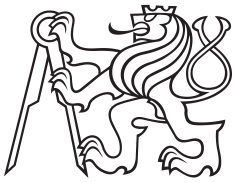
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Mobilní záznamová jednotka
Mobile Acquisition Unit

Michal Gabriel

Vedoucí: doc. Ing. Pavel Pačes Ph.D.
Obor: Kybernetika a robotika
Studijní program: Systémy a řízení
Květen 2017

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu práce, panu doc. Ing Pavlu Pačesovi, Ph.D., za příkladné vedení. Rovněž bych chtěl poděkovat rodině a přátelům za podporu, obzvláště panu Richardu Štěcovi za zpětnou vazbu během psaní práce, slečně Evičce Uhliarikové a panu Jiřímu Hodnému za jazykovou korekturu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

.....

V Praze, 25. května 2017

Abstrakt

Cílem bakalářské práce je návrh a realizace softwaru pro sběr dat ze sensoru do počítače s linuxovým operačním systémem. Textová data s časovou známkou jsou synchronně po dávkách ukládaná do souboru a současně přeposílaná přes počítačovou sít. Video data jsou ukládaná do samostatných souborů.

Klíčová slova: senzory, sber dat, linux

Vedoucí: doc. Ing. Pavel Pačes Ph.D.
Katedra pocitacu,
Karlovo náměstí 13,
Praha 2

Abstract

The aim of this bachelor thesis is design and realization of data acquisition software for computer with linux operating system. Text data with a timestamp are saved to files synchronously in batches and simultaneously sended via computer network. Video data are saved to separate files.

Keywords: sensors, data acquisition, linux

Title translation: Mobile Acquisition Unit

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Na podzim roku 2015 vznikla potřeba sbírat data z tlakových senzorů během experimentu PREDATOR programu REXUS/BEXUS. Jako zařízení pro získávání a ukládání dat byl zvolen počítač s linuxovým operačním systémem. Za tímto účelem byl v jazyce C napsán víceprocesový program, který navíc, k ukládání dat na disk, přes počítačovou síť data posílal v reálném čase, informoval o případných chybách a přijímal příkazy, umožňující například restartování jednotlivých senzorů či procesů. Aplikace pracovala převážně s textovými daty (binární data ze senzorů byla převedena na textová), která různě přeposílala a zajišťovala jejich integritu. Značná část softwaru tak nebyla specifická pro konkrétní senzor. Při dalším projektu, potřebujícím ukládání dat na disk, se proto objevila myšlenka upravit navržený systém. Mohl tak vzniknou modulární program pro sběr dat, kdy je třeba pro různé senzory implementovat obsluhu, nicméně jádro aplikace zůstává stejné. Vzhledem k tomu, že první verze softwaru byla, a nová měla být, navržena i naprogramována jednou stejnou osobou, bylo možné téma zpracovat jako bakalářskou práci.

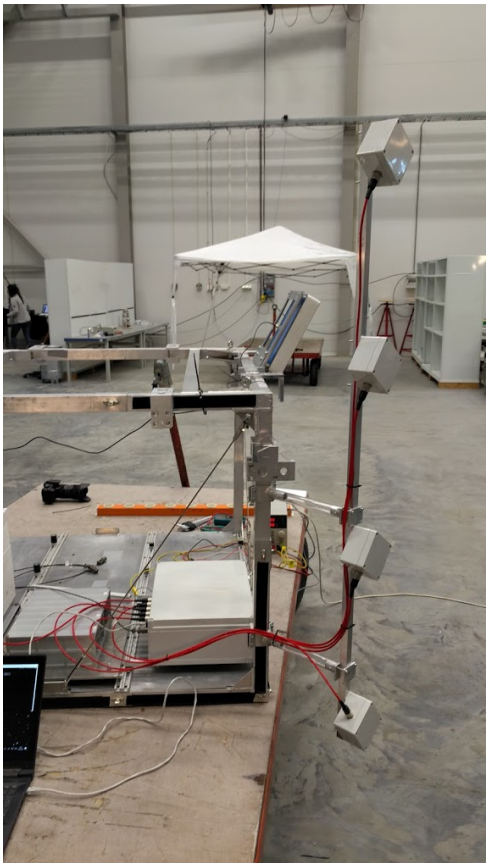
1.1 Experiment PREDATOR

Současné systémy pro určování náklonu letadel vycházející z gyroskopu vyžadují znalost přesné pozice a jsou tak závislé na signálu globálních satelitních navigačních systémů.[20] Proto jsou hledány alternativní metody pro kalibrování gyroskopu. Metodou, vycházející z nápadu Pavla Pačese, vedoucího teto práce, je určení náklonu letadla z rozdílu atmosférického tlaku

na koncích křídel. Tato metoda byla testována v laboratorních podmínkách a na UAV v nízké letové výšce. Za účelem ověření metody ve výškách relevantních při použití na letadle a získání vzorků rozdílů tlaku napříč zájmovou částí atmosféry vznikl projekt experimentu PREDATOR v rámci programu REXUS/BEXUS.

Projekty BEXUS (Balloon EXperiments for University Students) se zakládají na provedení experimentu během letu stratosférického balónu. Zorganizovány jsou dva lety ročně, na každé gondole se typicky nachází čtyři studentské experimenty. Doba letu je několik hodin a maximální dosažená výška okolo 30 km. S užitečným zatížením je udržováno rádiové spojení poskytující nestabilní počítačovou síť.[10]

Obsahem experimentu PREDATOR bylo měření tlaku v pevné stanovené vertikální vzdálenosti, s úmyslem pozorovat rozdíly mezi nimi. Systém získával data z pěti tlakových senzorů rozmístěných na gondole. Vzhledem k nemožnému fyzickému přístupu k experimentu a nezaručenému datovému přístupu během jediného pokusu o sběr dat musel být systém co nejvíce stabilní a plně autonomní.



Obrázek 1.1: Rozmístění hardwaru experimentu PREDATOR na gondole

Program REXUS/BEXUS je realizován na základě bilaterální dohody mezi německou (DLR) a švédskou (SNSB) vesmírnou agenturou. Švédský díl užitečného zatížení je dostupný pro studenty evropských zemí prostřednictvím spolupráce s Evropskou kosmickou agenturou (ESA). Za realizací letu stojí EuroLaunch - spolupráce DLR (jeho části MORABA) a Swedish Space Corporation (SSC). Studentům jsou k dispozici experti z DLR, SSC, ZARM a ESA.[7]

1.2 Kontrola ploch pro nouzové přistání

Druhým projektem, využívající tento systém sběru dat, je projekt kontroly ploch vytipovaných pro nouzové přistání. Velmi zjednodušeně řečeno, nad danou oblastí proletí letadlo zaznamenávající video oblasti pod sebou a algoritmy z obrazu určí, zda nedošlo ke změně plochy, která by znemožnila její využití pro nouzové přistání.

Kapitola 2

Realizace

2.1 Obecna cast programu

2.1.1 Modularita

Aktuální, druhá verze softwaru je zamýšlena jako modulární, co se týče použitých senzorů. Modulárnost spočívá jak

1) v obsluze jednotlivých senzorů stejného typu, kdy se rozdíl zakládá v různé adrese, tedy v závislosti na připojení prvku odlišující jednotlivé senzory (zpravidla název speciálního souboru v složce */dev*), případně i v různém nastavení konkrétních senzorů; tak
2) ve využití senzorů různých typů, vyžadující zvláštní implementace funkcí pro získávání dat.

Při přítomnosti funkcí schopných získávat data z daného druhu senzorů se před kompilací pro každý jednotlivý senzor vytvoří a naplní struktura nesoucí ukazatele na řídící funkce a různé parametry.

