebesség tartását!

- a szabályozót mintavételes rendszerként valósítsa meg $\boldsymbol{T_s = 0.1\mathrm{s}}$ mintavételi idő mellett
- a blokk **bemenet**ei legyenek az aktuális sebesség és a tempomat állapota (be/ki)
- a blokk **kimenet**e legyen a 0 és 1 közé korlátozott beavatkozó jel
- a szabályozó K_p és K_i paramétereit az s-függvény **paraméter**eiként kezelje
- az integrátor állapota a szabályozó egyetlen **diszkrétidejű állapotváltozó**ja (`xD[0]`) legyen
- a szabályozó algoritmusát az *Updates* metódusban valósítsa meg, az elintegrálódás kezelését biztosító algoritmus szerint:

$$u_k = K_p e_k + K_I I_{k-1} + K_I T_s e_k$$

$$I_k = I_{k-1} + \begin{cases} T_s e_k & \text{ha } |u_k| \geq u_{min} \text{ és } |u_k| \leq u_{max} \\ 0 & \text{egyébként} \end{cases}$$

- a fenti számítások után a **beavatkozó jelet korlátozza 0 és 1 közé**, valamint biztosítsa, hogy **nullázza az integrátor állapotváltozóját**, amennyiben a tempomat ki van kapcsolva
- az **alapjel**et a tempomat bekapcsolásakor (az bekapcsolt állapotot jelző bemenet felfutó élére) állítsa be, a jel előző állapotát egy **`static`** minősítésű változóban tárolhatja

Kapcsolja össze a járműmodellt az előzőekben implementált szabályozóval!

- **kapcsolja össze** a járműmodell sebesség-kimenetét a szabályozó sebesség-bemenetével, a járműmodell pedálállás-bemenetét pedig a szabályozó beavatkozó jel kimenetével
- a tempomat **bekapcsoló jel**ére kössön **konstans 1** értéket.
- állítsa be a járműmodell **kezdeti sebesség**ét (az integrator kezdeti állapotát) az autópályán megengedett sebességnek megfelelő 35 m/s-os értékre.
- a **lejtőszög-bemenetre kössön egy jelgenerátort** (`Signal Builder`), ami 20 másodpercig 0 lejtőszöget, majd 40 másodpercig 5 fokos lejtőszöget, végül újabb 40 másodpercig 0 lejtőszöget biztosít (ne feledje, hogy a Simulink trigonometrikus blokkjai **radián** egységet használnak!)

A modellt szimulálva hangolja be empirikus úton a szabályozó paramétereit úgy, hogy az képes legyen a beállított sebesség tartására!

- jól beállított szabályozó esetén bekapcsoláskor, illetve a lejtőszög változásakor **rövid ideig a kívántnál alacsonyabb sebesség megengedett**, de a maradó hibának zérusnak kell lennie
- ügyelni kell a jelentős túllövések, különösen pedig az **oszcilláció elkerülésére**
- a hangolás során a **beavatkozó jelet is figyelje meg**, és biztosítsa, hogy abban se jelenjenek meg nagy lengések, illetve hirtelen változások

4 A sebességtartó automatika implementálása beágyazott platformon

Az előzőekben megtervezett és behangolt szabályozó algoritmus a valóságban egy beágyazott processzoron fut, környezetével fizikai jeleken keresztül tartja a kapcsolatot, a mikrokontrolleren az irányítási algoritmus mellett az IO-kezelést is meg kell valósítani. Példánkban a tempomatvezérlő közvetlenül a kerekeken elhelyezett ABS-jeladó jeleit fogadja, a motorvezérlő elektronikával (ECU) pedig digitális buszon keresztül kommunikál (ezt a mérés során egyszerű UART kommunikációval valósítjuk meg). A tempomat be- illetve kikapcsolt állapotát megadó jelet (akár egy fizikai kapcsoló állását) közvetlenül a mikrokontroller egy digitális bemenetéhez csatlakoztatjuk.

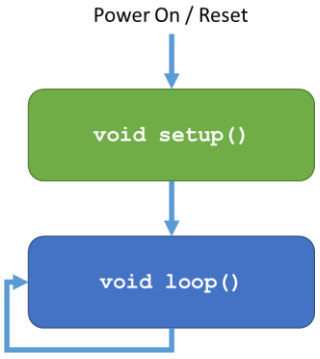
A tempomatvezérlőt a mérés során egy Arduino Mega 2560-as fejlesztőkártyán implementáljuk, az alsóbb szintű hardverillesztést megvalósító függvények az `iit1.h` fájlban már rendelkezésre állnak, így azokat nem kell megvalósítani. A következőkben az Arduino alkalmazások felépítését, a hardveres interfész- és ütemező függvényeket, valamint a fejlesztőkörnyezet legfontosabb funkcióit tekintjük át.

4.1 Arduino-alkalmazások felépítése

Az Arduino IDE a C és C++ nyelvet támogatja néhány speciális, a kód struktúráját meghatározó szabály és számos „lazítás” mellett. Ez utóbbiak közé tartozik például, hogy nem kell minden esetben mereven ragaszkodni az adattípusokhoz: ha például egy beépített függvény `int` típusú paramétert vár, meghívhatjuk akár `long` vagy `double` típusú változóval is. Ez sokszor előnyös, azonban a helyes gyakorlat mindig a megfelelő típusú változók használata.

