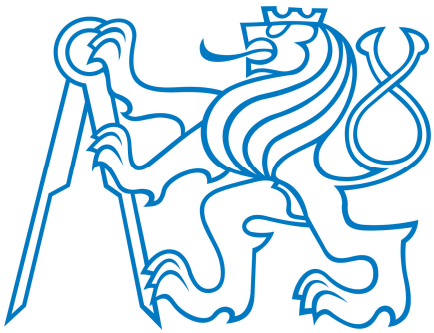
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta strojní – Ústav přístrojové a řídicí techniky



BAKALÁŘSKÁ PRÁCE

Komunikační rozhraní pro měřicí kartu

Communication interface for data acquisition card

Vedoucí: Ing. Pavel Trnka, Ph.D.

Jan Vlk

2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vlk** Jméno: **Jan** Osobní číslo: **475005**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídící techniky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Komunikační rozhraní Simulink-Labjack

Název bakalářské práce anglicky:

Communication interface Simulink-Labjack

Pokyny pro vypracování:

1) Vytvořte Level-2 s-funkci pro řízení laboratorních úloh Automatického řízení (AŘ) prostřednictvím softwarového prostředí Matlab - Simulink a USB měřicí jednotky LabJack.
2) S-funkci otestujte na laboratorní úloze Kulička na tyči.
3) Pro uvedenou úlohu vytvořte uživatelská rozhraní Simulink pro laboratorní cvičení předmětu AŘ.

Seznam doporučené literatury:

[1] Write Level-2 MATLAB S-Functions. Dostupné z www: https://www.mathworks.com/help/simulink/sfg/ writing-level-2-matlab-s-functions.html
[2] MATLAB for UD - Windows. Dostupné z www: https://labjack.com/support/software/examples/ud/matlab
[3] Fišer Jaromír. Úloha O7 'Kulička na tyči'. Dostupné z www: http://vlab.fs.cvut.cz/navody/files/kul_tyc_navod.pdf
[4] Další návody k úlohám AŘ - viz http://vlab.fs.cvut.cz/navody/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Trnka, Ph.D., U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **30.04.2021** Termín odevzdání bakalářské práce: **24.08.2021**

Platnost zadání bakalářské práce: _____

Ing. Pavel Trnka, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ Datum převzetí zadání

_____ Podpis studenta

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal

Datum: . . . .

podpis

Poděkování

Chtěl bych poděkovat rodině za podporu nejenom během psaní práce, ale i v průběhu celého studia a mému vedoucímu Ing. Pavlovi Trnkovi, Ph.D., za jeho užitečné rady a připomínky a za jeho vstřícný přístup.

Anotace

Cílem této práce je seznámit čtenáře s laboratorní kartou LabJack U3-HV a jejím ovládáním pomocí prostředí MATLAB, dále s Level-2 S-Funkcí v prostředí Simulink a způsoby jejich vytváření a s laboratorní úlohou 'Kulička na tyči'. Za pomocí těchto znalostí je poté navržena Level-2 S-funkce pro komunikaci s kartou LabJack, její otestování a zasazení do uživatelského rozhraní pro laboratorní úlohu 'Kulička na tyči'.
Klíčová slova: Laboratorní úloha, Level-2 S-Funkce, LabJack, MATLAB, Simulink

Abstract

The aim of this work is to familiarise the reader with the LabJack U3-HV lab card and its control using the MATLAB environment, as well as the Level-2 S-Function in the Simulink environment and the methods of creating them, and the laboratory task 'Ball and Beam'. Using this knowledge, the Level-2 S-function to communicate with the LabJack card is designed, tested and implemented into user interface for laboratory task 'Ball and Beam'.
Key words: Laboratory task, Level-2 S-Function, LabJack, MATLAB, Simulink

Obsah

Seznam použitých značek, zkratek a symbolů

HW - Hardware

SW - Software

PC - počítač

USB - Universal Serial Bus

HV - High Voltage

tzv.- takzvaný

např. - například

FIFO - First In, First Out

LJ - LabJack

ID - Identifikace

IP - Internet Protocol

PID - Proporcionální-integrační-derivační

V - Volt

mV - milivolt

mA - miliampér

Ω- ohm

mm - milimetr

s - sekunda

E/P - elektricko-pneumatický

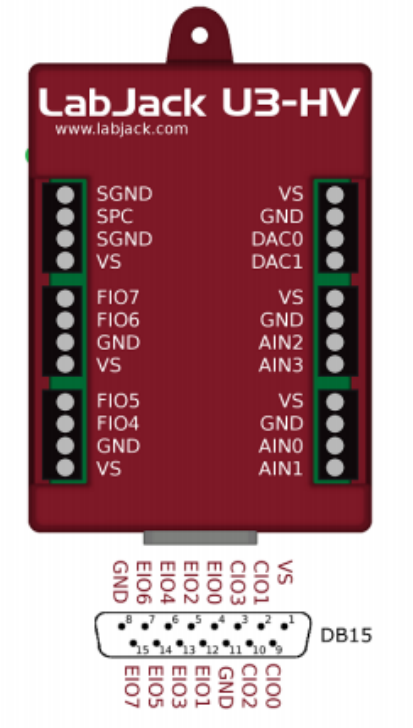
API - Application Programming Interface

1 Úvod

Mým cílem při zpracování práce bylo nejprve se seznámit s laboratorní kartou LabJack a to jak z HW (hardwarové), tak SW (softwarové) stránky, tedy významem jednotlivých svorek a s ovládáním karty pomocí počítače, následované seznámením se Simulinkovými Level-2 S-Funkcemi, které mi umožní ovládat kartu LabJack pomocí řídícího obvodu v programu Simulink. Následně bylo mým úkolem tyto znalosti aplikovat vytvořením funkce a prostředí pro laboratorní úlohu 'Kulička na tyči' a tuto funkci otestovat.

2 Laboratorní karta LabJack

2.1 Hardwarový popis karty



Obr. 1: Scháma karty LabJack [1]

2.1.1 USB

LJ (LabJack) je s počítačem propojen pomocí full speed USB 2.0 (případně 1.1), které zajišťuje napájení i přenos dat. Fungování spojení je zajištěno ovladači na počítači (v mém případě LJUD). [1]

2.1.2 GND a SGND svorky

Svorky GND a SGND zajišťují připojení k uzemnění. Toto uzemnění je vedeno přes USB a tudíž je společné s uzemněním počítače, tedy i s uzemněním v zásuvkách. Na tento fakt je potřeba myslet při zapojování externího zdroje, aby nedošlo ke zkratu. Svorka SGND má navíc tepelnou pojistku. Tato svorka je vhodná pokud není jisté zda nezdílíme uzemnění s jiným zařízením. [1]

2.1.3 VS svorky

Svroky VS slouží jako zdroj napětí o hodnotě 5 V, které poskytuje USB připojení. Maximální možný odebíraný proud je 450 mA [1]

2.1.4 Flexible I/O svorky

Označené FIO0-7 a EIO0-7 Jednotlivé svorky se dají nastavit jako digitální vstup/výstup (čtení/zapisování binární informace o stavu), nebo analogový výstup. FIO jsou přístupné na svorkovnici, EIO jsou na DB15 konektoru (viz obázek 1). Pokud je svorka nastavena jako analogový vstup (AIN) pak je označována jako AIN0-7 pro FIO a AIN8-15 pro EIO. Pro variantu HV jsou první 4 svorky (FIO0-3) pevně nastaveny jako AIN a nelze s nimi pracovat jinak. [1]

2.1.5 Analogové vstupy (AIN)

Rozsah AIN je běžně 0-2.44 V, speciální rozsah 0-3.6 V vůči svorkám GND. Lze také zapojit jako $\pm$ 2.4 V pseudobipolárně, měří se pouze rozdíl mezi dvěma napětími, přičemž maximální rozsah je -0.3 až +3.6 V. Ve verzi HV, kde jsou první čtyři FIO automaticky nastaveny jako AIN jsou tyto vstupy v rozsahu $\pm$ 10 V (skuteřně bipolární, tedy limit každé svorky je $\pm$ 10 V) a -10 až +20 V pro speciální rozsah. Rozlišení analogových kanálů je 12 bitů. [1]

2.1.6 Analogové výstupy (DAC)

Jedná se o svorky DAC0 a DAC1, které jsou napevno nastavené jako analogový výstupy s rozsahem 0.04 až 4.95 voltů vůči svorkám GND. Díky stabilnějšímu zdroji (jsou interně napojeny na 3.3 V zdroj a následně zesíleny) je jejich výstup kvalitnější, než výstup VS svorek. [1]

2.2 Módy provozu

Karta LabJack má 2 módy provozu. Prvním je tzv. Command/Response mód (příkaz/reakce), ve kterém si čtení vstupů vyžádá ovladač. Tento přístup je pomalejší a hodí se tak pro nižší frekvenci čtení. Druhým módem je Stream. V tomto módu jsou vstupy zaznamenávány automaticky s danou frekvencí do paměti karty a uživatel si následně vyžádá větší objem dat najednou. [1]

2.2.1 Command/Response

Komunikace s kartou v tomto módu se skládá ze 3 kroků a to vytvoření sady příkazů, jejich odeslání a vykonání a nakonec sběr získaných dat. Rychlost čtení je dána několika faktory a to například rychlostí běhu programu, či počtem a náročností operací. Časovou odezvu nejvíce ovlivňuje čtení analogových vstupů (AIN), při kterém je odezva lineárně závislá na počtu kanálů. Téměř bez vlivu jsou operace digitálního čtení či zápisu. Počet příkazů a odpovědí je limitován objemem přenesených dat (57 bitů příkazů a 55 bitů odpovědí, přičemž například funkce AIN má 3 příkazové a 2 odpovědní bity). Pokud je tento objem

dat překročen karta požadavky rozdělí. Tento mód není vhodný pro vysokofrekvenční práci zejména kvůli komunikaci s počítačem. [1]