Pro každý druh senzoru je potřeba uvést funkce pro

- inicializaci,
- odeslání požadavku sběru dat,
- přijmutí textových dat,

- uzavření komunikace,
- nastavení senzoru
- a přímou komunikaci se senzorem.

Samozřejmě ne všechny funkce musí mít činnou implementaci, například přímá komunikace nemusí být ani relevantní v případě některých senzorů. Zmíněné funkce se zapisují do samostatných souborů a hlavičkový soubor, obsahující prototypy těchto funkcí, se zahrne na k tomu určeném místě.

Jelikož různé senzory mohou poskytovat data s různou frekvencí, lze za účelem získávání dat s různou periodou nastavit každý kolikátý běh hlavního cyklu budou data ze senzoru požadována a čtena. Je tedy nastavena délka jednoho běhu smyčky pro získávání dat a data mohou být získávána v dané násobnosti této časové doby.

Různé senzory potřebují různě dlouhou časovou prodlevu mezi určitými akcemi, ať už z důvodu hardwarové komunikace s počítačem či délky samotného odběru dat. Nastavením minimálních intervalů po inicializaci senzoru, mezi odesláním požadavku o data a čtením dat a mezi deinicializaci a inicializaci senzoru zajistí potřebný časový úsek. Správné seřazení senzorů může zlepšit využití času.

Pro rozlišení jednotlivých senzorů slouží proměnné adresy a atributy. Na obě jsou předávány senzorově specifickým funkcím adresy v paměti ukazateli obecného typu, je tak možné, aby pro různé druhy senzorů byly předávány různé typy proměnných, struktur nevyjímaje. Všechny funkce jsou volány s oběma argumenty, takže využití proměnných adres a atributů lze zaměnit, či použít pouze jednu. Typicky je jako atribut předán ukazatel na textový řetězec reprezentující speciální soubor v adresáři */dev* a v rámci inicializační funkce je do předem připravené celočíselné proměnné zapsán souborový deskriptor.

Shrnuto, při zapsání jednotlivých senzorů je třeba pro každý vytvořit strukturu a v té doplnit:

- zmíněné funkce pro obsluhu,
- násobnost periody hlavní smyčky pro odběr dat,
- minimální časové intervaly mezi různými operacemi,
- adresu specifickou pro senzor a případné atributy.

Dále struktura obsahuje pole s čítačem, kolikrát proběhla hlavní smyčka od posledního čtení ze senzoru, a časový údaj poslední operace se senzorem jakožto výchozí doba pro měřené minimální zpoždění.

V praxi lze ve vyznačeném místě v kódu nastavit předmětné senzory a vytvořit zkompilovanou verzi programu pro konkrétní využití. Během automatického obsluhování všech senzorů, tedy inicializace a periodického čtení dat, dojde k proiterování pole obsahující struktury pro každý jednotlivý senzor, z kterého mají být získávána data.

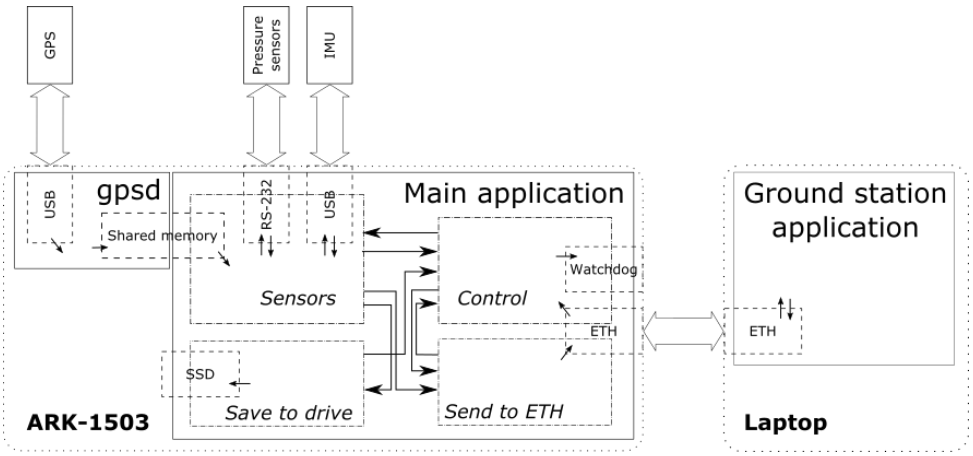
2.1.2 Rozdělení programu do procesu

Běh celé aplikace spočívá v běhu čtyř samostatných procesů. Původní, spouštěný proces vytvoří dceřiné procesy, následně slouží jako kontrolní - přijímá zprávy o stavu ostatních procesů a případně daný proces ukončí a znovu vytvoří, přeposílá chybové zprávy procesu pro odesílání přes síť a příjímá řídící příkazy ze sítě, které buď realizuje (jako příkazy pro ukončení a znovuvytvoření procesu) nebo předá příslušnému procesu (příkazy pro jednotlivé senzory). Samotná obsluha jednotlivých senzorů probíhá v druhém procesu. Jeho úloha je senzory inicializovat, generovat žádosti o data, obdržená data přečíst z příslušných rozhraní, vytvořit jednu zprávu obsahující časovou známku a data ze všech senzorů čtených v daném cyklu, tu předat procesům pro ukládání na disk a odesílání přes síť a autonomně či na příkaz ze sítě předaný kontrolním procesem řešit problémy se senzory (například restartováním či znovunastavením). Zbývající dva procesy slouží k uložení dat ze senzorů na disk a k přeposílání těchto dat a chybových hlášení přes síť.

Jednotlivé procesy mezi sebou komunikují za pomocí sedmi rour. Dvě roury přenáší data z procesu pro sběr dat do procesu pro ukládání na disk a do procesu pro odesílání přes síť, tři roury slouží pro informování kontrolního procesu o stavu procesu zbylých, jedna roura slouží pro řízení senzorů kontrolním procesem a poslední k přeposílání zpráv kontrolního procesu přes síť. Roury jsou vytvořeny před vytvořením dceřinných procesů, kterým jsou předány souborové deskriptory potřebných konců rour. Po rozdělení procesů jsou nepotřebné konce rour uzavřeny.

2.1.3 Kontrolní proces

Prvotní spuštený proces obsahující funkci main vytváří roury pro meziprocesovou komunikaci a ostatní procesy, následně kontroluje jejich stav a řeší případné problémy. Zpracovává dva různé vstupy. Prvním typem jsou roury se stavem jednotlivých zbývajících procesů, do kterých jsou zasílány periodicky zprávy potvrzující bezchybný stav procesu a případně chybová



Obrázek 2.1: Schéma původního návrhu programu

hlášení. Druhým jsou řídící příkazy přicházející přes síť z počítače umožňující vnější zásah.

Zprávy o stavu procesu

Každý ze zbylých tří procesů (proces pro sběr dat, proces pro ukládání dat ze senzoru na disk a proces pro odesílání dat přes síť) vyhodnotí na konci své hlavní smyčky chybový vektor a pokud se nevyskytuje chyba, která je nepřípustná pro další chod procesu, odešle "zprávu o bezchybném stavu (*SEN_OK, STD_OK, STE_OK*). Jelikož každá chyba mění jiný bit chybové proměnné, je možno ji přemaskovat a tím předem dané chyby ignorovat. Kontrolní proces si drží čas poslední zprávy o bezchybném stavu a v případě uplynutí časového limitu daný proces "respawnuje".

Tento systém hlídání chyby zvenku oproti rozpoznání chyby v procesu samém může pomoct i v případě, kdy samotný proces nepokračuje, jak by měl (například při nepředvídatelném zacyklení).

Chybová hlášení

Kontrolní proces přijímá veškerá hlášení o chybách, které mu přeposílají vlastními rourami jednotlivé procesy. V současné implementaci jsou zprávy přeposílány rourou procesu pro posílání přes síť a dále přeneseny do vzdáleného

počítače umožnující vnější zásah. Nicméně mechanismus je připraven pro autonomní vyhodnocení chyby a provedení odpovidajícího zásahu.