Változók és függvények a C/C++ nyelv szintaktikájának megfelelően deklarálhatók. Globális változók esetén a `volatile`, míg függvényeken belül retentív változók deklarálása esetén a `static` kulcsszó alkalmazhat.

Minden sketch (Arduino-alkalmazás) tartalmaz két különleges függvényt. A `void setup()` függvény a program indulásakor (a kontroller bekapcsolásakor vagy újraindításakor) fut le egyetlen egyszer, ezután a `void loop()` függvény fut ciklikusan, mint ha a függvénytörzset egy végtelen ciklusba helyeznénk (26. ábra). Ennek megfelelően nincs szükség (és lehetőség) *main* függvény használatára, annak funkciói a `setup()` és `loop()` függvények között oszlanak meg. Mind a `setup()`, mind a `loop()` függvény hívhat más függvényeket.



28. ábra – Arduino-alkalmazások szerkezete

Ezt az egyszerű működési modellt megszakításkezelő rutinok használatával egészíthetjük ki, melyek egyes hardveresemények (pl. bemenet felfutó éle) hatására vagy éppen adott periódusidővel ütemezve kerülnek meghívásra. Ez utóbbi lehetőséget használjuk majd arra, hogy a szabályozót megvalósító algoritmust pontosan 0.1s-os periódusidővel hajtsuk végre.

4.2 Interfész- és ütemező függvények

A mérés során rendelkezésre álló, a hardveres interfészek kezelését végző, illetve ütemező függvények az alábbiak.

```
void initializeHAL(void)
```

Az `initializeHAL()` függvény beállítja a szükséges hardveres vonalak irányát, megnyitja a soros kapcsolatot és elindítja a sebességmérést. Ezt egyszer, a program indításakor kell meghívni a `setup()` függvényen belül.

```
void scheduleController(void (*fcn))
```

A `scheduleController()` függvény biztosítja, hogy a szabályozási algoritmust megvalósító függvény 0.1s-os mintavételi idővel, periodikusan kerüljön végrehajtásra. Szintén egyszer szükséges hívni a program indításakor, paraméterként az ütemezni kívánt függvény azonosítóját átadva. Ha a mintavételes szabályozót pl. a `void tempomat(void)` függvényben valósítjuk meg, akkor az ütemező függvényt `scheduleController(tempomat)` módon kel hívnunk.

```
bool getTempomatState(void)
```

A `getTempomatState()` függvény a tempomat állapotának lekérdezésére szolgál, visszatérési értéke igaz, ha az be van kapcsolva.

```
double getSpeed(void)
```

A `getSpeed()` függvény a jármű aktuális, az ABS-jeladó impulzusainak periódusideje alapján számított sebességével tér vissza. A visszatérési érték közvetlenül m/s-ban adott, így további átalakításra nincsen szükség.

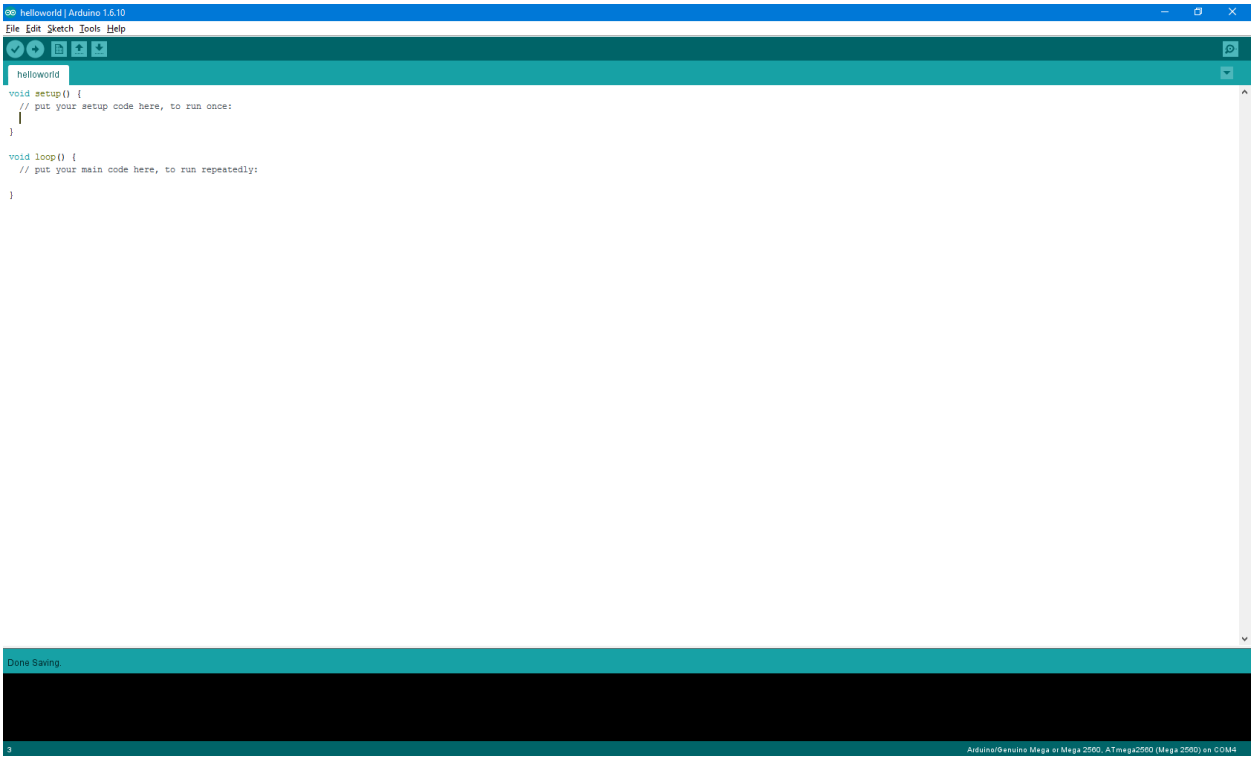
```
bool sendThrottle(double throttle)
```