2.2.2 Stream

Tento mód se běžně využívá pro vysokofrekvenčí komunikaci (velký počet dat). Frekvence čtení je dána kartou (tím je zajištěna její stálost a přesnost). Nejprve je nutné objasnit některé pojmy: vzorek je čtení z jednoho kanálu a sken je čtení ze všech kanálů. Vzhledem k omezené rychlosti spojení mezi počítačem a kartou využívá tento mód vyrovnávací paměť v režimu FIFO (zachovává se pořadí dat tak, že data která přijdou do paměti první, jsou z ní jako první i získávána), do které lze uložit až 984 vzorků. Tyto data jsou poté ve větších balíčcíh posílána na počítač. V případě že dojde k přeplnění vyrovnávací paměti, začne karta data mazat. Kvůli zachování správného časování je smazaný počet vzorků zaznamenán a následně je v ovladači vytvořen potřebný počet chybových vzorků (-9999.0). Maximální vzorkovací frekvence této karty je 50 000 vzroků za sekundu, ale při této frekvenci je čtení náchylnější na rušení a jeho rozlišení není plné (kolem 10 bitů). Pro plné rozlišení (12 bitů) je maximální vzorkování 2500 vzorků za sekundu. Pokud chci streamovat funkce jako například časovače, nebo čítače, musím příslušné kanály nastavit před začátkem streamování. [1]

2.3 Ovladač LabJackUD

Jedná se o vysokoúrovňový ovladač využívaný většinou aplikací. Funguje na základním principu otevření komunikace, sestavení listu požadavků, jejich vykonání a získání dat. Tyto principy jsou vykonávány čtyřmi základními funkcemi: Open, AddRequest, Go a GetResult. Každá akce je dále specifikována konstantami, které je vhodné mít k dispozici pro kopírování. Tyto konstanty poté předáváme jako jednotlivé parametry funkcím. [1]
Důležité společné parametry funkcí:
Handle: tento parametr předává funkcím informaci se kterou kartou mají komunikovat (nutné i v případě jedné karty)
IOType: tímto parametrem určíme jaký typ akce chceme vykonat (zda chceme číst, či zapisovat analogově, nebo digitálně)
Channel: zde předáváme informaci o kanálu, který chceme využívat, nebo konfigurovat
Value: tento parametr je hodnota použitá pro zápis, či získaná ze čtení dat
x1: Tento parametr je určen pro funkce vyžadující speciální informaci
UserData: zde předáváme data, se kterými se nic neděje a slouží například k popisu
Výstupy jsou obvykle označeny písmenem p (pIOType a pod.) [1]

2.3.1 Základní funkce

Funkce OpenLabJack()

Touto funkcí se zahajuje komunikace s LJ, pomocí které získáváme parametr Handle, používaný v jiných funkcích.
deklarace: OpenLabJack(DeviceType, ConnectionType, *pAddress, FirstFound, LJ_HANDLE *pHandle)
parametry:
DeviceType, kterým specifikujeme typ LJ (konstanty najdeme v souboru labjackud.h)
ConncectionType, kde definujeme typ spojení (USB, nebo Ethernet)
pAddress předává informace o ID nebo sériovém čísle (USB), nebo IP (Ethernet)
FirstFound Pokud je tento parametr zadán jako pravdivý (True), jsou předtím zadané hodnoty typu zařízení, připojení a ID ignorovány a použije se první nalezená karta
Výstupy:
pHandle vrací ukazatel na objekt handle využívaný ostatními funkcemi (zde konkrétně ho ukládáme pod názvem LJ_HANDLE) [1]

AddRequest()

Touto funkcí sestavujeme list požadavků, které se budou následně vykonávat. Všechny požadavky jsou smazány poté co je funkce zavolána znovu (pokud je použito více zařízení, mažou se pouze požadavky pro jedno konkrétní z nich podle parametru handle).
deklarace: AddRequest(Handle, IOType, Channel, Value, x1, UserData)
Vstupy jsou základní parametry, které jsou popsány výše, výstup nemá tato funkce žádný. [1]

Go(), GoOne()

Tímto příkazem vykonáme list požadavků vytvořený funkcí AddRequest() na všech dostupných zařízeních a vytvoří list výsledků. Tato funkce list požadavků nesmaže, je tedy možné ji volat na stejný list požadavků opakovaně. Pořadí, v jakém se požadavky vykonají nelze předvídat, jediná jistá posloupnost je, že požadavky jako nastavení rozsahů, nebo stream módu jsou vykonány před čtením, či zápisem dat. Funkce GoOne() tyto požadavky vykoná pouze na jednom specifikovaném zařízení.
deklarace: Go(), GoOne(Handle)
Funkce Go() nemá žádné vstupy, funkce GoOne() má pouze vstup Handle. Obě funkce nemají žádný výstup. [1]

GetResult()

Touto funkcí čteme list výsledků vytvořený funkcí Go(). Funkci použijeme na každý IOType a Channel, ze kterého chceme získat výsledky. Je vhodné tuto funkci volat i na požadavky, které zapisují data z důvodu získání potenciální chybové hlášky. Je důležité si uvědomit, že tato funkce list výsledků nevytváří, ale pouze čte. List výsledků je smazán až dalším zavoláním funkce AddRequest().
deklarace: GetResult(Handle, IOType, Channel, pValue)
Vstupy jsou opět základní parametry (pokud nás daný parametr není potřeba, použijeme hodnotu NULL), výstupem je parametr pValue, ve kterém je výsledná hodnota (případně ukazatel na něj). [1]

GetFirstResult(), GetNextResult()

Těmito funkcemi lze získávát výsledky postupně. Pokud nejsou žádná data k získání, vrátí funkce speciální chybovou hlášku.
deklarace (pro obě stejná): GetFirstResult(Handle, pIOType, pChannel, pValue, px1, pUserData)
Jako vstup slouží pouze Handle k určení zařízení. Výstupy jsou informace o výsledku, který jsme získali (případně ukazatele na ně) [1]

eGet() a ePut()

Tyto funkce slouží k provedení příkazů AddRequest, Go a GetResult v jednom kroku, přičemž eGet slouží k získávání dat a ePut k jejich zapisování.
Deklarace: eGet(Handle, IOType, Channel, *pValue, x1)
Pro vstupy slouží stejné vlastnosti jako popsané výše, výstup je parametr pValue, uchovávající ukazatel na danou hodnotu. [1]

2.3.2 Nastavení

Nastavení se provádí stejně jako jiné operace pomocí funkcí AddRequest() a Go(), a to za pomocí konstanty LJ_ioPUT_CONFIG na místo parametru IOType a požadovaného nastavení jako parametr Channel. (1)

2.3.3 Ovládání Stream modu

Stream mód se ovládá pomocí funkcí používaných ke komunikaci s kartou i v Command/Response módu, se specifickými hodnotami parametrů. [1]

Parametry IOType

Některé hodnoty používané na pozici IOType pro ovládání Stream módu:
LJ_ioCLEAR_STREAM_CHANNELS slouží k vymazání předchozích streamovacích kanálů
LJ_ioADD_STREAM_CHANNEL slouží k přidání jednoho streamovacího kanálu
LJ_ioADD_STREAM_CHANNEL_DIFF přidává několik kanálů najednom v daném rozsahu
LJ_ioSTART_STREAM zapne stream
LJ_ioSTOP_STREAM vypne stream
LJ_ioGET_STREAM_DATA se používá pro získávání dat [1]

Parametry Channel

Stream se nastavuje stejně jako vše ostatní pomocí parametu PUT_CONFIG na pozici IOType a následně konkréttních nastavení na pozici Channel. Některé z důležitých nastavení jsou:
LJ_chSTREAM_SCAN_FREQUENCY nastavuje frekvenci skenování
LJ_chSTREAM_BUFFER_SIZE nastavuje velikost mezipaměti v kartě
LJ_chSTREAM_WAIT_MODE nastavuje způsob čekání na data
LJ_chSTREAM_SAMPLES_PER_PACKET nastavuje počet vzorků zaslaných jedním paketem
LJ_chSTREAM_READS_PER_SECOND nastavuje frekvenci čtení [1]

Parametry Value

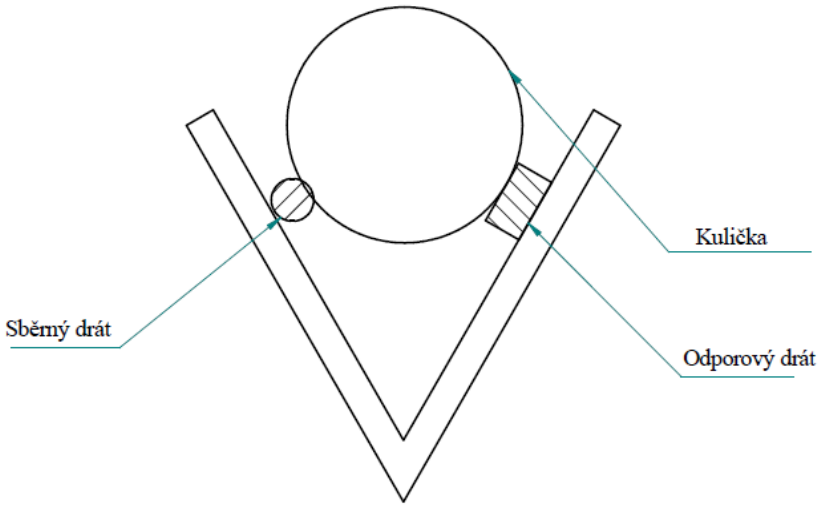
Na pozici parametru Value se udávají konkrétní hodnoty, jako frekvence, nebo například pro wait mode upřesňující informace. Těmi se upřesňuje na co konkrétně má program čekat (např. LJ_swSLEEP bude čekat dokud se nezíská požadovaný počet dat). [1]

3 Úloha kulička na tyči

Úkolem této laboratorní úlohy je určit parametry regulátoru například metodou Ziegler-Nichols, nebo metodou relé a požadované kvality regulace (odstranění trvalé regulační odchylky, útlum vlastních kmitů).