Řídicí příkazy

Z obsluhujícího počítače lze manuálně přes síť poslat příkaz pro "respawnování"některého z procesů. Další příkazy jsou přeposlány rourou procesu pro sběr dat, kde mohou způsobit například znovuspuštění některého senzoru.

Znovuvytvoření procesu

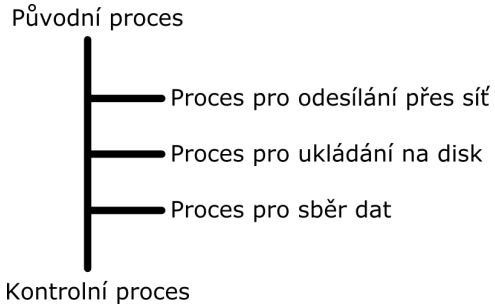
Takzvaný respawn spočívá v ukončení běžícího procesu a následném vytvoření nového procesu a spuštění dané úlohy. V případě procesu pro sběr dat ze senzoru a odesílání přes síť je ukončení provedeno vystavením signálu "SIGKILL"pro dany proces, v případě procesu na uložení dat do souboru je nejdříve vystaven signál "SIGTERM"a teprve po dané časové prodlevě i signál "SIGKILL". Zmíněné chování ma za účel umožnit procesu pro zápis dat do souboru uložit aktuální obsah bufferu do souboru.

Watchdog

Kontrolním mechanismem pro případ selhání celého systému je použití hardwarového watchdogu. Jeho funkce je obecně založena v odečítání hodnoty časovače speciálním obvodem a v případě dosazení nuly vovolání restartování operačního systému. Událost označovaná za "kopnutí psa"("kicking the dog") způsobí resetování hodnoty časovače na předem danou úrověň. Princip spočívá v resetování časovače při přípustném stavu v periodě mnohem menší než hodnota časovače. V případě přeskočení čítače instrukcí či nechtěného ukončení běhu programu dojde k vypršení časovače, restartu operačního systému a znovuspuštění aplikace.

Průběh

Proces začíná inicializací (povolením) watchdog časovače a vytvořením všech rour. Následuje vytvoření prvního dceřiného procesu systémovým voláním fork. V případě běhu nového procesu spuštění hlavní funkce daného procesu. V opačném případě znovu voláním rozvětvení a stejně jako v předchozím případě, až vzniknou tři procesy jako dceřinné jednoho původního. Ten pokračuje jako kontrolní. Před začátkem běhu hlavního kódu každého procesu dojde k uzavření nepotřebných konců rour, jejichž souborové deskriptory byly zděděny při rozvětvení procesu.



Obrázek 2.2: Zobrazení postupu rozvětvení procesů

Průběh kontrolního procesu začíná vytvořením a připojením socketu pro příchozí (řídící) komunikaci. V nekonečné smyčce následně dochází k čtení ze síťového socketu. Pokud jsou přijata nějaká data, jsou přeposlána procesu pro obsluhu senzorů, jedná-li se o příkaz vztahující se ke konkrétnímu senzoru, nebo jsou rovnou provedeny odpovídající akce. Příkazy zpracovávané přímo v kontrolním procesu jsou ve stávající implementaci příkazy pro ukončení a znovuspuštění nějakého ze tří zbývajících procesů.

V nekonečné smyčce po kontrole řídících signálů dochází ke kontrole doby poslední zprávy o bezchybném stavu procesu a případně odnastavení příslušného příznaku. Dále ke čtení z chybových rour - chybové zprávy jsou přeposlány procesu pro odesílání přes síť, zprávy o bezchybném stavu způsobí zaznamenání času přijmu a nastavení příznaku o stavu procesu. Zmíněný příznak je záhy vyhodnocen, pokud není nastaven, dojde k ukončení a znovuvytvoření daného procesu (respawn).

Poslední úkon spočívá v restartování časovače watchdogu ("kopnutí psa") - za předpokladu nastavených příznaků o zdraví všech procesů. Smyčka končí krátkým uspáním procesu.

2.1.4 Proces pro sběr dat

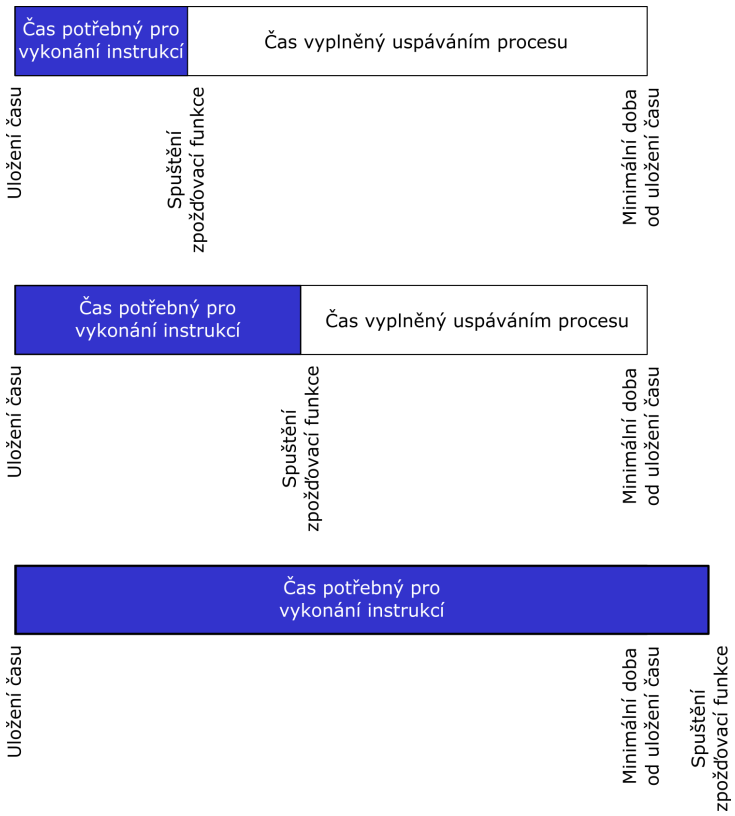
Ústředním procesem je proces obsluhy senzoru. Tento proces řeší komunikaci se všemi senzory, inicializuje je, odesílá požadavky o data a následně tato data čte, sestavuje celkovou zprávu ze všech senzorů, kterou předává procesům pro ukládání a přeposílání dat. V případě potřeby vyřizuje restarty a přenastavení senzoru, ať už automaticky, nebo na pokyn přicházející z vnějšku přes síť. V tomto procesu běží implementace specifik jednotlivých senzorů.

Zpozdovaci funkce

Potřeba přesné a neměnné délky každého průběhu hlavní smyčky neumožňuje využití prosté funkce pro uspání procesu s předem známou dobou uspání. Čas potřebný pro vykonání určité části programu může být při každém opakování těla smyčky různý. Z tohoto důvodu byla navrhnuta zpždovací funkce s dvěma argumenty - první je čas, od kterého se zadaná pauza požaduje, a druhý ceklová časová prodleva. Použití spočívalo v zaznamenání času do proměnné na začátku intervalu s požadovanou délkou a po provedení všech úkoů zavoláním této zpožďovací funkce s předáním odkazu na zmíněnou proměnnou nesoucí čas počátku předmětného intervalu. Mohla tak být zajištěna přesná minimální délka určitého úseku, čehož bylo využito nejen v zajištění přesné délky hlavního cyklu získávání dat, kde se předpokládalo vykonání předchozích instrukcí ve výrazně kratší době, ale i například pro zajištění minimální doby mezi odesláním požadavku o data a pokusu o čtení těhcto dat, kdy na času potřebném pro vykonání instrukcí mezi těmito okamžiky nezáleželo. Rozdělení operací pro více prvků, mezi kterými je potřeba zajistit časový odstup, do dvou cyklů může zefektivnit využití času - během doby nevyužitelné pro jeden prvek mohou probíhat úkony pro pvrky další.

