VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Řídící software motorizované panoramatické hlavy

Managing Software Motorized Panoramic Head

2014 Tomáš Obadal

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Obadal**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Řídící software motorizované panoramatické hlavy
Managing Software Motorized Panoramic Head

Zásady pro vypracování:

Cílem bakalářské práce je implementace řídícího programu k obsloužení motorizované panoramatické hlavy za účelem tvorby fotografických panoramat ve vysokém rozlišení. Program bude pracovat na řídící jednotce ARDUINO.
1. Proveďte rešerši řídící jednotky ARDUINO a možnosti vývoje softwaru pro tuto jednotku.
2. Navrhněte potřebné funkce pro chod panoramatické hlavy.
3. Implementujte řídící software panoramatické hlavy.
4. Otestujte výsledný software v různých podmínkách.

Seznam doporučené odborné literatury:

[1] Sedgewick R.: Algoritmy v C, části 1-4, SoftPress, Praha, 2003
[2] Brian W. Kernighan, Dennis M. Ritchie: The C Programming Language, Second Edition, Prentice-Hall, 1988
[3] Programming guide for RobotC, http://www.robotc.net/wiki/NXT

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Seidl, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2014 ..... Obadal T.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 ..... Obadal T.

Rád bych poděkoval vedoucímu bakalářské práce Ing. Davidu Seidlovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této práce. Dále také rodině za bezmeznou podporu, zvlášť svému otci za pomoc při stavbě zařízení.

Abstrakt

Práce se zabývá implementací řídicího systému motorizované panoramatické hlavy, kdy hlavní funkcí je tvorba panoramat ve velmi vysokém rozlišení. Takováto výsledná fotografie pak poskytuje obrovské množství informací, které jsou využitelné v různých oborech. Příkladem může být cestovní ruch, kde například provozovatel lyžařského areálu může prezentovat vybavení a služby, či město své dominanty. Pomocí panoramat ve vysokém rozlišení lze také detailně sledovat vývoj daného území, kdy lze s velkou přesností vysledovat rozdíly v čase. Panoramatická fotografie se nemusí omezovat jen na objekty naší planety, ale lze ji použít i pro focení objektů hlubokého vesmíru.

Klíčová slova: panoramatická fotografie, gigapixel, astrofotografie, Arduino

Abstract

The work deals with the implementation of the control system of motorized panoramic heads, the main function is to create panoramas in very high resolution. Such resulting photo then provides a huge amount of information that can be used in various fields. An example may be tourism, for example, where the ski area operator can present equipment and services, or your city dominants. Using high-resolution panoramas can also closely monitor the development of the territory, which can be traced with great precision differences in time. Panoramic photographs may not be limited only to the objects of our planet, but also can be used for photographing deep sky objects.

Keywords: panorama photo, gigapixel, astrophotography, Arduino

Seznam použitých zkratek a symbolů

PWM	–	Pulse-Width Modulation
SPI	–	Serial Peripheral Interface
TWI	–	Two Wire Interface
I^2C	–	Inter-Integrated Circuit
USB	–	Universal Serial Bus
LCD	–	Liquid-Crystal Display
EEPROM	–	Electrically Erasable Programmable Read-Only Memory
SRAM	–	Static Random-Access Memory
SD	–	Secure Digital
XML	–	Extensible Markup Language
GPS	–	Global Positioning System
PSB	–	PhotoShop Big
JPEG	–	Joint Photographic Experts Group
GPIX	–	GigaPixel
PTP	–	Picture Transfer Protocol

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

Seznam tabulek

1 Úvod

Práce se bude nejdříve zaobírat problematikou panoramatické fotografie a problematikou potřebného hardwaru a softwaru. Dalším tématem bude samotná implementace řídicího systému. Práce je psána a strukturována tak, aby srozumitelnou cestou vedla k pochopení dané problematiky. Při popisu implementace řídícího software motorizované panoramatické hlavy jsem postupoval tak, jak reálně docházelo k implementaci funkcionalit, dle mnou zvolených priorit. Tento postup vedl k úspěšné implementaci všech potřebných funkcí.