3.1 Popis úlohy

Jako regulovaný systém je v této úloze použita kulička na naklopitelné tyči. Pomocí naklápění tyče a vlivu gravitace se může kulička pohybovat a tím měnit polohu y. Tu zjišťujeme pomocí odporového a sběrného drátu, po kterých se kulička odvaluje. Polohu pak porovnáváme s požadovanou hodnotou w a tím získáváme regulační odchylku $e = w - y$. Podle této odchylky je pak pomocí PID regulátoru nastaven požadovaný náklon tyče (akční člen). Ten je řízen pneumatickým signálem získaným z E/P (elektricko-pneumatického) převodníku, který ovládá pneumatický membránový ventil, jehož pohyb je přímo převáděn na daný náklon. [4]



Obr. 2: Schéma kontaktu kuličky

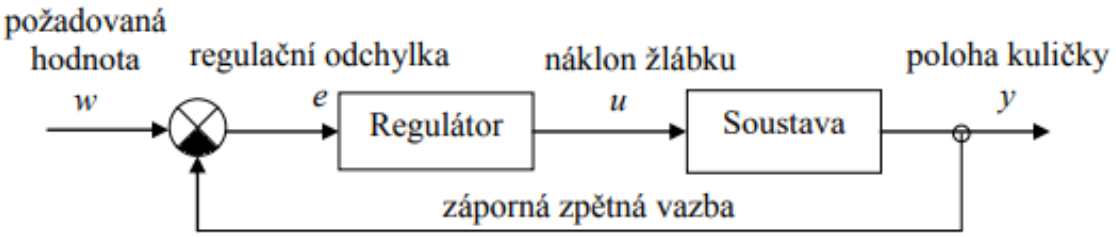
Pohyb kuličky lze popsat rovnicí [4]

$$a_2 \frac{d^2 y|(t)}{dt^2} = K \cdot u(t), a_2 > 0, K > 0 \tag{1}$$

kde $u(t)$ je výstup z PID regulátoru [4]

$$u(t) = r_0 e(t) + \frac{r_0}{T_i} \int_0^t e(\tau) d\tau + r_0 T_d \frac{de(t)}{dt}. \tag{2}$$

Spojením těchto rovnic dostaneme uzavřený regulační obvod (URO)



Obr. 3: Schéma URO soustavy [4]

Úkolem tohoto regulačního obvodu je kuličku dostat na požadované místo a v případě působení chybové veličiny (například do kuličky někdo strčí) ji na toto místo zase vrátit. [4]

4 Level-2 MATLAB S-funkce

S-Funkce (System-functions) nám dovolují vytvořit vlastní programové bloky, které lze využít jako součást modelu v programu Simulink. Pro fungování se Simulinkem využívají tyto funkce S-functions API (Application Programming Interface), která funguje na principu jako běžné bloky. S-funkce lze psát v různých programovacích jazycích (např. MATLAB, C, C++) a podle stylu v jakém jsou napsány se dělí na různé typy. Pro jazyky jiné než MATLAB je potřeba použít C MEX Funkci, kterou matlab pak automaticky načítá. Další možností jsou Level-1 a Level-2 S-Funkce, které jsou psány v jazyce MATLAB. Nejčastějším použitím S-funkce je vytváření logického bloku, který pro potřeby simulace v Simulinkuu chybí a lze ho s různými parametry používat opakovaně. Další případy využití jsou například řízení hardwarového zařízení (náš připad), nebo implementace již existujících kódu či animací. [2]

4.1 Fáze simulace

Každý model v Simulinku běží ve fázích. Ihned po spuštění simulace se spustí fáze inicializace, která proběhne pouze jednou a po ní následuje smyčka iterační fáze. Ve fázi inicializace se načtou počáteční hodnoty simulace. V iterační fázi se pak opakují kroky výpočtů nových vstupů, stavů a výstupů po časových krocích, dokud simulace nedosáhne požadovaného simulačního času. Celá simulace končí fází Terminate, která zajistí správné ukončení simulace. [3]

4.2 Základní vlastnosti Level-2 S-funkce

Level-2 S-funkce fungují na principu spouštění jednotlivých částí programu (tzv Callback metody) v různých částech simulace. Pro přenos dat mezi simulinkem a těmito metodami je jako argument předáván Run-Time object, který předává metodám informace o vstupech a výstupech bloku, parametry a stavy. Tento objekt je vytvořen automaticky pro každý blok reprezentující Level-2 S-Funkci v modelu. [2]

4.3 Callback metody

Metoda Setup

Touto metodou se nastavují vstupy, výstupy a další parametry S-Funkce. Jedná se o první metodu, kterou simulink zavolá při spuštění simulace.
Zavedení: setup(s)
Nastavení počtu parametrů pomocí s.NumDialogPrms:
s.DialogPrmsTunable určení nastavitelnosti parametrů (Tunable, Notunable)
Nastavení vstupů:
s.NumInputPort - počet vstupů

s.InputPort(i).Dimensions - rozměry i-tého vstupu
s.InputPort(i).SampleTime - nastavení časového kroku pro daný vstup
s.InputPort(i).DirectFeedthrough - nastavení průchodnosti parametru (pokud je parametr použit v metodě Outputs, nastavit jako 1)
Nastavení výstupů:
s.NumOutputPorts - počet výstupů
s.OutputPort(i).Dimensions - rozměry i-tého výstupu
s.OutputPort(i).SampleTime - nastavení časového kroku pro daný výstup
s.SampleTimes - nastavení časového kroku pro celý blok [2]

Metoda Outputs

Tato metoda je spuštěna na konci každého kroku a počítá hodnoty, které následně využívá blok S-Funkce jako výstupy. Tyto hodnoty jsou spolu se současným časovým krokem předány do Run-time objektu. Pokud jsou časové kroky proměnné, pak také počítá hodnotu dalšího časového kroku
zavedení: Outputs(s) [2]

Metoda Terminate

Slouží k provedení akcí ke správnému ukončení simulace, jako například mazání paměti. Spustí se pokud je simulace zastavena, nebo je blok S-Funkce vymazán.
zavedeni: Terminate(s) [2]

Metoda Derivatives

Tato metoda je spuštěna (pokud je zavedena) v každém časovém kroku a počítá derivace spojitých stavů. Hodnoty by měly být uloženy ve stavovém derivačním vektoru. Při každém jejím spuštění by se měly všechny hodnoty uvádět znovu, i pokud zůstanou nezměněné, protože vektor stavů neuchovává informace z předchozího časového kroku.
Zavedení: Derivatives(s) [2]

Metoda ProcessParameters

Metoda ProcessParameters zpracovává nově změněné parametry v simulaci. Pokud je ve funkci použita, volá se na začátku simulační smyčky, kdy je možné bezpečně parametry měnit.
Zavedení: ProcessParameters(s) [2]

Metoda InitializeConditions

Jedná se o další volitelnou metodu, která se spouští na začátku simulace a používá se zejména k inicializaci spojitých a nespojitých stavů. Pokud je to nutné, lze zde zavádět i jiné akce. Tato metoda je volána také pokud jsou v simulaci využity podsystémy vyžadující obnovení stavů. Protože je tato metoda volána před výpočtem vstupních signálů, neměly by být v této metodě využívány jako parametry. Pokud je nutno podmínky zavádět s využitím vstupů, mělo by se tak učinit v metodě Outputs.
Zavedení: InitializeConditions(s) [2]

Metoda Update

Touto volitelnou metodou jsou v každém hlavním časovém kroku aktualizovány stavy simulace ve stavovém vektoru, případně vykonány další činnosti vyžadovené v každém hlavním kroku.
Zavedení: Update(s) [2]

Metoda Start

Podobně jako InitializeConditions se tato metoda spouští pouze jednou na začátku simulace (pokud je zavedena). Na rozdíl od InitializeConditions se nepoužívá k zavedení stavů, ale pouze k provedení procesů jako je alokace paměti, nebo nastavení uživatelských dat.
Zavedení: Start(s) [2]

Metoda Set*PortDataType

Jedná se o dvě metody, které řeší jaký typ dat je očekáván na vstupu či výstupu. Pro obě možnosti funkce fungují stejně. Těmito funkcemi lze přiřazovat typy dat pouze k portům, které zatím žádný typ přiřazený nemají.
Zavedení: SetOutputPortDataType(s, port, id) pro výstupní porty
SetInputPortDataType(s, port, id) pro vstupní porty
Parametrem port předáváme číslo portu a parametrem id na jaký typ dat jej chceme nastavit. Pro samotné nastavení použijeme příkaz s.*Port(port).DatatypeID = id;. Pokud tato metoda není použita, pak simulace umožní použití jakéhokoli typu dat a port nastaví podle dat, která jsou jím posílána. Pokud je tato metoda použita, pak je volána do tě doby, doklud všechny nenastavené porty nemají svůj typ dat. [2]

Metoda Set*PortDimensions

Podobně jako v předchozím případě se jedná o dvě metody zajišťující velikost vstupních a výstupních portů.
Zavedení: SetOutputPortDimensions(s, port, dimsInfo) pro výstupní porty

SetInputPortDimensions(s, port, dimsInfo) pro vstupní porty
Parametr dimsInfo předává informace o rozměrech podporovaných daným portem (např. [a] pro vektor o a řádcích, nebo [a b] pro matici o a řádcích a b sloupcích) a nastavení je vyvoláno příkazem s.InputPort(port).Dimensions = dimsInfo;. Obdobně jak opro nastavení typu dat je tato metoda volána(pokud zavedena) dokud nemají všechny porty nastavené rozměry, tyto metody jsou ovšem volány pouze pokud lze rozměry portu zjistit z portu ke kterému je připojen. [2]