Textový tvar proměnné typu double

Pro převedení dat uložených ve formátu s plovoucí čárkou s dvojitou přesností dle standartu IEEE 754 (typ "double"nebo též "binary64")[14] do textového řetězce používající tisknutelné znaky sady ASCII byl navržen formát, který zachovává všechny informace uložené v binární podobě. Je tak možné z textového řetězce získat přesně stejný binární obraz, včetně všech reprezentací speciálních hodnot (jako nekonečno, nečíselných hodnot - NaN) a to i se zachováním informace o původu případné chyby.



Obrázek 2.3: Náčtrt doby trvání zpožďovací funkce

Běžná čísla jsou tisknuta ve vědeckém formátu, přičemž mantisa i exponent jsou zapsána jako hexadecimální čísla a základem pro exponent je dvojka. Využití základu a číselné soustavy vycházejících z mocnin dvou snižuje nároky na výpočet, namísto operací aritmetických jsou používány operace bitové (bitový posun, bitové maskování, ...).

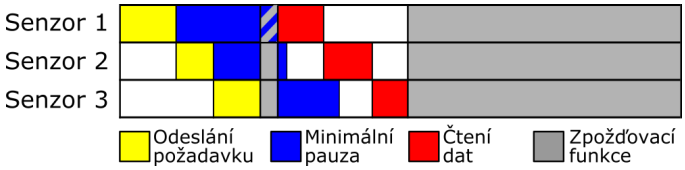
Speciální hodnoty, konkrétně tedy kladné nekonečno, záporné nekonečno, nečíselné hodnoty ("NaN"), jsou reprezentovány zvláštním textovým řetězcem.

Datový typ s plovoucí čárkou s původní velikostí osmi bajtů je tedy vytisknut do textového řetězce pevné délky dvaceti bajtů.

Průběh

V modulární verzi softwaru dojde nejprve k nastavení hodnot struktur nesoucí informace o jednotlivých senzorech. Před hlavní, nekonečnou smyčkou proces spustí postupně inicializační funkce všech zaregistrovaných senzorů

a uloží čas po ukončení funkce pro inicializaci. V další smyčce proběhnou zpožďovací funkce s hodnotami pro minimální čas potřebný po inicializaci od času uloženém v struktuře.



Obrázek 2.4: Příklad rozvržení získávání dat ze senzorů

V hlavní, nekonečné smyčce dojde hned po uložení času začátku cyklu k čtení z roury pro řídící příkazy a jejich případnému vyhodnocení. Na začátek textového řetězce, který bude obsahovat všechny textová data ze senzoru čtených v konkrétním běhu cyklu, se vytiskne časová známka - unixovy čas. Obstarání dat ze senzoru se sestává z dvou cyklů iterující všechny zaregistrované senzory. První smyčka danému senzoru odesílá požadavek pro data, druhá teprve skutečně čte data ze systémových bufferů. Toto chování šetří oproti použití jedné smyčky čas, protože během pauzy potřebné pro odeslání dat dochází k vyřizování požadavků ostatních senzorů. Čtení z daného senzoru proběhne pouze tehdy, pokud počet průběhů hlavní smyčky od posledního čtení odpovídá předem nastavené konstantě - různé typy senzorů mohou být čtené s různou frekvencí. Celý sestavený řetězec je zapsán do rour, čtených procesy pro ukládání a síťové přeposílání dat. Pokud předchozí operace proběhly bezchybně, je o tomto poslána zpráva řídicímu procesu. Zbytek smyčky je vyplněn během zpožďovací funkce pro zajištění přesné délky jednoho cyklu.

2.1.5 Proces pro ukládání dat na disk

Ukládání textových dat na disk je řešeno vlastním procesem. Proces pro sbírání dat ze senzoru data odešle do roury, jejíž druhý konec je obsluhován právě procesem pro ukládání dat do souboru, který řeší vytváření rozdělení dat do souboru, vytváření těchto souborů, zápis do nich a nakonec uzavření souboru.

V otázce volby velikosti a s tím nepřímou úměrou spjatým počtem souborů hrály roly dvě skutečnosti. Menší sobuory snižují riziko, přesněji řečeno, omezují následky při poškození jednoho souboru. Na druhou stranu operace přistupující k disku jsou časově náročné v porovnání s běžnými operacemi pracujícími s daty z operační paměti. Žádoucí je minimalizovat čas otevření

souboru, při kterém existuje zvýšené riziko poškození integrity dat při náhlém ukončení běhu operačního systému. Výsledným řešením jest zapisování dat z příchozí roury do bufferu, jehož obsah se po dosažení určitého zaplnění kapacity zapíše do souboru. Všechny souborové operace (vytvoření, zapsání a uzavření) probíhají okamžitě za sebou, minimalizuje se tak čas, kdy je soubor pro tento proces otevřen. Při zastavení běhu systému by tak došlo ke ztrátě dat uložených v bufferu, nikoliv však dat již zapsaných do souboru.

Duplicita souboru

Pro případ poškození souborů jsou data ukládána duplicitně. Pro každou sadu souborů je jeden buffer, do kterého se zapisují data přicházející z roury. Zapisování příslušných souborů pak probíhá ve dvou různých složkách. Při inicializaci se nastaví jedna z proměnných informující o zaplnění bufferu na polovinu kapacity. To způsobí časový posuv vyprázdňování bufferu a tím i odlišné hranici dat v sadách souboru. Obsah souboru tak lze v časové oblasti přirovnat k překrývání cihel. Rozloží se tak souborové operace a neprobíhá zápis do dvou souborů hned po sobě. Při náhlém vyřazení systému z provozu dojde ke ztrátě dat odpovídající obsahu maximálně poloviny bufferu (data budou zapsána jen v jednom souboru).

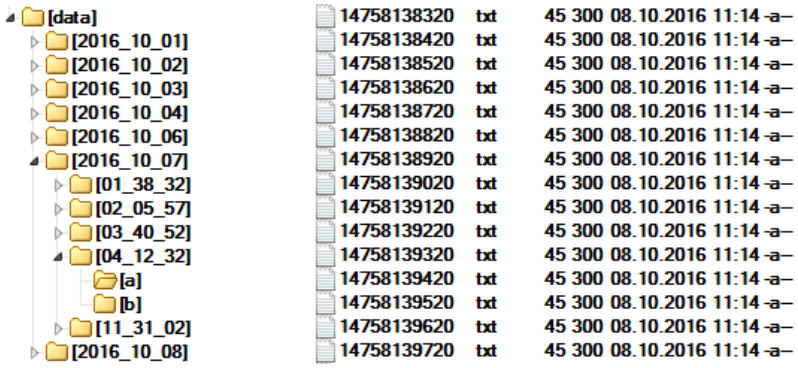
Soubor	Číslo vzorku																			
A	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
B	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20

Obrázek 2.5: Znázornění překrývání obsahu duplicitních souborů

Struktura složek a jména souborů

Všechny datové soubory jsou ukládány do složky data umístěné (či vytvořené) v aktuální složce. V adresářové struktuře následuje složka označující den spuštění programu a složka označující čas spuštění. Pro každou z dvou kopií dat je určena jedna složka - složka /textita a složka /textitb. V těchto složkách pak jsou přítomny jednotlivé textové soubory obsahující textová data ze senzoru. Jejich jméno je vytvořeno z časové známky - unixového času a pořadového čísla souboru vytvořeného v dané sekundě (první pořadové číslo jest 0). Existence souboru s nižším pořadovým číslem je kontrolována cyklem a funkcí access, která informuje o přístupových právech aktuálního uživatele k danému souboru, popřípadě právě o existenci souboru. Jako zabezpečení proti zacyklení slouží omezení maximálního počtu pokusů, poté je text připojen k

poslednímu zkoušenému souboru. Textové řetězce jsou generovány z času v časovém pásmu UTC+0 (GMT).