1.1 Panoramatická fotografie

Pojmem panoramatické fotografie rozumíme fotografii s velkým úhlem záběru. Velkým úhlem záběru je myšlen úhel stejný nebo větší, než má lidské oko. Jedná se typicky o úhly $160\,^\circ$ horizontálně a $75\,^\circ$ vertikálně, ale za panorama je možno považovat i fotografie s úhly o něco menšími. Historie se datuje do první poloviny 19. století, kdy se Josephu Puchbergervi z Rakouska podařilo vytvořit snímek s úhlem záběru $150\,^\circ$. Vznik panoramatu se dá rozdělit do dvou hlavních způsobů:

- **Specializovaným panoramatických fotoaparátem nebo objektivem** – jedná se o typ vzniku, kdy kompletní širokoúhlá fotografie je vytvořena pořízením jediného snímku. Používá se k tomu buď fotoaparát s primárním zaměřením na panoramatickou fotografii či fotoaparátem s možností výměny objektivů a použitím vhodného objektivu.

- **Skládáním snímku s běžným úhlem záběru** – v dnešní době jde o používanější metodu, kdy lze použít libovolný fotoaparát. Panorama pak vzniká skládáním jednotlivých fotografií do jediné výstupní. Tento způsob pak vyžaduje, aby se jednotlivé fotografie překrývaly a bylo tak možné je pomocí společných znaků složit do jediného panoramatu. Jsou ovšem kladeny velké nároky na správné nastavení fotoaparátu tak, aby po složení snímků do jediné fotky vznikl celistvý dojem. Bude-li v této práci dále hovořeno o tvorbě panoramatického snímku, bude myšlena tato metoda vzniku.

Při sledování okolí pozorovatele se svět může jevit jako vnitřek koule. Při prohlížení na monitoru počítače, či na tištěné fotografii, je ovšem prohlížení vždy ploché. Aby nedocházelo při zobrazování panoramat ke zkreslení, musí být zvolena vhodná projekce. Těch je mnoho typů. Zde se zaměřím na tři základní:

- **Cylindrické panorama** - vytváří při projekci dojem, že pozorovatel stojí uprostřed válce. Působí přirozeně, pokud je zorný úhel plných $360\,^\circ$.

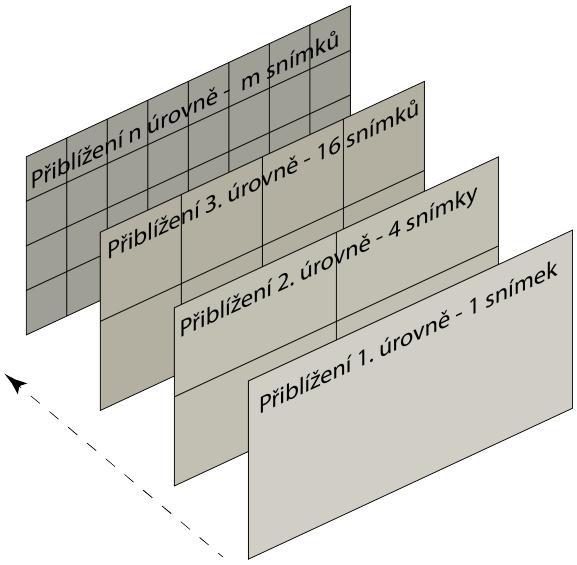
- **Krychlové panorama** - panorama je složeno z šesti stěn krychle. Používá se pro zobrazení při vertikálním úhlu větším, než $140\,^\circ$, kdy je tato projekce nejúspornější ve srovnání s ostatními.