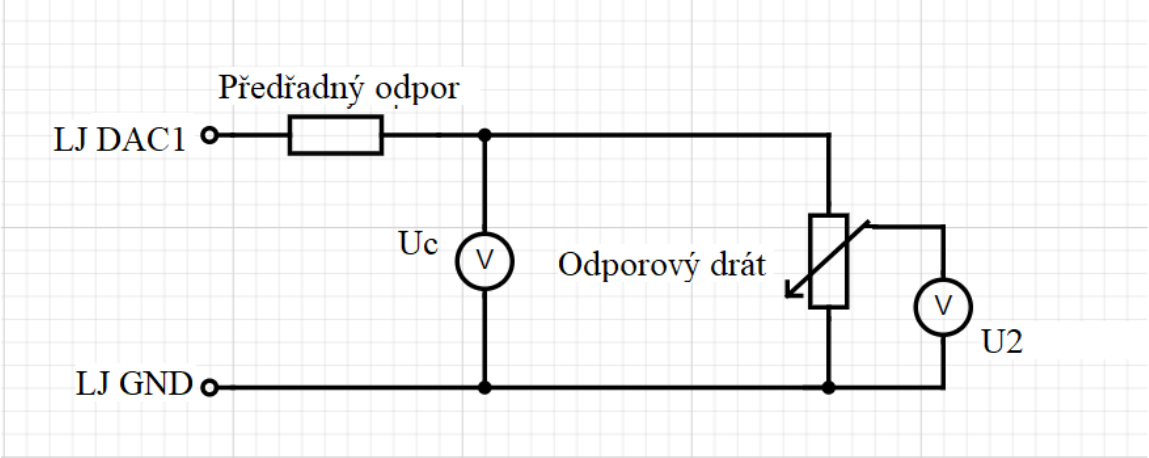
Metoda Set*PortSampleTime

Opět se jedná o dvojici metod, tentokrát pro nastavení periody vzorkování na vstupních a výstupních portech. Metoda je volána (pokud zavedena) s parametrem získaným z portu, ke kterému je požadovaný port připojen a je volána dokud všechny porty nemají zaveden vzorkovací čas. Vzorkovací časy se obvykle předávají ve směru simulace, ovšem pro výstupní porty se může stát, že simulace využije následujícího bloku, pokud je to možné.
Zavedení: SetInputPortSampleTime(s, port, time) Pro vstupní porty
SetOutputPortSampleTime(s, port, time) Pro výstupní porty
Parametr time je dvouprvkové pole specifikující vzorkovací periodu a kompenzaci ([perioda kompenzace]) a samotné nastavení je provedeno příkazem s.*Port(port).SampleTime = time;. [2]

Praktická část

5 Zapojení úlohy

5.1 Měřící část obvodu



Obr. 4: Náhradní schéma zapojení úlohy

Pro zapojení části úlohy určené k měření využijeme svorky DAC1 (tato svorka zvolena pro možnost volby napájecího napětí a stabilnějšího signálu) jako zdroj napájení 2 V a svorku GND jako společné uzemnění. Pro samotné měření pak svorky AIN0 a AIN1, na kterých budeme odečítat napětí Uc a U2. Pro ochranu zdroje napětí bude možná nutno přidat předřadný rezistor. Pro určení hodnoty tohoto odporu jsem vycházel z maximálního proudu, které by měl být schopný dodat každý USB 2.0 konektor. Typická hodnota pro běžné USB 2.0 je 100 mA a po odečtení vlastní spotřeby laboratorní karty, která činí 50 mA [1] nám zbývá 50 mA na napájení obvodu. Při výpočtu vycházím z ova zákona a z předpokladu, že žádnou ze svorek určených k měření neprotéká proud, což je Ohmvzhledem k tomu že se jedná pouze o ochranný postředek, který neovlivňuje samotné měření dostatečné. Pak proud protékající obvodem je

$$I = \frac{U}{R_p + R_c} \tag{3}$$

kde $U = 2V$ je napětí přiváděné na svorku DAC1 vůči svorce GND
$R_p[\Omega]$ je hledaný předřadný odpor
a $R_c[\Omega]$ je celkový odpor odporového drátu.

Úpravou této rovnice dostáváme celkovou potřebnou velikost odporu:

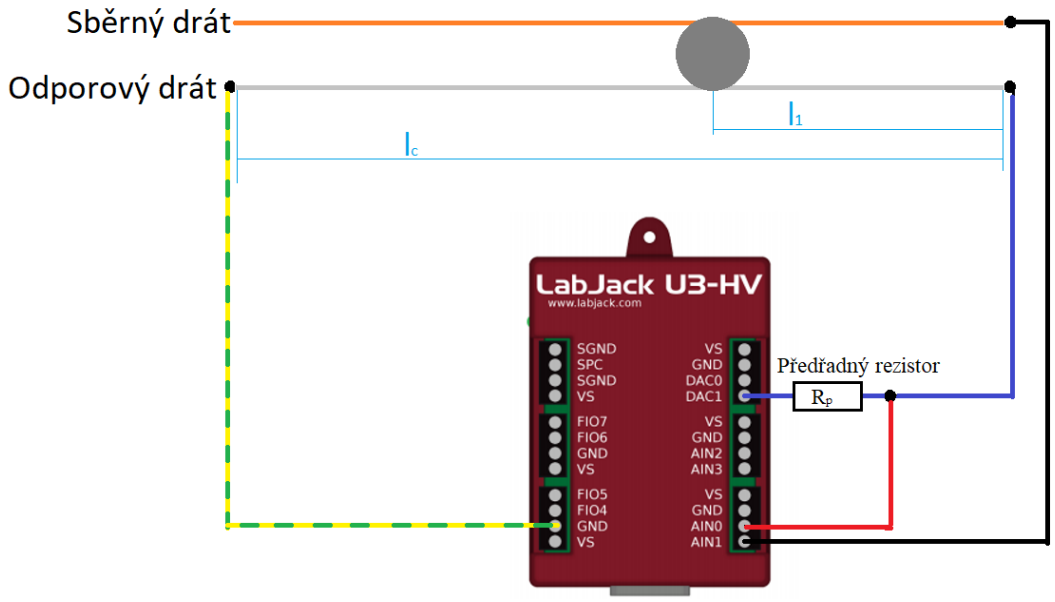
$$R_p + R_c \geq \frac{U}{I} = \frac{2}{50 \cdot 10^{-3}} = 40\Omega \tag{4}$$

R_c je v našem zapojení zhruba 20 Ω, předřadný rezistor tedy musí být 20 Ω.
Zpojením předřadného rezistoru dojde ke snížení napětí, které měříme, ale vzhledem k tomu že porovnáváme hodnoty napětí na celém odporovém drátu a na jeho části, dojde tímto krokem pouze ke snížení rozsahu a to podle rovnice

$$U_c = U\frac{R_c}{R_c + R_p} = 2 \cdot \frac{20}{20 + 20} = 1V \tag{5}$$

na rozsah zhruba 1 V. I zde nevadí nepřesná velikost celkového odporu drátu a jiná zanedbání, jelikož přesnou hodnotu R_c měříme. Tento rozsah nám vzhledem k rozlišení LabJacku 0.6 mV pro svorky AIN při zapojení pro rozsah 0-2.44 V dává na metrové tyči rozlišení alespoň 0.6 mm, což je pro tuto úlohu dostačující.

Ve skutečmosti pak bude úloha zapojena takto, přičemž pneumatický ventil pro ovládání náklonu bude připojen k pinu DAC0.



Obr. 5: schéma připojení k LabJacku

6 Level 2 S-Funkce

Jak již bylo zmíněno výše, pro ovládání laboratorní karty LabJack je možno využít dvou módů a to Command/Response nebo Stream. Rozhodl jsem se vyzkoušet oba a na základě testů rozhodnout, který se pro aplikaci na úlohu kulička na tyči hodí více. Liší se především v citlivosti na šum a přístupu k udržování reálného času.
Pro správné fungování těchto funkcí můsí být kromě MATLABu nainstalován i ovladač LJUD, který je dostupný ke stažení na [5]

6.1 S-funkce pomocí Command/response módu

Kód je k dispozici v příloze 1.
Jedná se o jednodušší variantu, kde je čas udržován podle času v počítači a to díky nástavbě Simulink Desktop Real-Time, která nám umožňuje přidat do modelu v Simulinku blok Real-Time Sync, který synchronizuje časový krok simulace s časem řízeným počítačem. Nevýhodou tohoto přístupu je nestálost času počítače v závislosti na jeho vytížení a skutečnost, že při každém kroku simulace čteme hodnotu pouze jednou, tedy je zde větší náchylnost k chybám.

Metoda Setup

V tomto módu řešení jsou v metodě Setup pouze nastaveny parametry S-funkce, tedy počet vstupů a výstupů, jejich vlastností, nastavení časového kroku a deklarování dalších použitých metod. Jsou využity celkem tři porty, jeden vstup a dva výstupy. Jejich vlastnosti jsou nastaveny jako převzaté. Získavají je tedy podle jiných portů, ke kterým jsou připojeny.

Metoda SetInpPortFrameData

Tato metoda slouží pouze ke zprovoznění práce s více vstupy nebo výstupy.