Obrázek 2.6: Zobrazení adresářové struktury pro textová data

Průběh

Hlavní smyčka tohoto procesu čte ze vstupní roury vždy při zapsání dat procesem pro sběr dat. Zapsání do roury vyvolá návrat z funkce poll a připsání dat z roury do dvou bufferů. Pokud dříve dojde k vypršení časového limitu funkce poll, čtení neproběhne a celá smyčka probíhá od začátku. Jelikož je ukládaná pozice posledního znaku v poli, není třeba v připojovací funkci iterovat od začátku pole do nalezení znaku $\backslash 0$ označujícího konec textového řetězce. Pokud příchozí data nekončí znakem nového řádku ($\backslash n$), je tento znak přidán.

Dalším krokem je kontrola obsazenosti bufferu a případně zapsání do souboru. Data jsou do roury zapisována i čtena po řádcích, za normálního chodu tak vyvolá vyprázdnění bufferu určitý počet řádků v něm. V praxi bylo použito 100 řádků, tedy 100 sad dat ze senzoru. Pro zabezpečení dostatku místa v bufferu je zápis spuštěn i při obsazenosti určitého podílu bytové kapacity pole.

Při bezchybném průběhu těla smyčky dojde k zapsání informace o korektním stavu do speciální roury pro hlášení chyb. Díky tomu nebude proces zastaven a znovuspuštěn kontrolním vláknem.

Smyčka je ukončena přes zvláštní příznak, nastavený při ukončení roury druhým procesem (pokud ještě v rouře nezůstávají nepřečtená data). Obě skutečnosti jsou zjišťovány funkcí ze struktury upravované funkci poll. Tato

vlastnost je důležitá pro zapsání dat na konci běhu programu, kdy nejsou buffery považovány za zaplněné.

2.1.6 Proces pro odesílíní dat přes síť

Pro předávání dat v reálném čase slouží síťová komunikace. Speciální proces řeší posílání senzorových dat a chybových hlášení přes běžnou počítačovou síť pomoci User Datagram Protocol (UDP) do obsluhujícího počítače. Proces má tak dvě vstupní roury, které obsluhuje rovnocenně - z procesu pro sbírání dat a z kontrolního procesu. Cíl odesíláni je určen při kompilaci užitím maker, definována je IP adresa a port.

UDP vs. TCP

Během experimentu PREDATOR na balónu projektu REXUS/BEXUS nebylo možné zaručit stabilní spojení s pozemní stanicí, proto bylo užití UDP v protikladu k TCP (Transmission Control Protocol) jasnou volbou. Výhody TCP, tedy spolehlivost přenosu a zajištění správného pořadí přenosu, nejsou v této aplikaci vyžadovány. Neaktuální data nemají užitek, pro potřebu výsledného zpracování naměřených dat jsou uložena v souborech na disku. Naopak čas pro ustanovení spojení by působil problémy. Další výhodou UDP je nenáročnost na režii.[15]

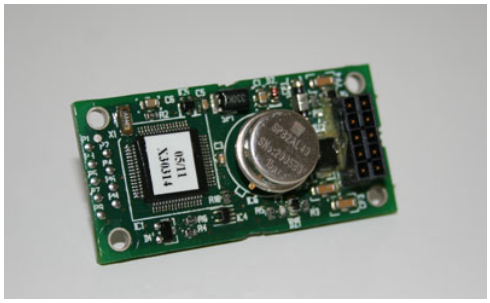
Průběh

Na začátku procesu je vytvořen socket pro datagramy. V každém běhu cyklu je pro každou ze vstupních rour samostatně realizováno čtení z roury a posílání funkcí sendto, které se jako parametr předává cílová adresa. Následuje volání funkce poll, která hlídá stav obou dvou rour. Vyhodnocení stavu struktur, na které byly funkcí předány reference, zapříčiní nastavení příznaku o přítomnosti nepřečtených dat na vstupu. Případná shoda zavěšení obou rour a nepřítomnost nezpracovaných dat v nich má za následek nastavení příznaku o ukončení a tím nespuštění další iterace hlavního cyklu.

2.2 Obsluhované senzory

2.2.1 Tlakový senzor Memscap TP3100

Hlavním senzorem, jehož data byla získávána, ukládána a odesílaná při prvním reálnem použití tohoto softwaru, byl inteligentní senzor absolutního tlaku TP3100 od firmy Memscap. Vystup samotného senzoru tlaku, čímž je zařízeni SP82 stejného výrobce, je zpracováván mikrokontrolérem. Dochází zde ke kompenzaci nelinearity i vlivu teploty. Komunikace s okolím je realizovaná výlučně asynchronní sériovou komunikaci při logických úrovních odpovídajících TTL, osmi datových bitech, jednom stop bitu s nastavitelnou paritou a rychlosti komunikace (baudratem).[17]



Obrázek 2.7: Tlakový senzor MEMSCAP TP3100, převzato z [6]

Protokol se sestává z definovaných příkazů a dotazů v textovém (ASCII) modu. Senzor zde vystupuje jako závislé zařízení (slave) - na každou jednotlivou zprávu jednou odpoví jednou zprávou (vyjma modu, ve kterém periodicky vzorkuje a odesílá data). V závislosti na nastavení může být za samotnou zprávu připojen 16bitový kontrolní součet (dle algoritmu Cyclic Reduntant Check s polynomem $x^{16} + x^{15} + x^2 + 1$), který je zapsán do čtyř hexadecimálních číslic v ASCII. Pokud je kontrolní součet povolen, a ve zprávě příchozí do senzoru součet nesouhlasí, mikroprocesor odpoví chybovou zprávou a příkaz nevýkonná. Každá zpráva se sestává z identifikace typu, případných dat a u odchozích zprav informaci o stavu zařízení.[16]

Senzor pracuje v jednom ze tří módů:

a) standartní, kdy přijímá všechny definované zprávy, včetně požadavků pro jednorázový odběr,
b) "continuous sample mode", kdy sám odesílá změřená data periodicky v předem nastaveném časovém intervalu,

c) "trigged sample mode", kdy očekává jednoznakovou zprávu (konkrétně znak konce řádku), na kterou odpoví daty s minimální prodlevou, případně zprávu o ukončení tohoto modu.

[16]

Nastavitelnými parametry senzoru jsou:

1) povoleni CRC (2 moznosti: povoleno/zakázáno),
2) výchozí mód při startu (3 moznosti: continuous off/sample/trigged sample),
3) povoleni filtru (2 moznosti: povoleno/zakázáno),
4) povolení sudé parity sériové komunikace (2 moznosti: povoleno/zakázáno),
5) rychlost sériové komunikace (4 možnosti: 4800/9600/19200/38400bd),
6) frekvence periodického zasílaní dat (10 moznosti: 20-200ms).