- **Kulové panorama** - také nazývané sférické. Převládající způsob projekce panoramat. Při kompletním $360/180^\circ$ panoramatu je zastoupení dominantní, působí zde nejpřirozeněji. Divák se tak kromě klasického pohledu před sebe může rozhlížet i nahoru a pod sebe. Člověk se dívá na plášť vnitřku koule.

1.2 Prezentace on-line

Protože gigapixelové panorama mívá běžně velikost desítek gigabajtů, není možné jej přímo zobrazovat na internetu. Fotografie je jednoduše příliš velká, aby si ji uživatel celou stáhl do počítače a pak zobrazoval.

Byl proto vymyšlen způsob, jak i takhle veliké fotografie uživatelům přes internet zprostředkovat. Myšlenka je taková, že uživatel si stáhne jen tu konkrétní část obrazu, na kterou se zrovna dívá. Celé panorama se rozdělí do vrstev, kdy každá vrstva odpovídá jen určité míře přiblížení. Prohlížeč uživatele tak musí stáhnout jen několik snímků, které náleží jen konkrétní oblasti a konkrétnímu přiblížení na které je zrovna uživatel zaměřen. Každá taková vrstva je pak dále rozdělena do tisíců menších fotografií o průměrné velkosti 30 KB. Toto řešení rapidně zmenší objem dat, který si uživatel pro zobrazení musí stáhnout. K těmto malým fotografiím je navíc přiřazena pozice v panoramatu, kterou specializovaný prohlížeč dokáže přečíst a správně podle nich umístit do matice. Prohlížeče také musí implementovat funkce pro plynulé přecházení mezi vrstvami přiblížení, aby byl pohyb co nejplynulejší a uživatel tak nebyl obtěžován nepříjemným trháním. Dále lze také do panoramat přidávat interaktivní obsah, od popisu např. pamětihodností, vrcholů hor, až po přidávání videí přiřazených určitému místu, a jiné.



Obrázek 1: Vrstvení snímků podle míry přiblížení

1.3 Hardware panoramatické fotografie

K vytvoření základního panoramatického snímku je potřeba fotoaparát osazený objektivem. Pro snadnou manipulaci se snímky se používá fotoaparát digitální. Lze použít prakticky každý přístroj osazený fotoaparátem. Pro tvorbu gigapixelových panoramat se ale výhradně využívají digitální zrcadlovky. Velkou pomoc pro stabilizaci snímku je pak fotografický stativ. K základní tvorbě panoramata není více potřeba. Problém nastává, pokud je potřeba vytvořit panorama ve velmi vysokém rozlišení. Aby bylo takového rozlišení dosaženo, je potřeba vyfotit stovky, případně tisíce fotografií s objektivem s velkým ohniskem, typicky 200mm a více. Zorný úhel jednotlivé fotografie je pak tak malý, že je nemožné zvládnout ručně posouvat fotoaparátem tak, aby se jednotlivé snímky navzájem překrývaly jak je potřeba a byla jistota, že nebyla vynechána žádná část fotografované oblasti. Podstatnou komplikací je i to, že se musí ručně několik set krát, či několik set tisíckrát mačkat spoušť. Pro co nejlepší záběr je mnohdy potřebné fotografovat z těžko přístupných míst, kde by bylo extrémně obtížné fotit ručně.

Je proto nutné použít specializovaný hardware, který za nás tuto práci vykoná automaticky. S použitím tohoto zařízení pak zajistíme, že je fotografovaná oblast perfektně pokryta snímky, které se vzájemně překrývají a že počet snímků je jen minimální možný, pro pokrytí focené oblasti. To ušetří výpočetní čas při skládání panoramata ve specializovaném softwaru.