A kiszámított beavatkozó jel (pedálállás) ECU felé történő továbbítására a `sendThrottle()` függvény használható, melynek egy 0 és 1 közé eső lebegőpontos kell átadni. A függvény visszatérési értéke hamis, ha az átadott érték nem esik a megfelelő tartományba, egyéb esetben igaz. A beavatkozó jel (a gyakorlatban megszokott módon) nem lebegőpontos értékként, hanem kvantálva (esetünkben a 0-255 tartományra skálázva) egészként kerül továbbításra.

4.3 Arduino IDE

Az Arduino IDE egy Java-alapú, cross-platform környezet, mely a szerkesztő mellett GNU-alapú fordító eszközöket is tartalmaz

A fejlesztői környezet ablakát a 29. ábra mutatja be. Az ablak legnagyobb részét a szerkesztőfelület foglalja el, mely természetesen kontextustól függő színezést használ, a sketch különböző fájljait füleken jeleníti meg. A szerkesztő alatti, zöld hátterű sáv a legfontosabb státuszüzenetet tartalmazza, míg a részletek a fekete hátterű sávban jelennek meg.



29. ábra – Az Arduino IDE ablaka

Az ablak felső ikonsorával balról jobbra a következő funkciók érhetők el:

- *Verify*: sketch fordítása
- *Upload*: sketch fordítása és feltöltése
- *New*: új sketch létrehozása
- *Open*: sketch megnyitása
- *Save*: sketch mentése

A fejlesztés megkezdése előtt ki kell választanunk, hogy milyen platformot szeretnénk használni. Ezt a `Tools > Board` menüpontban tehetjük meg, válasszuk az `Arduino / Genuino Mega or Mega 2560` opciót. A `Tools > Processor` menüpontban a többféle mikrokontrollerrel is szerelt kártyák esetén választható ki a processzor típusa, esetünkben ez az alapértelmezett `Mega 2560` modell. A programozásra használt soros portot a `Tools > Port` menüpontban kell kiválasztani, mely a valódi és virtuális soros portok listáját ajánlja fel. Amennyiben ezek közül valamelyikre támogatott eszköz csatlakozik, a COM port azonosítója mellett zárójelben annak típusa is megjelenik. A kapcsolat tesztelésére használható a `Tools > Get Board Info` menüpont, ami a választott soros portra csatlakoztatott kártya paramétereit kérdezi le.

Az elkészült program fordítása az ikonsor első ikonjával, a `Sketch > Verify/Compile` menüponttal vagy a `Ctrt+R` billentyűparanccsal indítható. Ilyenkor csak a fordítás történik meg, az esetleges hibaüzenetek a fekete hátterű értesítési sávban jelennek meg.

A program az ikonsor második ikonjával, a `Sketch > Upload` menüponttal vagy a `Ctrl+U` billentyűkombinációval tölthető le a beágyazott processzorra. A letöltést mindenképpen megelőzi egy

fordítási fázis, még akkor is, ha az utolsó fordítás óta nem módosítottuk a programot (ezért érdemes közvetlenül a feltöltést választani, nem szükséges előtte külön fordítani a programot). Sikeres fordítás után a futtatható kód letöltésre kerül a mikrokontrollerre (erről, valamint a programunk által felhasznált erőforrásokról az értesítési sávban kapunk tájékoztatást), és rögtön el is indul.

A fejlesztést és a hibakeresést segítendő az Arduino IDE rendelkezik beépített soros terminállal is. Ezt a `Tools > Serial Monitor` menüpontban, illetve a `Ctrl+Shift+M` billentyűparanccsal érhetjük el.