Metoda Output

V této metodě dochází k veškeré komunikaci s kartou a zpracování dat. Zahájení komunikace musí probíhat zde, protože během ní dochází k vytvoření parametru Handle, který je přiřazen konkrétnímu spojení. Problémem S-Funkcí je, že si jednotlivé metody tento parametr neumí mezi sebou předat. Je tedy nutné ho vytvořit pokaždé, když chceme komunikovat. Během tohoto procesu se musí postupně:
• načíst knihovna LJUDDotNET pomocí příkazu *NET.addAssembly('LJUDDotNet');*.
• vytvořit objekt *ljudObj = LabJack.LabJackUD.LJUD;*, pomocí kterého přistupujeme ke kartě
•příkazem *[~, ljhandle] = ljudObj.OpenLabJackS('LJ_dtU3', 'LJ_ctUSB', '0', true, 0);*

připojit první LabJack připojený k USB a vytvořit parametr handle (zde pojmenovaný ljhandle).
Všechny další příkazy použité k ovládání karty jsou pak psány formátem ljudObj.funkce(). Funkce využité v tomto případě jsou AddRequestS, ePutS, GoOne, GetFirstResult a GetNextResult, které jsou popsané v kapitole 2.3.1.
•Aktivace výstupu DAC se provede funkcí AddRequestS, kde na pozici IOType bude příkaz *LJ_ioPUT_DAC*. Číslo kanálu se předává v parametru Channel, žádaná hodnota v parametru Value a ostatní parametry necháme nulové.
•Čtení hodnot z porů AIN je podobné, jen se na pozici IOType využije příkaz *LJ_ioGET_AIN* a hodnota Value se nechá 0.
•Pro uložení získaných hodnot se pak musí vytvořit strukturované pole [ljerror, ioType, channel, dblValue, dummyInt, dummyDbl], do kterého postupně ukládáme výsledky příkazy GetFirstResult a GetNextResult s jediným nenulovým parametrem ljhandle. Výsledky získáváme podle pořadí zadání požadavků (zde tedy zápis na DAC1, zápis na DAC0, čtení z AIN0 a čtení z AIN1). Z výsledků pro zapisování není třeba nic ukládat, z výsledků čtení nás pak zajímá konkrétně hodnota dblValue. Tu si tedy po získání daného výsledku uložíme do proměnné inAIN0 nebo inAIN1.
•Nakonec jsou tyto výsledky předány výstupním porům pomocí příkazu *block.OutputPort(i).Data = x;*. [5] [2]

Metoda Terminate

V tomto způsobu řešení je zde metoda Terminate pouze pro vynulování výstupů, aby nebyla úloha pod napětím když to není potřeba. Bohužel i zde je nutné zahájení komunikace samostatně.

6.2 S-funkce pomocí stream módu

Kód je k dispozici v příloze 2.
Tento přístup je o něco komplikovanější, ale umožní nám hlídat časový krok pomocí hodin v laboratorní kartě, které jsou méně náchylné ke zpoždění (u PC závisí na vytížení systému) a dostaneme v každém časovém kroku více hodnot, díky čemuž budou data méně náchylná na šum a výpadky kontaktu (použita funkce filloutliers metodou center a zprůměrování hodnot). Hlídání udržení reálného časového kroku je zajištěno díky vlastnosti stream módu čekat na potřebný počet dat. Časový krok je tedy nastaven pomocí snímkovací frekvence a požadovaného množství dat (v mém případě jsem využil snímkovací frekvenci 10000 vzorků za sekundu a požadovaný počet skenů na 100, tedy časový krok 0.01 s). Pro správnou funkčnost musí být stejně velký časový krok nastaven i v Simulinku. Pokud je toto zajištěno, pak čas ukázaný Simulinkem odpovídá reálnému času v sekundách.

Metoda Setup

Kromě nastavení S-funkce a deklarace metod, které jsou stejné jako v předchozím případě, se zde v metodě Setup navíc nastavuje stream.

•Připojení karty a knihoven probíhá opět stejným způsobem a stejnými příkazy.

•Stream je pak nastaven pomocí příkazu AddRequestSS a parametrů uvedených v kapitole 2.3. Použitím *LJ_ioPUT_CONFIG* jako parametru IOType a parametrů pro Channel popsaných v kapitole 2.3.3 je nastavena frekvence skenování (zde na 10000), velikost mezipaměti(na 5 s) v kartě a metoda čekání na data (na SLEEP).

•Pomocí dalších parametrů IOType jsou přidány kanály, které chceme streamovat (v tomto případě AIN0 a AIN1). Číslo kanálu je předáno jako parametr Channel.

•Dále je zde zapnut výstup pro napájení obvodu (DAC1).

•Příkazem GoOne jsou pak vykonány všechny výše popsané požadavky.

•Nakonec je zapnut Stream *eGetS(ljhandle, 'LJ_ioSTART_STREAM', 0, 0, 0);* [5] [2]

SetInpPortFrameData

Zde se opět jen nastavuje používání více portů.

Output

Stejně jako v předchozím případě musí dojít i zde k opětovnému připojení karty. Naštěstí tato akce nepřeruší běžící stream, ani nezmění jeho nastavení. Stejným způsobem jako ve variantě Command/Response je zde vyřešeno i odesílání dat na výstup DAC0. Změna nastává u sběru dat.

•Nejprve je nutné vytvořit dostatečně velkou proměnnou, do které dudeme data z mezipaměti na kartě ukládat. Data získáváme fe formátu System.Double, ke kterému matlab neumí správně přistupovat, bude tedy nutné je později upravit.

•Data získáme příkazem
eGetPtr(ljhandle, LJ_ioGET_STREAM_DATA, LJ_chALL_CHANNELS, pocetskenu, data);,
který říká programu aby získal data ze všech streamovaných kanálů, vyžádal si jich určité množství (pocetskenu) a uložil je do proměnné data. variantu Ptr je nutno použít protože nechceme využívat posledníhoo parametru x1. Získaná data jou uložena ve tvaru [ch0(1), ch1(1), ch0(2), ch1(2)...], tedy liché pozice pro kanál 0 a sudé pro kanál 1. Je tedy nutné data nejenom převést do formátu čitelného matlabem, ale také je roztřídit podle kanálů. Finální úpravou dat je vyřazení vyčnívajících hodnot a zprůměrování (viz obr. 6) [5] [2]

Rozdělení dat pro jednotlivé kanály

```
%transformace dat do formátu srozumitelného pro MATLAB
Datamat = double(data);
%předvytvoření polí pro jednotlivé kanály (kvůli výůočetní rychlosti
CH0 = zeros(1, pocetskenu/2); %ch0
CH1 = zeros(1, pocetskenu/2); %ch1
%rozdělení podle sudosti indexu
for i=1:pocetskenu
    if rem(i,2)
        CH0(floor(i/2 + 1)) = Datamat(i);
    else
        CH1(i/2) = Datamat(i);
    end
end

%zprůměrování hodnot získané při jednom kroku (alespoň částečné vyhlazení)
inAIN0 = mean(filloutliers(CH0,'center'));
inAIN1 = mean(filloutliers(CH1,'center'));

%předání hodnot získaných z LJ výstupům z S-funkce
block.OutputPort(1).Data = inAIN0;
block.OutputPort(2).Data = inAIN1;
```

Obr. 6: Zdrojový kód pro zpracování dat

Tento algoritmus nám zajišťuje rozdělení dat podle jednotlivých kanálů do proměnných CH0 a CH1. Tyto data jsou poté zbavena vyčnívajících hodnot, které jsou nahrazeny hodnotou středí (funkcí filloutliers) a zprůměrována. Tím je získána jedna hodnota, kterou lze poslat na výstup S-Funkce. Díky tomuto zpracování jsou data méně náchylná na šum a je větší šance, že nenastane výpadek signálu. To je konkrétně u úlohy kulička na tyči častý problém.

Metoda Terminate

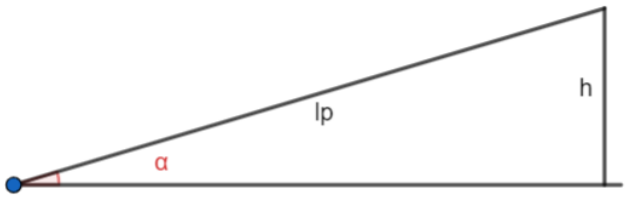
V této metodě musíme navíc kromě vypnutí výstupů navíc zastavit stream.

7 Uživatelské rozhraní pro laboratorní úlohu

Vzhledem k tomu, že přímo v S-funkci je řešeno pouze odesílání a získávání dat v podobě napětí, je potřeba pro uživatelské rozhraní úlohy zajistit přepočet požadovaného úhlu náklonu získaného z PID regulátoru na napětí, které posíláme do pneumatického ventilu a následně přepočet získaných napětí na polohu kuličky. Pro každý z těchto úkonů jsem vytvořil subsystém v programu Simulink.

7.1 Přepočet úhlu na napětí

V tomto přepočtu vycházíme ze znalosti geometrie zdvižného mechanismu a jednotlivých krajních poloh pro 0 a 5 voltů. Víme, že poloha umístění zdvižného pístu je lp = 100 mm a rozdíl krajních poloh od středu je 20 mm.



Obr. 7: Geometrie naklápěcího mechanismu

Pak tedy lze z požadovaného úhlu získat požadovaný zdvih:

$$h = sin(\alpha) \cdot l_p \tag{6}$$

Výšku musíme přepočítat na požadované napětí a to vytvořením soustavy rovnic z předpisu

$$U = a + b \cdot h \tag{7}$$

a znalosti že pro 5 V odpovídá zdvih 20 mm a pro 0 V -20 mm. Tedy:

$$0 = a - 20 \cdot b \tag{8}$$

$$5 = a + 20 \cdot b \tag{9}$$

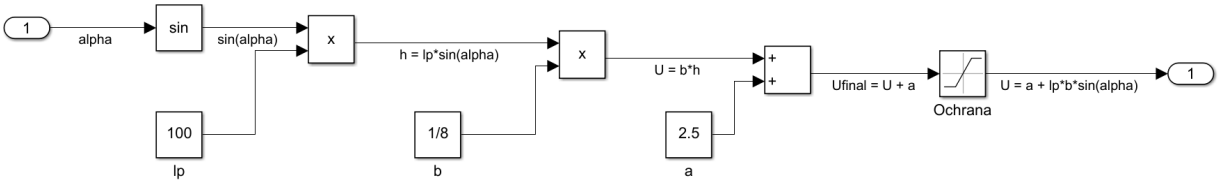
Jejím řešením získáváme hodnoty $a = 2.5$ a $b = \frac{1}{8}$.
Celá rovnice pro získání napětí je tedy obecně

$$U = a + l_p \cdot b \cdot sin(\alpha) \tag{10}$$

a po dosazení hodnot pro tuto úlohu

$$U = 2.5 + 100 \cdot \frac{1}{8} \cdot sin(\alpha) \tag{11}$$

Zanesení tohoto přepočtu do prostředí Simulink jsem pak vypracoval takto:

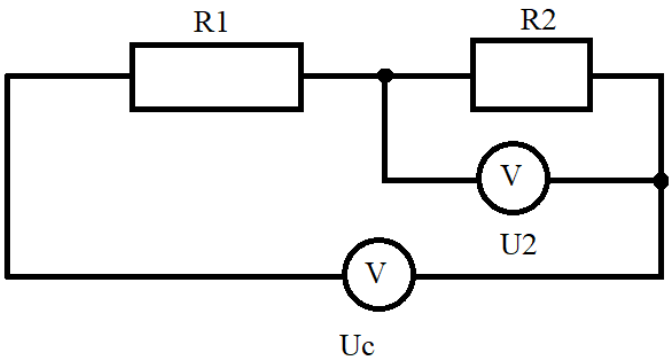


Obr. 8: Přepočet úhlu na napětí v Simulinku

Tento subsystém je k dispozici v příloze 3.
Jednotlivé parametry odpovídají parametrům ze získané rovnice a není problém je při změně podmínek upravovat. Poslední blok označený jako ochrana zajišťuje udržení napětí v hranicích 0 až 5 voltů i při požadavcích na větší náklon.