1.3.1 Vlastní hardware

Pro tyto potřeby jsem si vyvinul vlastní hardware. Než jsem ale zvolil cestu vlastního vývoje a výroby hardware, provedl jsem výzkum, zda-li se mi to vůbec vyplatí. Po důkladném průzkumu jsem zjistil následující:

- **neexistující programovatelné řešení** - nenalezl jsem žádný produkt, který by umožňoval snadný přístup ke zdrojovému kódu a umožnil tak vlastní úpravy.
- **cena** - po provedení propočtů jsem zjistil, že budu schopen hardware postavit za třetinovou cenu oproti komerčně nejprodávanějšímu produktu.
- **funkce** - funkcionalita bude větší nebo srovnatelná, než u většiny produktů mnohonásobně dražších.
- **vzdělávání a nasazení** - pro implementaci je potřeba využít nabyté znalosti a zkušenosti a naučit se mnohému novému. Také bylo motivací zapálení pro tento obor fotografie.

Při stavbě zařízení jsem využil běžně dostupných materiálů tak, aby v případě poškození mohlo dojít k bezproblémové opravě a zároveň cena zařízení zůstala na přijatelné úrovni.

Zařízení se skládá ze dvou hlavních částí:

- **Přístrojová skříň** - obsahuje řídící elektroniku spolu s akumulátorem. Na skříni je také držák pro uchycení na stativ, zapuštěn je displej a tlačítka.

- **Pohyblivé rameno** - tvoří jej dva spojené profily. Na vertikálním profilu je umístěn nastavitelný držák fotoaparátu. Pomocí dvou krokových motorů je poté zajištěn pohyb v osách x a y, který umožní vyfocení scény. Kabeláž vedoucí do přístrojové skříně nutná pro správný chod robota je poté vedena takovým způsobem, aby mohlo docházet k přetočení o více, než $360\,^{\circ}$ bez kroucení kabelů.

Od samého začátku bylo zařízení koncipováno tak, aby jeho stavba byla robustní, zvládla obsluhovat i těžší fotografické vybavení. Tomu odpovídá uchycení hřídelí do ložisek, použití přesných krokových motorů a jejich zpřevodování pro bezproblémový chod. Snímek robota je možné vidět v obrazové příloze na obrázku 10.

1.3.2 Elektronický hardware

V této části se budu věnovat hardware, který je potřeba k implementaci řídícího softwaru.

Při rozhodování, jaký hardware použít, bylo hlavním kritériem možnost vývoje v softwaru Microsoft Visual Studio, na kterém v několika předmětech probíhala má dosavadní výuka a byl jsem s ním podrobně obeznámen. Dalším kritériem byla snadná dostupnost a uživatelská základna z důvodu případných řešení problémů, které lze v bohaté komunitě snadno řešit.

Po zvážení těchto kritérii jsem se rozhodl pro využití open-source platformy založené na mikrokontrolerech ATmega od společnosti Atmel. Tato platforma nabízí možnost programování v jazyce C/C++, nabízí doplněk pro práci v Microsoft Visual Studiu, pohodlné nahrávání programů přes USB rozhraní a příznivou pořizovací cenu. Jedná se o rodinu produktů italské společnosti Arduino. Jako konkrétní model bylo zvoleno Arduino MEGA 2560, které nabízí velký počet vstupně/výstupních pinů pro připojení periférií, potřebných pro správné fungování panoramatického robota, dostatečný výkon, podporu technologií pro komunikaci a uspokojivé množství paměti. Zde je jeho základní charakteristika:

- **Operační napětí: 5V** - s tímto napětím pracuje většina periférii, není proto nutné využívat externích napěťových zdrojů. Mimo 5V nabízí i 3.3V pro možnost připojení další periferie.
- **Počet digitálních pinů: 54** - umožňuje připojení velkého množství čidel, senzorů a periférií. Díky dostatku pinů je možné přímo vyhradit samostatný pin pro každé tlačítko s možností hardwarového přerušení.
- **Počet PWM výstupních pinů: 15** - připojení LCD displeje, piezo snímače a reproduktoru, regulace rychlosti otáčení motorů.
- **Další I/O technologie:** Sériové porty, SPI komunikace, TWI komunikace.
- **Analogových pinů: 16** - možnost připojit analogové senzory, měřit napětí baterie, apod.
- **Flash paměť: 256 KB** - více než dostatečný prostor pro uložení i velmi velkých programů.