[16]

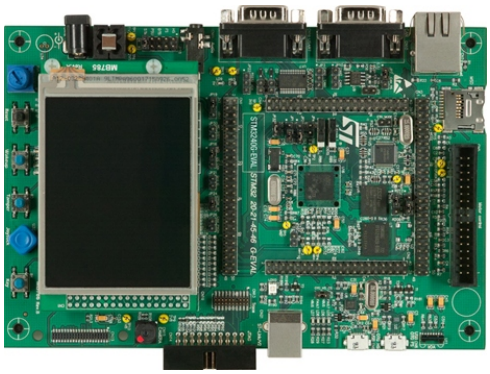
Použití senzoru

Vzhledem k cíli experimentu PREDATOR bylo zapotřebí zajistit minimální rozdíl navzorkování tlaku v jednotlivých senzorech. Za tímto účelem byl jako pracovní mód vybrán mód odpovědi na jednoznakovou spouštěcí zprávu ("trigged sample mode"). Podle specifikace by měla být odchylka začátku vzorkováni v tomto módu méně než $\pm 4\mu s$. Uvazován byl i systém, kdy by odesíláni do všech senzorů vycházelo z jednoho UARTu, což se ukázalo jako nepotřebné po změření časové prodlevy odeslaných zpráv osciloskopem. Kontrolní součet byl v nastavení povolen s tím, že ověření správnosti součtu u obdržených dat došlo až při dodatečném zpracovaní. Data byla i při chybném součtu ukládána, jiné řešeni, jako třeba nový požadavek odběru, neměla vzhledem ke krátké periodě odběru smysl. Filtr implementovaný v mikrokontroleru byl zakázán, jelikož jeho podstatou bylo průměrování posledních tří hodnot. Surová data umožňuji aplikovat stejný nebo i jiny filtr při dodatečném zpracování.

Hardwarové spojení

Chronologicky první volbou jednotky pro sběr a ukládání dat nebyl počítač s operačním systémem, ale vývojová deska STM3240G-eval osazena 32bitovym

ARM procesorem. Vzhledem k potřebě pěti nezávislých volných kanálů pro sériovou komunikaci bylo v plánu nevyužívat dedikovaný hardware (UART - Universal asynchronous receiver/transmitter), nýbrž nastavovat softwarem logické úrovně univerzálních vstupně-výstupních digitálních pinů (GPIO - General-purpose input/output).

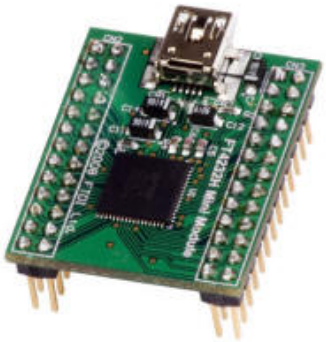


Obrázek 2.8: Vývojová deska STM3240G-eval, převzato z [8]

Za účelem snadnější obsluhy dalších senzorů, nakládání se soubory, on-line komunikace i implementace softwaru byl za řídící jednotku nakonec zvolen počítač pro vestavěné systémy od firmy Advantech - ARK-1503. Ten disponuje dvěma nativními sériovými porty, a také slotem pro mini PCIe kartu. Řešením deficitu tří sériových linek měla být právě mini PCIe karta, avšak nepovedlo se ani jednou z dvou modelů karty rozšiřující systém o čtyři sériové porty zprovoznit.

Konečným a mírně nouzovým řešením byl modul FT4232H od firmy FTDI, který prostřednictvím USB rozhraní rozšiřoval systém o čtyři UART linky s TTL logickými úrovněmi. Původní záměr vyhnout se užití USB zařízení vycházel z obsluhy USB ve vyšších vrstvách oproti PCIe a tím nižší spolehlivosti. Finálně byly dva klíčové senzory připojeny přes čip MAX232 převádějící logické úrovně TTL na úrovně specifikované ve standartu RS-232 k dvěma nativním sériovým portům počítače a zbylé tři senzory byly bez potřeby měnit logické úrovně připojeny k FTDI čipu.

Pro kontrolu sériové komunikace nebylo využito žádné řízení toku, které ani není senzorem podporováno. Rychlost komunikace byla nastavena na výchozí hodnotu senzoru - 38 400 baudu.



Obrázek 2.9: Převodník FT4232H Mini Module, převzato z [9]

Programová obsluha

Obsluha tlakových senzorů byla klíčová a také primárním účelem celého systému, proto byla naimplementována přímo do hlavní funkce procesu pro sběr dat. Nevyužívání samostatných funkcí značně znepřehlednilo zdrojový kód, na druhou stranu snížilo riziko nevykonání obsluhy z důvodů problémů na zásobníků volání funkcí.

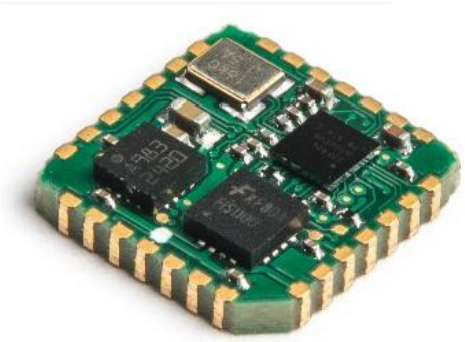
V bodě řízení sériové komunikace tkví velká výhoda využití linuxových systémů oproti prostředí operačního systému Windows. Pro sériové porty jsou v linuxových systémech vyhrazeny speciální soubory terminálu ve složce /dev (/dev/ttyS0, ...). K nastavení vlastností terminálů jako například rychlost fyzické komunikace slouží v jazyce C funkce s prototypy v hlavičkovém souboru termios.h[5]. K zapisování na linku a čtení z linky stačí použít funkce read a write, stejně jako při užívání roury nebo běžných soubor. Potřebný deskriptor souborů se získává funkcí open. Jako každý soubor by měl být po ukončení práce uzavřen funkcí close.

Při každém spuštění procesu došlo před hlavní (nekonečnou) smyčkou k inicializaci sériové komunikace. Ta spočívala v nastavení vlastností terminálu, otevření terminálového souboru a odeslání požadavků na zahájení módů pro přesný odběr dat (trigged sample mode). V každém běhu hlavní smyčky proběhly dva cykly iterující přes všechny tlakové senzory. První měl za úkol odesílat spouštěcí znak (0x0A) na jednotlivé linky, druhý pak číst obsah příjímacích bufferů jednotlivých UARTu. Rozdělením do dvou cyklů byl dosažen minimální časový rozdíl ve spuštění vzorkování jednotlivých senzor. Před samotným voláním funkce read byla volána funkce poll, jejímž úkolem

bylo počkat na příchozí data s daným časovým limitem. Pokud byla data k dispozici, byla spolu s informacemi o průběhu funkcí write a read připojena k bufferu pro data, který je odesílán na konci hlavní smyčky pomocí rour procesům pro ukládání na disk a pro odeslání přes síť. Kromě kontroly chybového hlášení o chybě v kontrolním součtu tak nedocházelo k žádnému zpracování příchozích dat, pouze k jejich předání. Pokud data nebyla v limitu k dispozici, k zmíněnému bufferu byl připojen text "No_data"a návratová hodnota read byla přepsána speciální hodnotou "-2". Pro takový případ, kdy by na začátku čtení mohly v bufferu sériového terminálu zůstat data z předchozího cyklu, která došla po limitu, byl čtecí buffer vyprázdněn před odesláním spouštěcího znaku. Případy, kdy nebyla příjmuta očekávaná data byly počítány a podle toho byla znovu odeslána žádost o začátek módu očekávajícího spouštěcí znak či příkaz restartu senzoru.