A PC-vel történő üzenetcsere az Arduino-kódban a `Serial` objektum segítségével lehetséges, melynek használatához a `setup()` függvényben `Serial.begin(9600)` paraméterezéssel meg kell hívnunk a megfelelő metódust (9600 baud-os sebesség és a 8N1 konfiguráció használata, ami megegyezik az Arduino IDE soros monitorának alapértelmezett paramétereivel). A soros kommunikáció inicializálása után a program tetszőleges függvényéből küldhetünk üzenetet a `Serial.print()` illetve `Serial.println()` metódusok segítségével. Ezek a logikai, numerikus illetve karakteres típusú data értéket továbbítanak a soros porton, mégpedig ASCII-karakterlánc formájában. A `Serial.print(FALSE)` metódus a „0” stringet, a `Serial.print(78)` a „78” karaktersorozatot továbbítja (2 bájton), mely a soros monitor ablakában közvetlenül kijelzésre kerül. A `println()` metódus csupán abban különbözik a `print()`-től, hogy az adatsztring után még kocsi vissza és új sor karaktert (CR LF) is fűz, így az üzenetek áttekinthetőbb formában jelennek meg.

4.4 Mérési feladatok

Valósítsa meg az előzőekben megtervezett és behangolt szabályozót Arduino platformon!

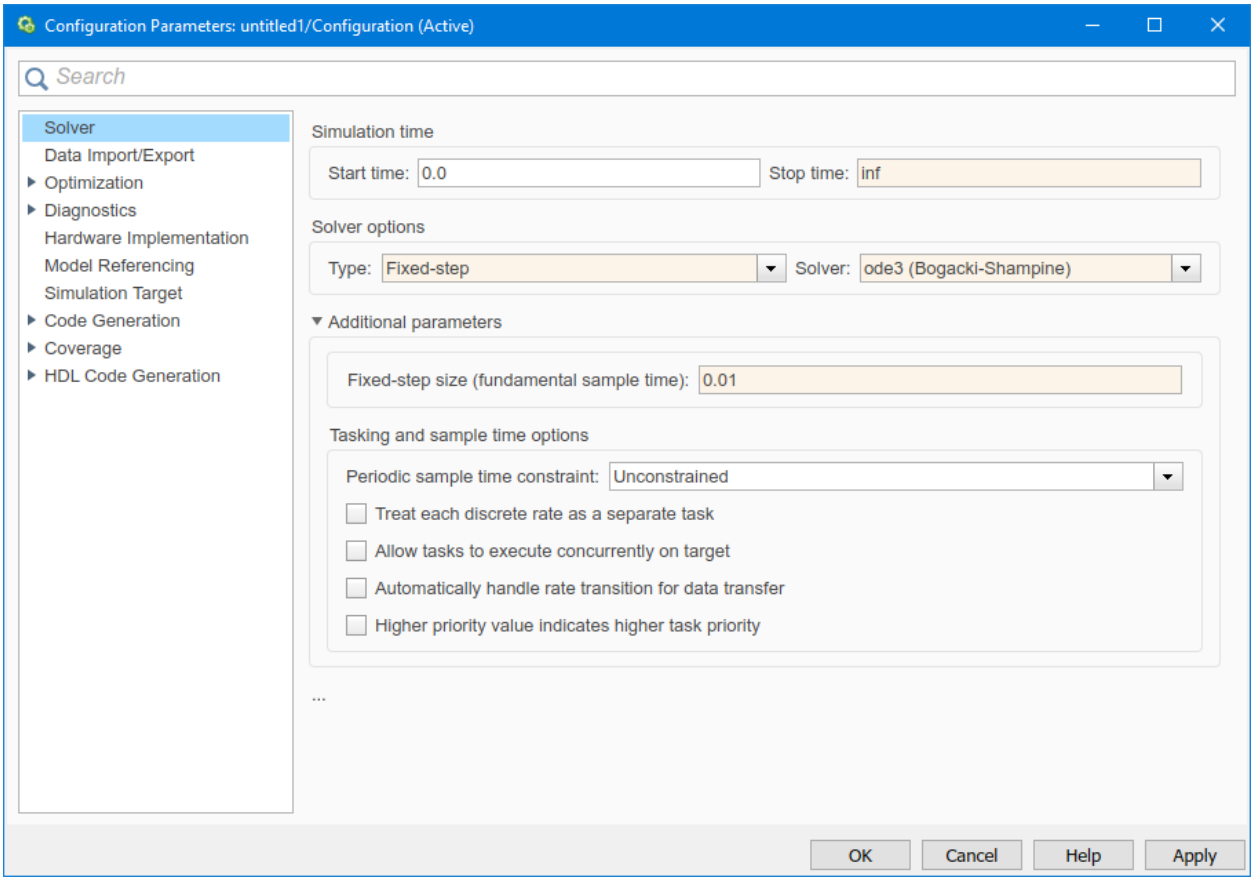
- a hardverkezelő és ütemező függvények használatához másolja az **`iit1.h`** fájlt a sketch könyvtárába (természetesen ne feledkezzen meg az **`#include "iit1.h"`** direktíváról sem)
- a szabályozót egy **különálló függvényben** implementálja, a megvalósítás során használja a rendelkezésre álló `getSpeed()`, `getTempomatState` és `sendThrottle()` függvényeket
- ne feledkezzen el a `setup()` függvényben a **hardverkezelő rutinok inicializálásáról** és a **szabályozót futtató algoritmus ütemezéséről** (mivel az algoritmust az ütemezetten futó függvény valósítja meg, a `loop()` függvény törzse üresen marad)