- **SRAM: 8 KB** - paměť, která postačuje pro uložení a manipulaci s proměnnýma za běhu programu.
- **EEPROM: 4 KB** - paměť na uložení dat, jejíchž informace je zachována i při odpojení zdroje napájení.
- **Taktovací frekvence: 16 MHz** - dostačující pro plynulý běh programu. V případě nutnosti je možné vyměnit 16 MHz oscilátor za rychlejší.

1.4 Software panoramatické fotografie

Pokud jsou snímky vyfoceny, při panoramatu ve vysokém rozlišení běžně stovky, či tisíce, je potřeba z nich vytvořit jedinou celistvou panoramatickou fotografii. K tomuto účelu se používá přímo specializovaný software. Existuje mnoho komerčních i nekomerčních řešení, které nabízejí odlišnou funkcionalitu a kvalitu výstupu.

Jak již bylo řečeno dříve, aby se snímky mohly spojit, je potřeba, aby se vzájemně překrývaly. Typicky se používá překrytí od 25% do 30%. Software pak v těchto překryvných oblastech nalezne společné body a fotky podle nich spojí. Ve většině případů se body nalézají na kontrastních přechodech, kde je pro software nejjednodušší nalezení homogenních bodů.

Pokročilejší nástroje nabízí i funkce pro barevnou korekci, neboť typickým problémem velmi velkých panoramat je, že se během focení změní množství světla, Slunce zajde za mrak, postupně se pohybuje po obloze, apod. Program pak dokáže tyto barevné změny mezi jednotlivými snímky detekovat a dokáže barevnost snímku korigovat tak, aby složený snímek vypadal celistvě a bez rušivých barevných změn. Častým problémem bývá špatně ustavený horizont, je proto nutné, aby software umožňoval jeho narovnání. Pokročilejší nástroje umožní korigovat i zkreslení a obvyklé výrobní vady objektivů.

2 Řídící software panoramatické hlavy

Před samotnou implementací musel být proveden výzkum, jaké funkce budou pro pořizování panoramatických snímků ve vysokém rozlišení potřeba. Brána v potaz musela být i možnosti využití robota k jiným účelům, a proto bylo dáno za úkol provést výzkum i v této oblasti. Po vyhodnocení výsledků byl zvolen následující postup implementace:

- **Uživatelské rozhraní** - snadná obsluha zařízení pomocí tlačítek s přehledným výstupem na displej.
- **Funkce panoramatu** - vložení nejdůležitějších funkcí. Po této části vývoje musí být motorizovaná hlava plně vybavena pro potřeby tvorby panoramatu.
- **Další funkce** - funkce, které mohou napomáhat tvorbě panoramatických snímků, či mají informativní charakter.
- **Astro-fotografie** - implementace funkcí pro tvorbu snímků hlubokého vesmíru. Provést napojení na již dostupné funkce.

Jako priorita byla také určena spolehlivost řídícího programu s minimálním výskytem chyb při běhu, které by mohly mít za následek znehodnocená, či zcela nepoužitelná data pro výrobu panoramatu. Chyby v běhu programu by mohly vést k následujícím problémům:

- **Chybějící snímek** - velice závažný problém, je možné jej napravit jen opětovným návratem na místo, kde bylo foceno a chybějící snímek dofotit.
- **Nepřesné vykonání procesu** - dojde-li v důsledku chyby k vykonání focení panoramatu jinak, než je očekáváno, vede obvykle k naprostému znehodnocení dat a jejich nepoužitelnosti pro další zpracování.