7.2 Přepočet napětí na polohu

Pro určení polohy kuličky si můžeme proměnlivý odpor rozkreslit jako dva odpory, vymezené polohou kuličky.



Obr. 9: Náhradní schéma rozdělení odporového drátu kuličkou

pak velikost jednotlivých odporů lze za pomocí vztahu

$$I = \frac{U_c}{I} \tag{12}$$

vyjádřit jako

$$R_2 = \frac{U_2}{I} = \frac{U_2}{U_c} R_c \tag{13}$$

$$R_1 = \frac{U_1}{I} = \frac{U_1}{U_c} R_c = \frac{U_c - U_2}{U_c} R_c \tag{14}$$

kde

$$R_c = \rho \cdot l_c \tag{15}$$

$$R_1 = \rho \cdot l_1 \tag{16}$$

$$R_2 = \rho \cdot l_2 \tag{17}$$

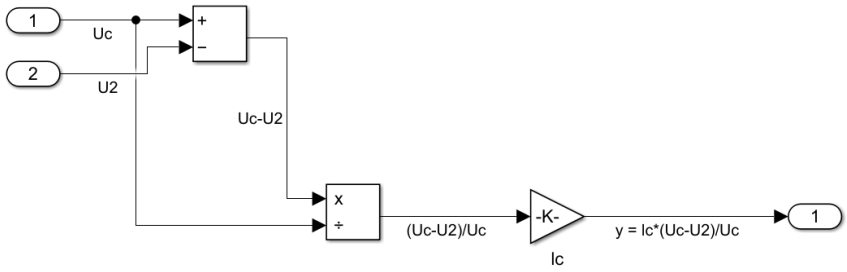
kde ρ představuje měrný odpor odporového drátu, $l_{1,2}$ jsou délky jednotlivých úseků a l_c je celková dělka drátu.

Po dosazení a vykrácení ρ dostáváme vztahy pro délky obou úseků

$$l_1 = \frac{U_c - U_2}{U_c} \cdot l_c \tag{18}$$

$$l_2 = \frac{U_2}{U_c} \cdot l_c. \tag{19}$$

Nás konkrétně zajímá délka úseku l_1, která nám udává polohu kuličky od okraje, na kterém je připojeno napájení. Převedení rovnice (18) do prostředí Simulink pak vypadá takto:



Obr. 10: Přepočet získaných napětí na polohu

Tento subsystém je k dispozici v příloze 4

7.3 Náhrada matematického modelu v URO

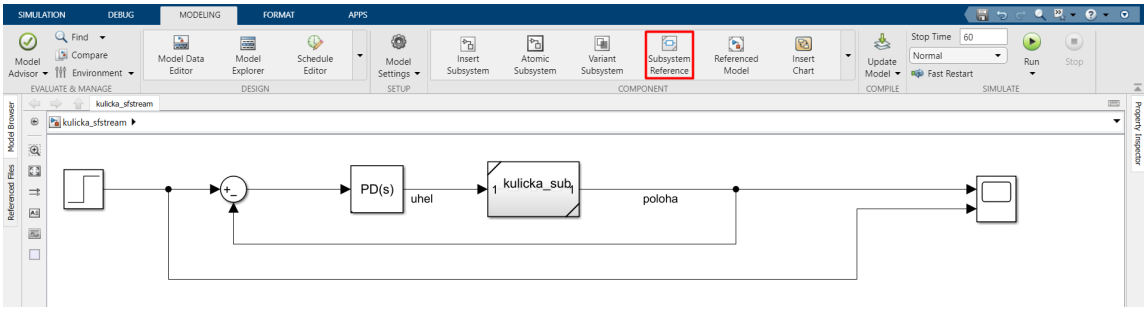
Pro jednoduchou implementaci do URO, který bude stejný jako URO pro simulovanou verzi této úlohy je potřeba vytvořit blok, který můžeme vložit místo matematického modelu. Aby bylo možno použít stejně se chovající řídící obvod, musíme mít na vstupu i výstupu stejný typ hodnot jako v simulovaném případě. Tedy vstup do bloku musí být požadovaný úhel náklonu a jako výstup potřebujeme polohu kuličky. K tomu využijeme výše vytvořené subsystémy a jejich spojením získáme subsystém plně nahrazující matematický model.

Obr. 11: Subsystém pro náhradu matematického modelu

Tento subsystém je k dispozici v příloze 5
Pro verzi v módu Command/Response se pouze změní název S-Funkce na LJCR.m a do celého modelu se přidá blok Real-Time Sync.

7.4 Zasazení do URO

V URO se pak pouze místo matematického modelu načte subsystém pomocí funkce Subsystem reference, která se nachází v Simulinku v záložce MODELING (na obrázku 12 zvýrazněno)

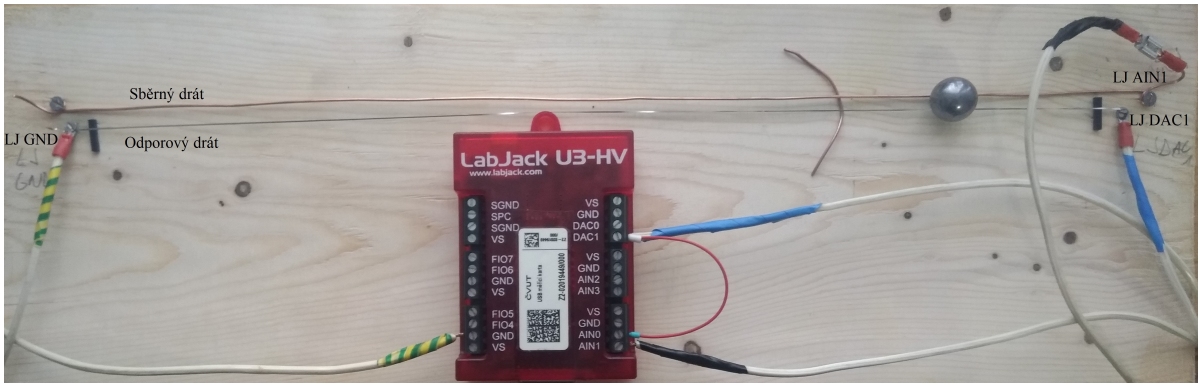


Obr. 12: Přidání subsystému do modelu v Simulinku

Model s již vloženým subsystémem pro metodu Command/Response je k dispozici v příloze 6 a pro metodu Stream v příloze 7.

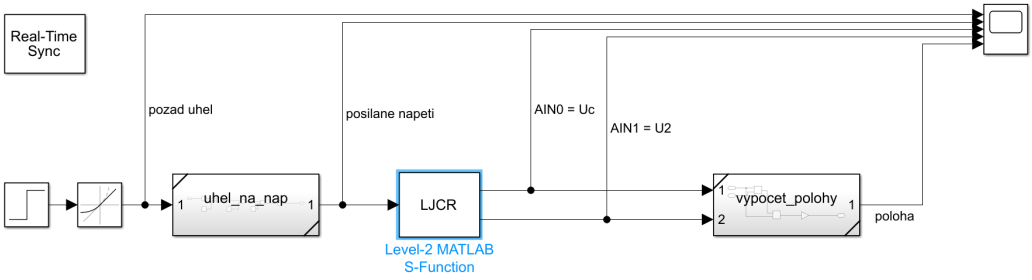
8 Test S-funkce

Vhledem k současné situaci pro mne bylo složité se dostat do laboratoří, takže jsem nemohl otestovat s-funkci přímo na laboratorní úloze. Kartu LabJack jsem tedy měl půjčenou a správné fungování funkce jsem musel ověřit na testovacím zapojení na obrázku 13, které zapojením odpovídá skutečné úloze (viz obr. 5) a na upraveném skriptu v Simulinku. Pro každou variantu S-funkce musel být skript nepatrně upraven, ale v obou případech slouží k odesílání postupně se měnícího signálu zastupující požadovaný úhel naklonění, který bych jinak získal z PID regulátoru a k zobrazení hodnot získaných z karty.



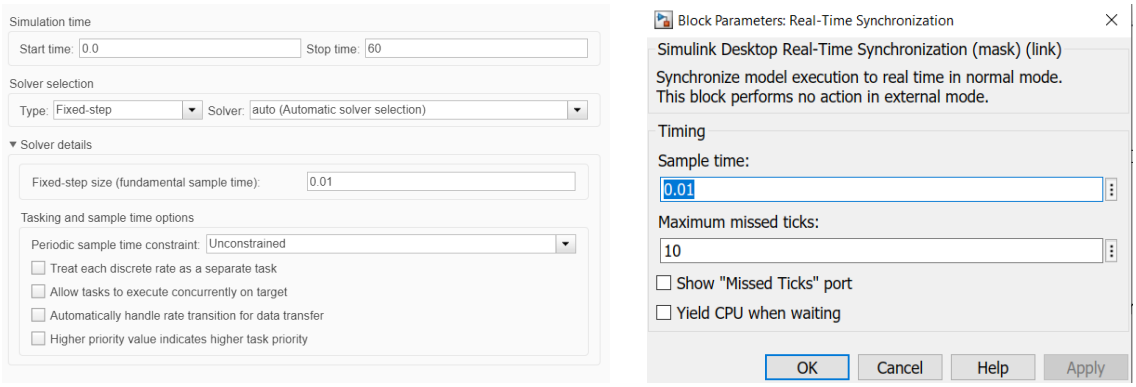
Obr. 13: Testovací zapojení

8.1 Test Command/Response módu



Obr. 14: Model testovacího zapojení pro metodu Command/Response

Testovací model je k dispozici v příloze 8
Jak již bylo zmíněno, čas simulace hlídá Real-Time Sync blok, který se do modelu umisťuje bez připojení. Ten musí být nastaven na stejnou vzorkovací periodu jako model.