5 A járműmodell futtatása külső módban

Ahhoz, hogy a beágyazott platformon implementált tempomatvezérlő megfelelő működéséről meggyőződhessünk, Hardware-In-the-Loop (HIL) tesztet használhatunk. Ennek során a járműmodellt szintén egy beágyazott, megfelelő hardveres interfészekkel rendelkező targeten (a mérés során egy másik Arduino Mega 2560-as fejlesztőpanelen) futtatjuk, melyet közvetlenül csatlakoztatunk az irányítási algoritmust futtató kártyához. A következőkben áttekintjük, hogyan kell előkészítenünk a járműmodellt a külső módban való futtatásra, illetve milyen kiegészítésekkel kell élnünk a megfelelő teszteléshez.

5.1 Modell-beállítások

A külső módban történő futtatáshoz megfelelő integrálási algoritmust (ODE solver), mind pedig megfelelő hardveres implementációs beállításokat kell választani. A modell konfigurációs beállításait a Simulink ikonsorának fogaskerék-ikonjával, a `Simulation > Model Configuration Parameters` menüparanccsal illetve a `Ctrl-E` billentyűzetkombinációval megnyitható párbeszédablakban állíthatjuk be.



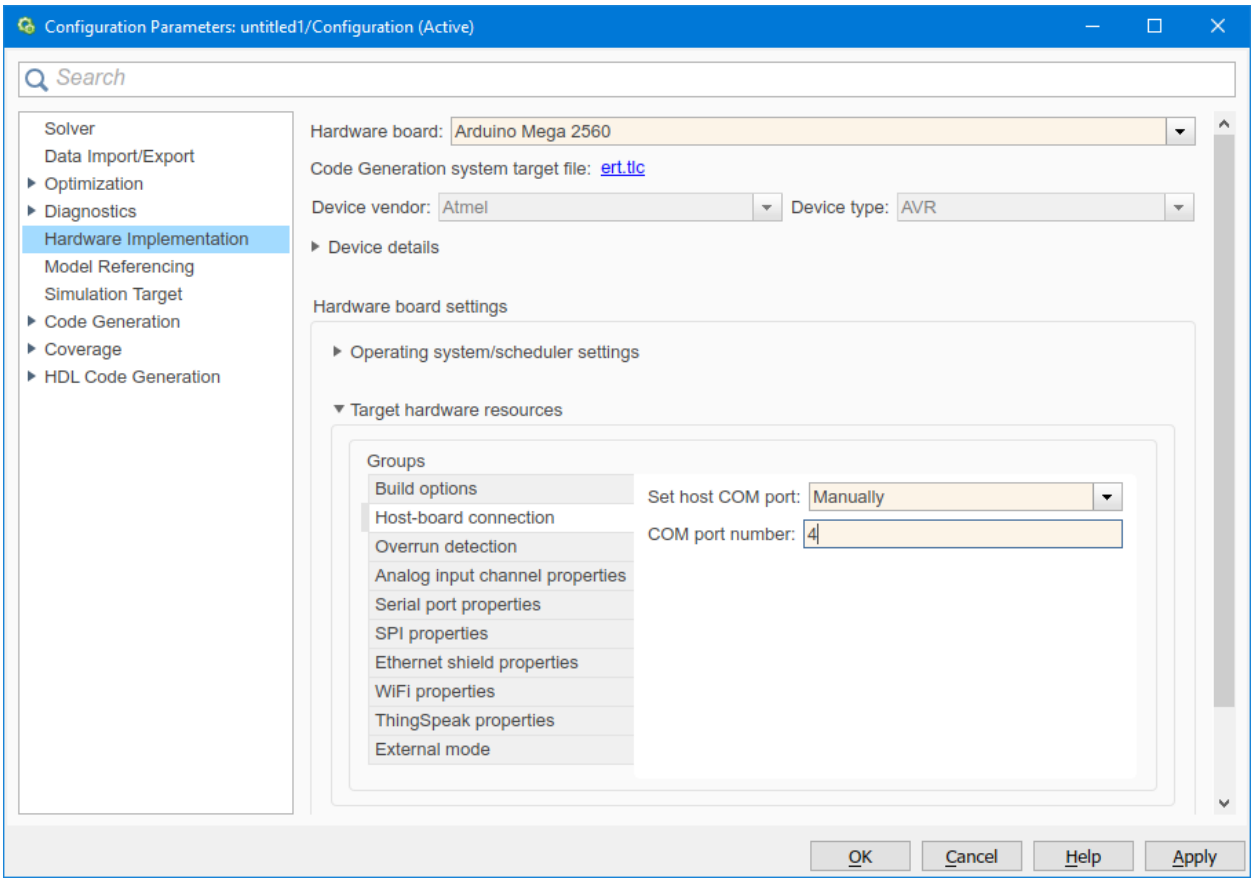
30. ábra – Az integrálási algoritmus beállítása