2.1 Menu

Aby bylo možné s motorizovanou panoramatickou hlavou pohodlně pracovat, bylo nutné implementovat uživatelské rozhraní, které by reagovalo na vstup z hardwarových tlačítek a přehledně zobrazovalo potřebné informace a nastavení na displej. Zvolil jsem proto řešení pohyblivého menu, kdy je možné pomocí tlačítek přecházet mezi jednotlivými položkami, vstupovat do podnabídek a popřípadě měnit hodnoty přiřazené dané položce. To vše muselo být podřízeno zvolenému typu displeje, který umí zobrazovat čtyři řádky po dvaceti znacích. Pro samotné zobrazení informací na displeji jsem využil veřejně dostupné knihovny *[Liquid Crystal]*[3] autora Francisco Malpartida. Při implementaci jsem si nejdříve definoval strukturu, která obsahuje tyto základní informace:

- *text* – jedná se o text, který bude zobrazen na displeji jako název položky
- *numberOfItems* – celočíselná hodnota informují o počtu položek v daném menu

- *up* – celočíselná hodnota informující o položce, na kterou se kurzor přesune po stisknutí tlačítka nahoru
- *down* – celočíselná hodnota informující o položce, na kterou se kurzor přesune po stisknutí tlačítka dolů
- *enter* – celočíselná hodnota informující o položce, na kterou se kurzor přesune po stisknutí potvrzovacího tlačítka
- *exit* – celočíselná hodnota informující o položce, na kterou se kurzor přesune po stisknutí tlačítka exit
- **fp* – ukazatel na funkci, který je-li nenulový, spustí po stlačení potvrzovacího tlačítka zvolenou funkci

Ukázka implementace struktury je pak následující:

```
{menu_000, p_pol0, 0,   0,   0,   0,   0},              // Prvni pevna polozka

{menu_001, p_pol0, 8,  2,   1,   1,    Accelerometer}, // Jednotlive polozky menu
{menu_002, p_pol0, 1,   3,   10,  2,   0},
{menu_003, p_pol0, 2,   4,   23,  3,   0},
{menu_004, p_pol0, 3,   5,   4,   4,   Move},
{menu_005, p_pol0, 4,   6,   5,   5,   DateTime},
{menu_006, p_pol0, 5,   7,   6,   6,   Thermometer},
{menu_007, p_pol0, 6,   8,   34,  7,   0},
{menu_008, p_pol0, 7,   1,   8,   8,   Battery},
```

Výpis 1: Struktura menu

Díky takto vytvořené struktuře jsem mohl definovat všechny potřebné položky menu. Při zobrazování menu vždy platí, že první řádek je pevný, ukazuje název menu, či podmenu v kterém se momentálně nacházím, další tři jsou proměnné, ukazují obsah v závislosti na stisku hardwarových tlačítek.

Dále je definována proměnná *cursor*, která určuje položku menu, s kterou je možná interakce, např. změnit přiřazenou hodnotu, či potvrzovacím tlačítek spustit funkci. Při vykreslování menu je nejdříve nutné zjistit začátek a konec současného menu, a to tak, že pomocí smyčky *while* procházíme položky, dokud je proměnná *end* menší nebo rovna pozici kurzoru. V každé iteraci pak přičteme k proměnné *end* počet položek daného menu, definovaného v struktuře menu proměnnou *numberOfItems*. Je-li znám konec menu, tak začátek je vypočten jednoduchým odečtem počtu položek daného menu od konečné hodnoty definovanou proměnnou *end*.