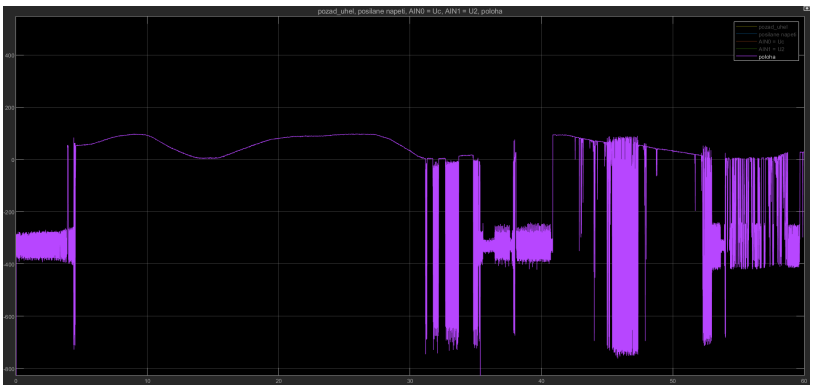
Obr. 15: Nastavení vzorkovací frekvence a Real-Time Sync bloku pro metodu Command/Response

Za těchto podmínek jsem nechal aplikaci běžet po dobu 60 a 300 sekund a ani na delším časovém úseku není časová odchylka nijak výrazná. Takto dobrých výsledků bylo dosaženo i přes to, že testy probíhaly na pozadí a mezitím jsem na počítači prováděl jiné úkony. U slabších počítačů by ale výsledky nemusely být tak slibné. První kolonka je pro test na 60 s, druhá na 300 s a poslední (modře zvýrazněná) je srovnávací test nastavený na 300 s bez bloku Real-Time Sync.

Path	Total Time (s)
zkouskaCR	60.381
zkouskaCR	300.109
zkouskaCR	119.849

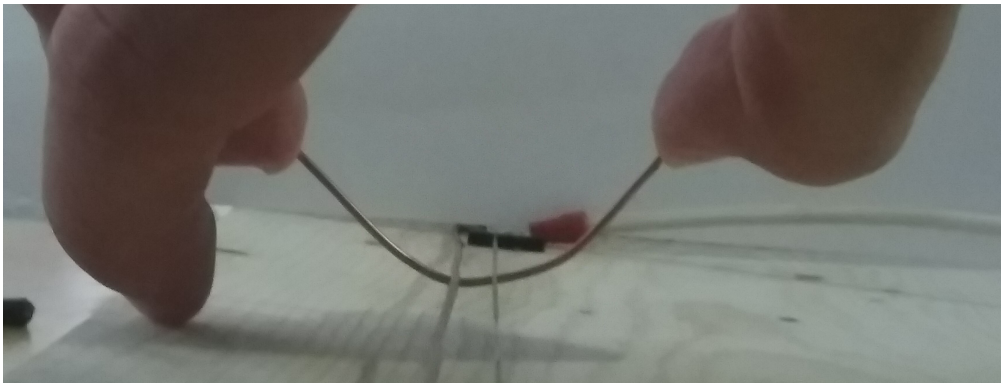
Obr. 16: Reálné časy chodu programu pro metodu Command/Response

Jako další test jsem hlídal zobrazení polohy. Pro jednoduchost jsem nechal polohu vykreslovat od 0 (propojeno v místě připojení napájení) do 100 (propojeno v místě připojení GND).



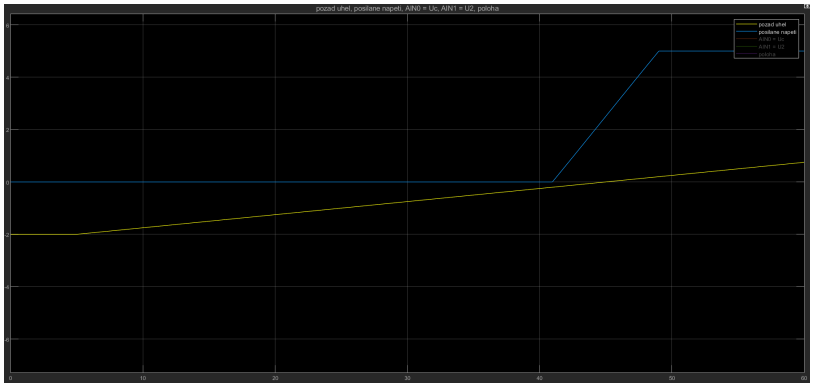
Obr. 17: Snímání a přepočet polohy pro metodu Command/Response

Zhruba do poloviny testu jsem dráty propojoval tak, aby byl kontakt kvalitní a bylo možné otestovat celou škálu viz 18. Tato část pokusu ověřila, že pokud drátem pohybuji od začátku do konce stupnice, tak zaznamenávaná poloha odpovídá skutečné. Poté jsem vyzkoušel i propojení pomocí kuličky. Je vidět, že pokud je kulička na dobrém místě, nebo ji přitlačím, tak je signál použitelný. Bohužel v mých podmínkách jsem nebyl schopen sestavit testovací žlábek, jako je ve skutečné úloze, který by zajistil lepší kontak kuličky a vylepšení dat.



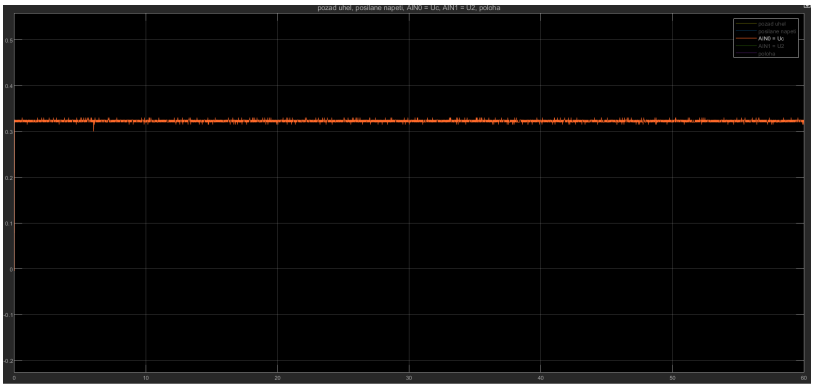
Obr. 18: Propojení odporového a sběrného drátu

Dalším testem bylo ověření přepočtu požadovaného úhlu a zamezení požadovanému napětí překročit meze 0 a 5 voltů. Tento test proběhl podle očekávání bez problémů. Shodu hodnoty posílané Simulinkem a honoty naměřené na kartě jsem ověřoval pomocí digitálního multimetru.



Obr. 19: Požadovaný úhel v radiánecha a odesílané napětí pro metodu Command/Response

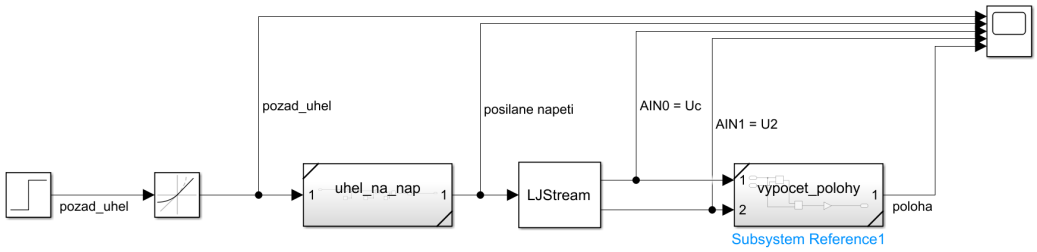
Jako poslední test jsem zkoumal kvalitu signálu, o kterém vím, že má být konstantní.



Obr. 20: Konstantní signál pro metodu Command/Response

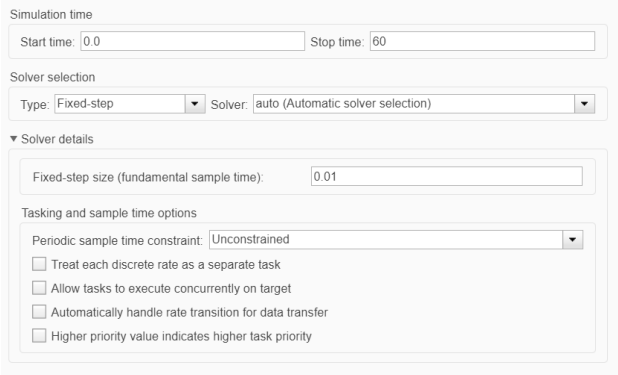
Signál nevykazuje žádné extrémy, ale také není úplně nejkvalitnější.

8.2 Test stream módu



Obr. 21: Model testovacího zapojení pro metodu stream

Testovací model je k dispozici v příloze 9
V tomto režimu nenní potřeba Real-Time Sync bloku, protože čas chodu programu je hlídaný pomocí karty. Nastavení vzorkovací periody musí odpovídat hodnotě vytvořené v kódu funkce jako $\frac{\text{počet skenů}}{\text{frekvence skenování}}$, což jsem nastavil jako 0.01 s.