Mivel a beágyazott processzoron a modellnek valósidőben kell futnia, ezért csak fix lépésközű integrálási algoritmus választható. Ezért a párbeszédablak *Solver* paneljén (30. ábra) válasszuk a fix lépésközű algoritmusok közül az `ode3` néven elérhető Bogacki-Shampine módszert (harmadrendű, explicit Runge-

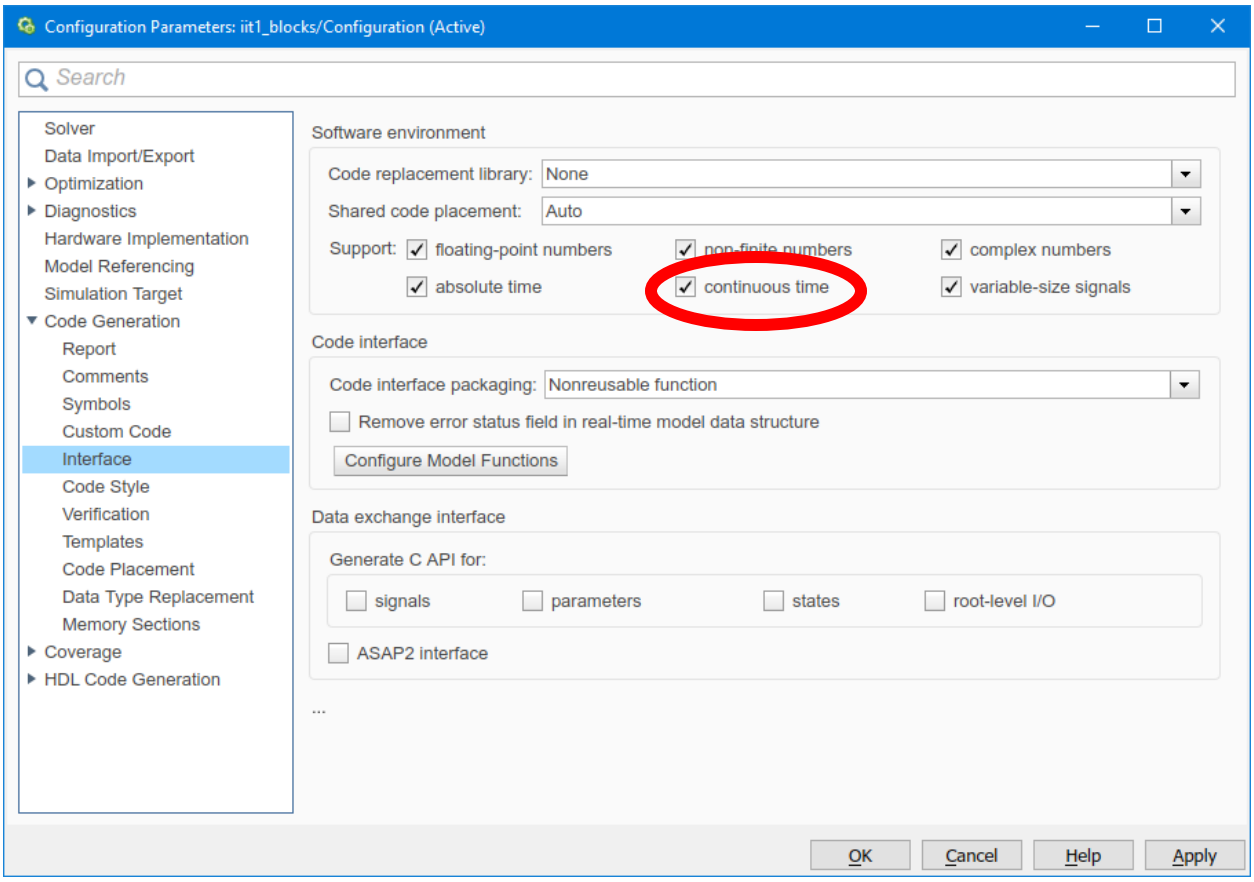
Kutta módszer), az *Additional parameters* panelt megnyitva pedig állítsunk be 0.01 másodperces lépésközt. Ez a beállítás azt jelenti, hogy az integrálási algoritmus a beágyazott processzoron 10ms-os periódusidővel fut, ami harmadrendű módszer mellett kellő pontosságot biztosít úgy, hogy nem terheli túl a mikrokontrollert (ne feledjük, emellett még a PC-vel is kapcsolatot kell tartania). A szimulációs idő végét (*Stop time*) érdemes `inf` értékre állítani, ekkor a modell mindaddig fut majd, amíg le nem állítjuk.

A modellt futtató processzor paramétereit a *Hardware Implementation* panelen (31. ábra) állíthatjuk be, legegyszerűbben úgy, ha a *Hardware Board* legödülőben az `Arduino Mega 2560`-at választjuk ki (a további beállításokat a Simulink ilyenkor automatikusan elvégzi). A külső módban használt fejlesztőkártyával a Simulink soros porton (pontosabban a fejlesztőkártyán található USB-UART átalakító által biztosított virtuális soros porton) keresztül tartja a kapcsolatot, a használni kívánt portot automatikusan választja ki. Amennyiben a PC-hez több, azonos típusú fejlesztőkártyát is csatlakoztatunk, célszerű a portot kézzel beállítani a *`Target hardware resources > Host-board connection`* panelen.