2.2.2 AHRS FMT1030

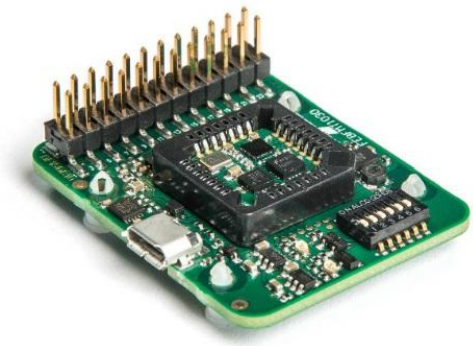
Druhým senzorem použitým při experimentu PREDATOR byla jednotka pro zjišťování inerciálních dat (AHRS - Attitude and Heading Reference System). Konkrétně modul FMT1030 firmy Fairchild Semiconductor poskytující data o úhlové rychlosti ve třech úhlech a následně také o náklonu (absolutní pozici) v těchto třech úhlech, zrychlení ve třech osách, magnetickém poli rovněž ve třech osách a teplotě. Všechny hodnoty je jednotka schopna vzorkovat s frekvencí 100 Hz, jen u teploty s maximální frekvencí 1 Hz. V případě orientačních úhlů, kvůli kterým byl senzor inerciálních veličin použit, je výrobcem udávána přesnost 3 stupně.[12]



Obrázek 2.10: AHRS modul Fairchild Semiconductor FMT1030, převzato z [12]

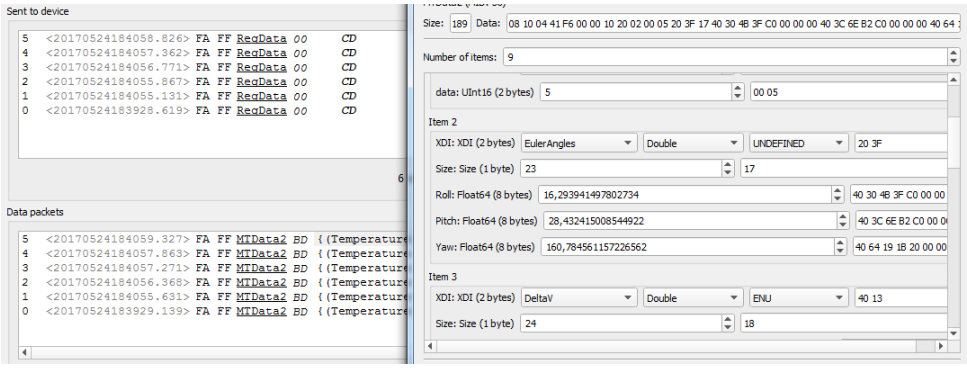
Senzor je osazen v evaluační desce označené jako FEBFMT1030_MEMS01 od stejného výrobce. Ta zajišťuje řízení spotřeby (stabilizaci vstupního napájení na potřebná nižší napětí) a komunikační rozhraní. Zatímco samotný

senzor komunikuje s okolím přes sběrnice určené pro použití na nižších úrovních (rozhraní mezi periferie a řídící modul), výčtem Inter-Integrated Circuit (I^2C), Serial Peripheral Interface Bus (SPI) a Universal Asynchronous Receiver/Transmitter (UART), evaluační kit přidává možnost komunikace přes standart RS-232 ("sériový port"- jiné logické úrovně při stejném komunikačním protokolu jako UART) a Universal Serial Bus (USB).[13]



Obrázek 2.11: Evaluační deska Fairchild Semiconductor FEB-FMT1030_MEMS01, převzato z [13]

Data jsou přenášena v binárním režimu. V závislosti na nastavení mohou být hodnoty přenášeny ve formě s plovoucí čárkou nebo ve formě s pevnou pozicí ciferní čárky.[11]

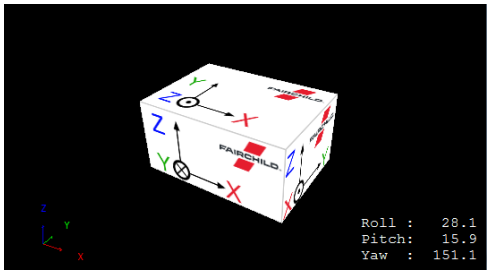


Obrázek 2.12: Prijem dat pres aplikaci Fairchild Semiconductor MT Manager

Použití senzoru

Pro dosazení synchronizace odesílání dat z jednotky a čtení v hlavní smyčce je využit princip odesláni datové zprávy na požadavek. Struktura zprávy

(zahrnuté veličiny, jejich formát a pořadí) vychází z předem určeného nastavení. V případě experimentu PREDATOR zpráva obsahovala pořadové číslo paketu, trojice hodnot (hodnoty ve třech osách) pro Eulerovy úhly, hodnoty akcelerace, úhlové rychlosti, magnetické pole; poté jednu hodnotu teploty, informaci o stavu senzoru a konečný kontrolní součet.



Obrázek 2.13: Vizualizace eulerových úhlů v aplikaci Fairchild Semiconductor MT Manage

Hardwarové spojení

Jednotka je připojována k počítači přes rozhraní Universal Serial Bus (USB). V prostředí operačního systému Windows využívá techniku virtuálního COM portu, obdobně v linuxových systémech se vstup a výstup obsluhuje stejně jako sériový port, pres speciální soubor ve složce */dev*.

Programová obsluha

S přihlédnutím ke své komplexnosti byla obsluha datové komunikace s AHRS implementována v samostatných funkcích ve zvláštním souboru již v nemodulární verzi aplikace. Implementovány byly funkce pro

- inicializaci komunikace, spočívající v nastavení a otevření sériového terminálu,
- ukončení komunikace, použitou při restartování komunikace,
- nastaveni senzoru na požadovanou konfiguraci, volanou při požadavku přes síť
- odeslání požadavku o data a
- příjem a parsování těchto dat.

Jelikož data přicházela za sensoru v binární podobě, bylo pro zachování konceptu systému potřeba data převést do textové formy. Podle identifikace

reprezentované veličiny zasílané s hodnotou byly do připravené struktury uloženy ukazatelé na místo v paměti obsahující odpovídající hodnotu. Vzhledem k opačné endianite došlo před zpracováním k prevrácení celého příchozího pole bajtů a tím byla zajištěna validita dat i při odkazování na adresu vícebajtového datového typu (v tomto případě 64bitoveho typu s "dvojitou přesnosti"-typ double [14]). Díky následnosti hodnot pro tři osy bylo možné použít na tyto data ukazatel pro za tímto účelem vytvořenou strukturu, sestávající se trojice hodnot typu double. Data adresovaná přes strukturu obsahující odkazy na hodnoty všech relevantních veličin získaných ze senzoru pak byla standartními funkcemi z knihovny stdio.h převedena do textové podoby [2].

2.2.3 GPS

Systém byl připraven i na obsluhu dat globálního navigačního satelitního systému, ačkoliv nakonec nebyl modul GPS během experimentu PREDATOR použít. Přijímač GPS odesílá do počítače zprávy dle standartu NMEA 0183, z kterých jsou kalkulovány pozice satelitu a následně i pozice přijímače. K tomuto účelu lze na operačních systémech využívat demon, tedy proces běžící na pozadí. Tyto typy programu vypočtené informace o stavu (pozici, rychlosti, ...) modulu GPS předávají dalším programům, které zpravidla nepotřebují původní zprávy. Demonem využitým v tomto softwaru byl *gpsd*, komunikující s uživatelskými programy pres síťový soket, případně umožňuje jednosměrnou komunikaci (pouze čtení) přes sdílenou paměť nebo softwarovou sběrnici D-Bus.[1]

Programová obsluha

Obsluha dat naprogramovaná v tomto softwaru spočívá ve čtení dat ze sdílené paměti plněné demonem *gpsd*. Před hlavni smyčkou je získaná adresa sdílené paměti v rámci funkce *gps_open*, poté periodicky dochází k načtení dat do struktury připravené v implementaci gpsd klienta a předmětné informace jsou vytisknuty do řetězce přidaného k textovým datům z ostatních senzorů.