Je-li znám začátek a konec, tak je možno vykreslit první pevnou položku menu, např. "Hlavní MENU", která je v rámci daného menu neměnná a nepohybuje se. Pojmenovává část uživatelského prostředí, v kterém se zrovna nacházíme. Poté je potřeba vykreslit položky v závislosti na poloze kurzoru. K tomu slouží tři podmínky, které se provedou v závislosti na tom, je-li kurzor na první položce, uprostřed menu nebo na konci. Ukázka kódu prvního případu je vidět na následujícím výpisu:

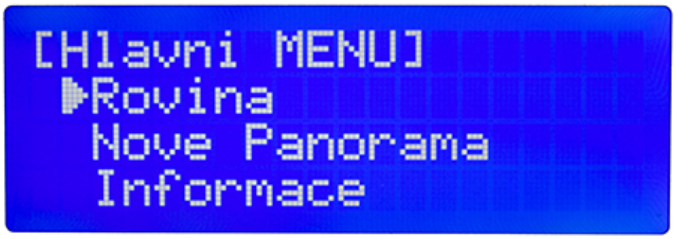
```
if (cursor < (begin + 2)) // pokud ma cursor pod sebou dve pozice
{
    begin = cursor;   // cursor se rovna prvni pozici v menu
    end = begin + 2; //  viditelny konec je o dve pozice nize (nulty radek je pevny)

    for (begin; begin<=end; begin++) // vykreslime polozky
    {
        lcd.setCursor(2, rowCount); //odsadime polozku v menu
        lcd.write(menu[begin].text); //zapiseme na displej polozku
        DeleteAfterText(begin,rowCount); // smazeme zbytek za textem
        rowCount++; // urcuje radek, kde dojde k odsazeni
    }
   //Formatovani vypisu na displej
    lcd.setCursor(1,2);
    lcd.write("␣");
    lcd.setCursor(1,3);
    lcd.write("␣");
    lcd.setCursor(1, cursor-temp);
    lcd.write(byte(6));
}
```

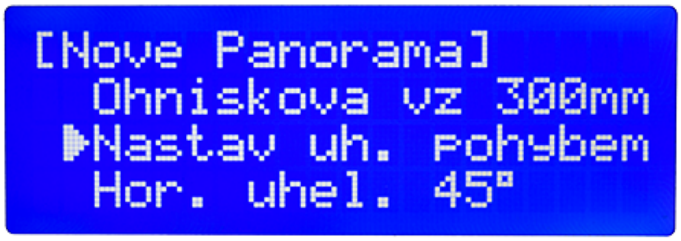
Výpis 2: Vykonání podmínky s kurzorem na začátku menu

Program nejdříve zjistí, má-li pod sebou dvě pozice, pokud ano, tak ví, že je na začátku menu a že konec menu, který se má vykreslit, je o dvě pozice níže. Poté se projdou všechny položky od zjištěného začátku a konce a vykreslí se na displej. Pokud při předchozím vykreslení položek byl text některé z položek delší, než je text položky současné, je třeba provést funkci *DeleteAfterText(begin,rowCount)*, která zjistí délku současného textu a smaže vše, co je za ním. Implementace funkcí je provedena tak, aby bylo při procházení menu poznat, kde se nacházíme. Na začátku může kurzor přecházet z první na druhou položku. Při procházení uprostřed menu se pak kurzor drží na druhé pohyblivé položce. Na konci pak může přecházet mezi druhou a třetí položkou, čímž dá najevo, že bylo dosaženo konce menu. Z důvodu praktičnosti je možné se přesouvat mezi poslední a první položkou menu a obráceně.

Ukázku, jak vypadá menu a zobrazení informací při focení, můžete vidět na následujících příkladech:



Obrázek 2: Hlavní menu



Obrázek 3: Podmenu

Obrázek 4: Displej při focení

Ve funkci *GoMenu(void)* se provádějí dvě hlavní operace. První je nastavování hodnot přiřazených některým položkám menu. Jsou to hodnoty, které přímo ovlivňují chování panoramatického robota a mění jeho funkčnost. Nastavení probíhá pomocí příkazů *switch, case*, kdy rozhodujícím argumentem je celočíselná hodnota proměnné *cursor*, tedy pozice, na které se momentálně nacházím v menu. V návěstích se pak definuje název proměnné, a o kolik se hodnota změní po stisku tlačítka. Dané jsou i krajní hodnoty, při jejíž dosažení již nebude hodnota při stisku tlačítka měněna.