Mivel a járműmodell folytonosidejű, ezért a fentieken kívül ellenőrizzük azt is, hogy a *Code Generation > Interface* panelen aktív-e a *Support continuous time* opció (32. ábra)!

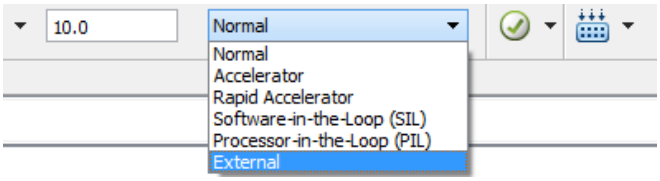


31. ábra – A felhasznált hardverplatform beállítása



32. ábra – A Code Generation > Interface panel

Külső futtatás esetén a megszokott *Normal* mód helyett az *External* szimulációs módot kell választani (32. ábra) indítás előtt. Külső futtatás esetén a fejlesztői PC és a céleszköz folyamatos, online kapcsolatban van: a PC-n futó Simulink-modell megfelelő blokkjai (pl. *Scope*-ok) a beágyazott platformon futtatott kód azonos változóit jelenítik meg, a források (pl. konstansok) értékeinek módosítása pedig hasonlóan érvényre jut a céleszközön futó kód megfelelő változóiban is.



33. ábra – Külső módú futtatás beállítása

Ebben az esetben nem közvetlenül a modellnek megfelelő kód kerül a beágyazott eszközre, az önmagában, a Simulink-kel való online kapcsolat nélkül nem működőképes.

A külső módban történő futtatás vezérlésére az *External Mode Control Panel* szolgál (`Code > External Mode Control Panel` menüpont). Külső futtatás közben megnyitva a `Disconnect` gombbal felfüggeszthetjük az erőforrásigényes adatcserét a Simulink és a targeten futó modell között, majd a `Connect` gomb segítségével újra csatlakozhatunk. Közben a modell futása nem áll le (csak a

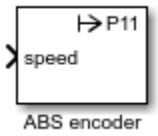
szimuláció megállítása esetén), így újra csatlakozva megspórolhatjuk a fordítás és feltöltés időrabló folyamatát. Természetesen ha közben változtatunk a modellen, azt újra kell fordítanunk.

5.2 Fizikai be- és kimenetek kezelése

Ahhoz, hogy a beágyazott platformon futtatott Simulink-modellt fizikailag is összekapcsolhassuk a szintén beágyazott platformon implementált szabályozóval, ki kell egészítenünk olyan elemekkel, melyek képesek a hardveres funkciók kezelésére (sebességfüggő enkóderjelek generálása, tempomat állapotától függő digitális kimenet kezelése, pedálállás-jel fogadása). Ezeket a blokkokat az `iit1_blocks.slx` fájl tartalmazza, onnan más modellbe is átmásolhatók.

ABS encoder

Az `ABS encoder` blokk a kerekeken lévő ABS-jeladó működését szimulálja, bemenete a jármű sebessége. A blokk a sebesség függvényében változó frekvenciájú, 50%-os kitöltésű négyszögjelet generál a 11-es lábon.



34. ábra – ABS encoder blokk

Tempomat On / Off

A `Tempomat On / Off` blokk a 49-es fizikai kimenet értékét állítja be attól függően, hogy bemenetére 0 (kikapcsolt állapot) vagy annál nagyobb értékű (bekapcsolt állapot) jelet kötünk.

35. ábra – Tempomat On / Off blokk

Read Throttle

A `Read Throttle` blokk a fejlesztőkártya 3. soros portján érkező adatok elérését segíti (a tempomatvezérlő ezen a csatornán küldi a kívánt pedálállást). A blokk a kimenetén 0 és 1 közé skálázva szolgáltatja a szabályozó által utoljára küldött beavatkozó jel-mintát.

36. ábra – Read Throttle blokk

5.3 A külső módban futtatandó járműmodell

A külső módban történő teszt során szeretnénk minél pontosabban modellezni a jármű és a tempomat használatát, így lehetőséget kell biztosítanunk arra, hogy a tempomat be- és kikapcsolását, kikapcsolt tempomat mellett a gázpedál állását, illetve a lejtőszöget futás közben a fejlesztő PC-ről módosíthatssuk.

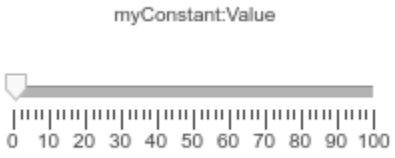
A pedálállást, a tempomat be- illetve kikapcsolt állapotát, illetve a lejtőszöget egy-egy konstansként (`Constant` blokk) érdemes szerepeltetnünk a modellben, ezek paramétere (a jel értéke) ugyanis külső módban a fejlesztői PC-ről futásidőben is kényelmesen módosítható.