Obr. 22: Nastavení vzorkovací frekvence pro metodu stream

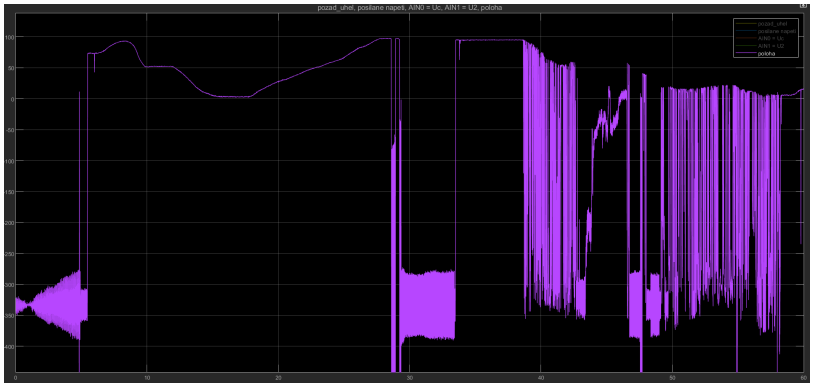
Za těchto podmínek jsem opět nechal aplikaci běžet po dobu 60 a 300 sekund a ani zde nejsou časové výchylky nikterak závažné. Pro tento mód nehraje roli další vytížení počítače, protože

pozdržení každého kroku je zajištěno pomocí LabJacku. Kontrolní vzorek bez hlídání času nelze vytvořit, protože pro jeho fungování by se musela změnit struktura získávání hodnot a tento vzorek by pak neměl žádnou vypovídající hodnotu.

Path	Total Time (s)
zkouskastream	60.327
zkouskastream	299.640

Obr. 23: Reálné časy chodu programu pro metodu stream

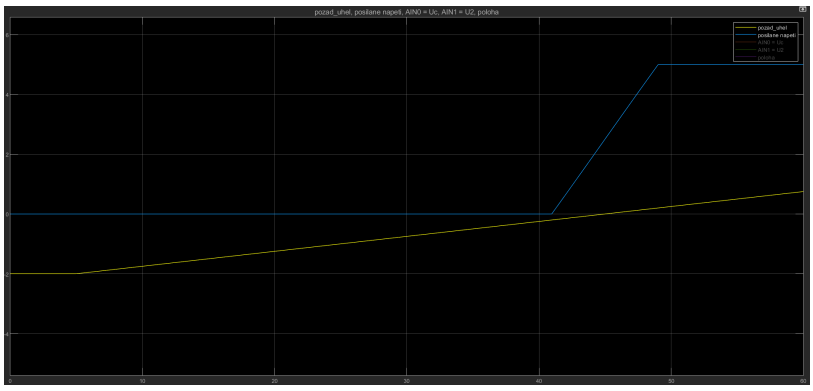
Dále jsem opět hlídal zobrazení polohy v rozmezí od 0 do 100 jako v předchozí metodě.



Obr. 24: Snímání a přepočet polohy pro metodu stream

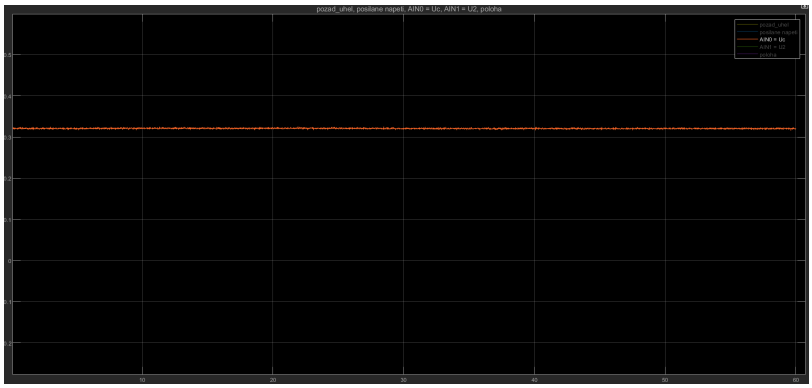
Stejně jako předtím jsem zhruba do poloviny testu propojoval dráty ručně. Zde je opět vidět záznam polohy v celém rozsahu a signál je o něco kvalitnější než v předchozí metodě. Poté jsem vyzkoušel i propojení pomocí kuličky. Na místech s dobrým rozestupem drátu jsou i data z kuličky dobře čitelná, pro místa s horším kontaktem je zde alespoň lépe vidět poloha kuličky jako horní hranice zašuměného signálu.

Ověření přepočtu probíhalo opět stejným způsobem a vzheldem k tomu, že je hodnota zapisována stejně, jsou dle očekávání shodné i výsledky.



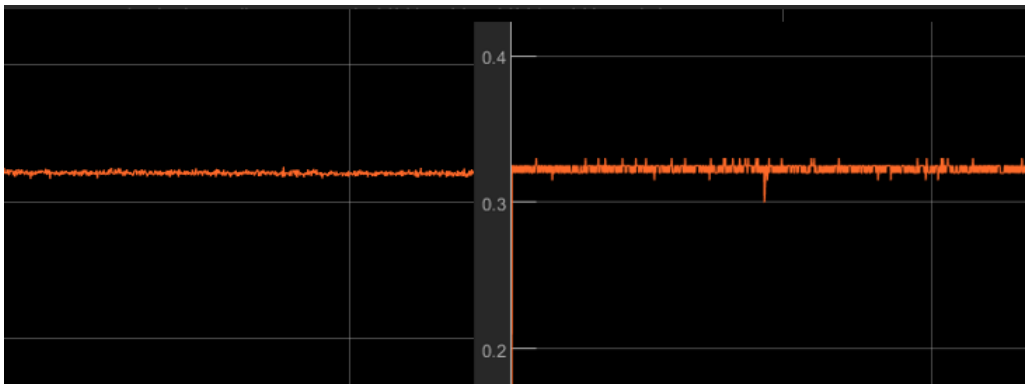
Obr. 25: Požadovaný úhel v radiánecha a odesílané napětí pro metodu stream

Jako poslední pokus byl opět na konstantním signálu.



Obr. 26: Konstantní signál pro metodu stream

Stupnice zobrazení je shodná s obr 20 a je zřetelně vidět, že díky většímu množství dat za jeden časový krok, jejich částečnému vyčištění a zprůměrování je signál získaný metodou stream kvalitnější. Pro porovnání jsem oba signály také umístil do jednoho obrázku (vlevo stream, vpravo Command/Response)



Obr. 27: Porovnání signálů z obou metod (vlevo Stream, vpravo Command/Response)

9 Závěr

Mým úkolem bylo v této práci vytvořit Level-2 S-Funkci pro ovládání laboratorní karty LabJack. Funkci jsem vytvořil pomocí dvou metod, které jsem pak v další části porovnával. Z výsledků testů lze usoudit, že vhodnějším řešením bude metoda Stream a to jak kvůli kvalitnějším datům, tak i lepší stabilitě a nezávislosti na dalším simulinkovém modulu. Během testů metody Command/Response jsem také narazil na pár problému s blokem Real-Time Sync. Jeho velkou slabinou je samotné prostředí Simulink, konkrétně zpomalení chodu systému pokud například otevírám blok Scope, nebo v něm např. měním měřítko. Z tohoto důvodu bych jednoznačně doporučil do úlohy implementovat variantu Stream. Tyto testy byly kvůli současné situaci provedeny na náhradním zapojení mimikující úlohu kulička na tyči, pro kterou je tato funkce primárně vytvořena. Posledním úkolem bylo vytvořit uživatelské rozhraní pro danou úlohu. To jsem vytvořil ve formě subsystému pro platformu Simulink, který se jednoduše vloží na pozici matematického modelu v URO této soustavy. V tomto subsystému jsou řešeny přepočty hodnot vytvářených modelem na napětí a zpět. Díky tomu, že jsou tyto výpočty řešeny až v rozhraní Simulink, lze vytvořenou S-Funkci pomocí pár jednoduchých úprav použít i na jiné aplikace. Jediné parametry, které se potenciálně mohou měnit jsou požadovaný počet portů, což lze ve funkci upravit změnou čísla v metodě setup, přidání nebo odebrání vstupů a výstupů, kterého lze docílit rozkopírováním a upravením požadavků na přidání kanálů a simulační perioda, která je upravitelná pomocí parametrů rychlsken a pocetskenu.

Seznam použité literatury a zdrojů

[1] *U3 Datasheet [online].* LabJack Corporation. c 2001-2099 [cit. 1.5.2021]. [Dostupné z: https://labjack.com/support/datasheets/u3]

[2] *Write Level-2 MATLAB S-Functions. [online]* The MathWorks, Inc. [cit. 24.3.2021] Dostupné z: https://www.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html

[3] *Simulink Engine Interaction with C S-Functions. [online]* The MathWorks, Inc. [cit. 28.3.2021], dostupné z: https://www.mathworks.com/help/simulink/sfg/how-the-simulink-engine-interacts-with-c-s-functions.html#f8-83424

[4] *Úloha 07 'Kulička na tyči' [online]* Fišer, Jaromír. [cit 25.6.2021], dostupné z: www: http://vlab.fs.cvut.cz/navody/files/kul_tyc_navod.pdf

[5] *UD Library Docs [online]* LabJack Corporation. c 2001-2099 [cit. 1.5.2021]. Dostupné z: https://labjack.com/support/software/api/ud

Seznam obrázků, grafů a tabulek

Seznam obrázků

Seznam použitého SW

- TeXstudio, TeX Live (LaTeX)
- MATLAB
- Simulink
- MSPaint

Seznam elektronických příloh

Všechny přílohy jsou kdispozici na přiloženém datovém nosiči spolu s elektronickou kopií práce.

Příloha 1: LJCR.m

Příloha 2: LJStream.m

Příloha 3: uhel_na_nap.slx

Příloha 4: vypocet_polohy.slx

Příloha 5: kulicka_sub.slx

Příloha 6: kulicka_sf.slx

Příloha 7: kulicka_sfstream.slx

Příloha 8: zkouskaCR.slx

Příloha 9: zkouskastream.slx