2.2.4 Webkamera

Zatím jediným implementovaným zdrojem dat, jehož data nelze snadno reprezentovat jako textové řetězce je video vstup. Webkamery a podobná

zařízení poskytující živý video výstup mohou být zpracovávány v linuxových systémech pres speciální soubor ve složce */dev* [18]. Tento systém umožňuje komunikovat se všemi podporovanými zařízeními stejně. Pro obsluhu těchto speciálních souborů, zpravidla číslovaných od názvu textit/dev/video0, je již navrženo mnoho programů pro získávání a ukládání video dat v předem zadané formě[19]. Softwarem využitým v této aplikaci je program *streamer*. Chováni je ovlivněno vstupními parametry zadanými při volání programu. [3]

Programová obsluha

Pro předávání informací o konkrétním video vstupu a jeho požadovaném nastavení je vytvořena zvláštní struktura. Ve funkci pro inicializaci senzoru dojde k vyhledání složky s daty, do které jsou ukládána textová data procesem pro ukládání dat, a adresa složky je přidána do struktury. Do pole struktury pro příkaz spuštění programu streamer je předpřipraven textový řetězec, obsahující informace o zdroji dat a parametry záznamu.

Ve funkci zamýšlené pro odeslání požadavku o data je k příkazu připojen výstupní soubor, vycházející ze složky již připravené ve struktuře a aktuálního času a příkaz je vykonán voláním funkce *system*, určené pro vykonání příkazu systémového shellu[4]. Znak ampersandu připojený za samotný příkaz způsobí spuštěni požadovaného programu v nově vytvořeném procesu a zabrání tak blokování původního procesu pro získávání senzorových dat.

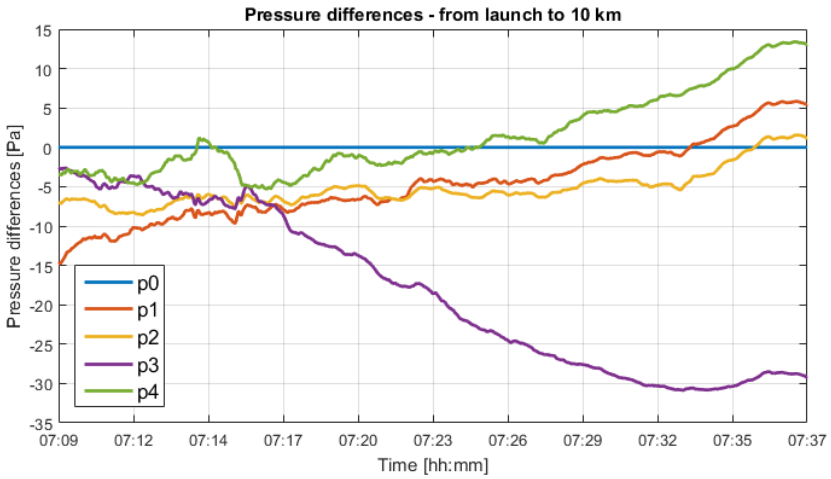
Kapitola 3

Závěr

Program byl realizován ve dvou verzích, což odráželo různé požadavky na vlastnosti systému. První verze se soustředila na spolehlivost a tomu odpovídala i její implementace. Kód této verze je dosti nepřehledný. Málo využívá volání funkcí, aby se snížilo riziko špatného volání či chyby na zásobníku volání. Tento požadavek nebyl u druhé verze tak silný, a proto se mohla realizace druhé verze více orientovat na rozšířitelnost systému a přehlednost kódu.

První verze programu byla v praxi použita v rámci experimentu PREDATOR projektu REXUS/BEXUS. Celý systém letěl jako část užitečného zatížení letu BX23, kdy získával data z pěti tlakových senzorů MEMSCAP TP3100 a AHRS jednotky Fairchild Semiconductors FMT1030, odesílal je přes počítačovou síť a ukládal na disk typu SSD (Solid State Drive). Aplikace byla automaticky spouštěna při startu operačního systému. Po celou dobu běhu program přijímal příchozí příkazy a realizoval je. Během provozu nedošlo ani k jednomu automatickému restartu. Počítač byl jednou vypnut příkazem přes SSH (Secure Shell) z důvodu přehříváni systému, poté byl bez problemu opět přiveden do chodu pomocí metody Wake-on-LAN. Data byla bez problému získána a jen výjimečně (řádově několik desítek z více než čtvrt milionu záznamů) byla poškozena a nebyla použita pro vyhodnocování.

Systém byl upraven na modulární a je připraven pro další nasazení. Pro jakýkoliv senzor, schopný dodat data na požadavek, mohou být implementovány řídicí funkce a senzor může být do systému přidán. V této chvíli jsou naprogramovány obsluhy AHRS (Attitude and Heading Reference System) jednotky Fairchild Semiconductor FMT1030 a obecných webkamer. V blízké



Obrázek 3.1: Vizualizace části dat získaných během experimentu PREDATOR

době bude aplikace nasazena do systému pro zaznamenávání dat na letadle.

Obě verze aplikace splnily velmi dobře očekávání a nejsou známy zásadní nedostatky.

Příloha A

Literatura

[1] *gpsd (8) Linux User's Manual.*

[2] *snprintf (3) Linux User's Manual.*

[3] *streamer (1) Linux User's Manual.*

[4] *system (3) Linux User's Manual.*

[5] *termios.h (3) Linux User's Manual.*

[6] *MEMSCAP TP3100*. 2012. online, http://www.memscap.com/products/aerospace-and-defense/pressure-transducers/tp-3100.

[7] *REXUS/BEXUS*. 2017. online, http://rexusbexus.net.

[8] *STM3240G-EVAL - Evaluation board with STM32F407IG MCU - STMicroelectronics*. 2017. online, http://www.st.com/en/evaluation-tools/stm3240g-eval.html.

[9] FTDI Chip. *Development Modules*. 2017. online, http://www.ftdichip.com/Products/Modules/DevelopmentModules.htm.

[10] EuroLaunch. *BEXUS User Manual*, 2 2016. Rev. 6.13.

[11] Fairchild Semiconductor. *FMT Low Level Communication Protocol Documentation*, 2015. Rev. 1.0.

[12] Fairchild Semiconductor. *FMT1000-series Motion Tracking Module with Output of Orientation, Inertial Motion Data and Magnetic Field*, 2015. Rev. 1.0.

[13] Fairchild Semiconductor. *User Guide for FEBFMT1030_MEMS01*, 2015. Rev. 1.0.

[14] IEEE. *754-2008 - IEEE Standard for Floating-Point Arithmetic*, 8 2008.

[15] Wei Liu, Carolyn Matthews, Lydia Parziale, Nicolas Rosselot, Chuck Davis, Jason Forrester, and David T. Britt. *TCP/IP Tutorial and Technical Overview*. IBM Redbooks, 2006.

[16] MEMSCAP. *TP 31XX customer protocol and commands*, 3 2004. Rev. 02.

[17] MEMSCAP. *TP3100 User Manual*, 3 2005. Rev. 03.

[18] Howard Shane. *Accessing the Video Device*. 2005. online, http://www.tldp.org/HOWTO/Webcam-HOWTO/dev-intro.html.

[19] Howard Shane. *Framegrabbing Applications*. 2005. online, http://www.tldp.org/HOWTO/Webcam-HOWTO/framegrabbers.html.

[20] Xsens. *AHRS Attitude Heading Reference System - Xsens 3D motion tracking*. 2017. online, https://www.xsens.com/tags/ahrs/.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Gabriel Michal**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Mobilní záznamová jednotka Mobile Acquisition Unit**

Pokyny pro vypracování:

1. Realizujte SW na vestavném počítači, který umožní synchronizovaný sběr dat z IMU (Fairchild) a několika videokamer (Microsoft).
2. SW vybavení realizujte na platformě Unix/Linux.
3. Data z IMU ukládejte do textového souboru spolu s informací o synchronizaci s videem.
4. Video zaznamenejte pomocí vhodného nástroje (např. mencoder).

Seznam odborné literatury:

[1] C.Hunt, Linux, 2002, SoftPress, ISBN: 80-86497-25-9
[2] The Predator Team, Predator Experiment, 2016

Vedoucí: doc. Ing. Pavel Pačes, Ph.D.

Platnost zadání: do konce letního semestru 2017/2018

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 21. 2. 2017