A járműmodell pedálállás-bemenetére a tesztelés során azonban nem egyetlen meghatározott értéket, hanem a tempomat bekapcsolt állapotától függően vagy a `Read Throttle` blokk által szolgáltatott értéket, vagy egy (a fejlesztői PC-ről futásidőben is beállítható értékű) konstans blokk kimenetét kötjük. Erre jól használható a korábbiakban ismertetett `Switch` blokk, melynek választó (második) bemenetére a tempomat állapotának 0 vagy 1 értékű jelét kötve a kimenetre bekapcsolt tempomat esetén az első, míg kikapcsolt tempomat esetén a harmadik bemenet jele kerül továbbításra.

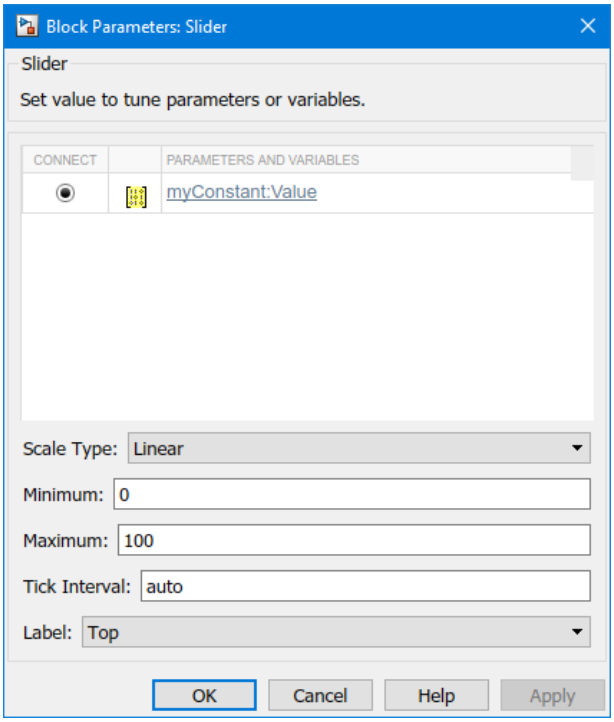
Természetesen a tempomat állapotát a `Tempomat On / Off` blokk bemenetére is be kell kötni, hogy az a beágyazott platformon futó tempomatvezérlő felé fizikai vonalon is továbbításra kerüljön. Hasonlóan a járműmodell által előállított sebességjelet az `ABS encoder` blokk bemenetére is be kell kötni.

Külső futtatás esetén a beágyazott targeten futó modell jeleinek illetve paramétereinek megjenítésére és beállítására kényelmes lehetőséget kínálnak a `Dashboard` könyvtár (`Simulink > Dashboard`) elemei. Elérhetők többek között logikai kapcsolók, csúszkák, tekerőgombok, illetve mutatós műszerek vagy éppen kijelzők is (sajnos a megjelenítő elemek a mérésen használt Arduino Mega 2560 paneleken futó modell esetén nem használhatók). A dashboard-elemek önmagukban nem alkalmazhatók, az általuk megjelenített vagy beállított érték csak más blokkok változóihoz (pl. egy konstans blokk értékéhez vagy egy integrátor kimenetéhez) köthetők.

A dashboard-elem párbeszédablakát (38. ábra) megnyitva majd a kiválasztott blokkdiagram-elemre (konstans, erősítés stb.) kattintva megjelennek az adott blokk azon változói, melyet a dashboard-elemhez köthetünk. Konstans esetén ez a konstans értéke, de például egy integrátor esetén akár a telítés határait is hozzárendelhetjük egy csúszkához. A megjelenő lehetőségek közül egyet köthetünk a dashboard-elemhez a párbeszédablak *connect* oszlop megfelelő elemét kijelölve. Ezen kívül egyéb paraméterek (pl. a skála alsó és felső határának vagy egy kapcsoló két állásához tartozó értékek) beállítására is van lehetőség.



37. ábra – Slider típusú dashboard-elem



38. ábra – Dashboard-elem kötése

5.4 Mérési feladatok

Módosítsa a járműmodellt úgy, hogy az külső módban futtatva alkalmas legyen a tempomatvezérlő HIL-tesztjének elvégzésére!

- adja hozzá a modellhez az **`iit1_blocks.slx`** modellben elérhető **IO-kezelő blokkok**at
- **módosítsa a modellt** úgy, hogy a tempomat állapotától függően a pedálállást egy konstans blokk vagy a tempomatvezérlő által küldött jel határozza meg
- a tempomat állapotát meghatározó konstanshoz rendeljen egy `Rocker Switch` típusú, míg a pedálállást és a lejtőszöget meghatározó konstansokhoz egy-egy (megfelelően skálázott) `Slider` típusú **dashboard-elem**et
- a jármű sebességéhez kössön egy **`Scope`** és egy **`Display`** blokkot (a *Simulation Data Inspector* külső módban nem használható)

A járműmodellt külső módban futtatva ellenőrizze a beágyazott platformon megvalósított tempomatvezérlő működését!