Druhou hlavní operací je tzv. debouncing, nebo-li potlačení falešných pulzů. Pokud je zmáčknuto tlačítko, nepřeskočí hodnota ihned z hodnoty 0 na 1, kde se ustálí, ale chvíli mezi hodnotami přeskakuje. To má za následek zachycení vícero stisků tlačítka, což může být nežádoucí. Řešit tento problém je možné dvěma způsoby:

- **hardwarově** - za použití kondenzátoru se vyhladí průběh signálu a získá tak čistší výstup bez přeskakování hodnot.

- **softwarově** - řešení, kdy je po prvním zjištění signálu čtení na určitou dobu přerušeno, čímž se nechtěné signály ignorují.

Jako řešení jsem zvolil druhou možnost, tedy softwarové řešení problému, a to z důvodu snadné implementace a spolehlivosti řešení. Díky tomuto způsobu si také mohu regulovat rychlost pohybu kurzoru v menu a rychlost změny nastavovaných hodnot.

2.1.1 Přerušení

Ať je vykonávána jakákoli funkce programu, je nutné zajistit, aby ji bylo možno stisknutím tlačítka přerušit a dostat se zpátky do menu uživatelského prostředí. Existuje několik způsobů, jak se s tímto problémem vypořádat.

- **Vlákna** - v samostatném vlákně se bude kontrolovat, zda-li nebylo zmáčknuto tlačítko. Tato možnost nemohla být realizována, protože Arduino vlákna nepodporuje.

- **Kontrola v každé funkci** - implementovat kontrolu stisknutého tlačítka do všech funkcí. Tato možnost by byla funkční, ale ne příliš elegantní. Kontrola v každé funkci by nezanedbatelně zvýšila velikost programu. U rozměrnějších funkcí by pak bylo nutné vkládat kontrolu vícekrát, aby byla reakce na stisknuté tlačítko okamžitá. To by ovšem mohlo mít za následek zpožděné vykonávání samotného účelu zařízení, kdy by kontrola zbrzďovala funkci.

- **Hardwarové přerušení** - asynchronní metoda, kdy se při stisku tlačítka přeruší právě vykonávané instrukce a vykoná se funkce přerušení. Protože Arduino umožňuje na konkrétních pinech hardwarové přerušení provozovat, zvolil jsem tuto metodu jako nejlepší pro řešení mého problému.

K správné implementaci mi postačily dvě funkce. První je *interrupts()*, která povolí přerušení používat. Druhou pak *attachInterrupt()* s příkladem implementace *attachInterrupt(5,SetMenu,RISING)*, kde první parametr funkce udává číslo přerušení, které odkazuje na konkrétní pin na Arduinu, kde je tlačítko připojeno, *SetMenu* je pak odkaz na funkci, která se při přerušení vykoná. V tomto případě se vrátí do menu uživatelského rozhraní. Poslední parametr dovoluje nastavit, za jakých okolností se hardwarové přerušení vykoná. Nabízí se tyto možností:

- **LOW** - k spuštění dojde, když je pin na LOW.

- **CHANGE** - k spuštění dojde kdykoli se hodnota na pinu změní.

- **RISING** - ke spuštění dojde, pokud se změní hodnota na pinu z LOW na HIGH.

- **FALLING** - ke spuštění dojde, pokud se změní hodnota na pinu z HIGH na LOW.

Vzhledem ke skutečnosti, že stisknutí tlačítka je implementováno jako změna pinu z LOW na HIGH, tak bylo logické stejným způsobem detekovat i hardwarové přerušení.

2.2 Panoramatické funkce

Nejdůležitější částí software jsou panoramatické funkce. Ty se starají o veškerou funkcionalitu potřebnou k zajištění operací nutných k pořízení panoramatu.

2.2.1 Časovače

Na začátku si definuji funkce časovačů, neboť při funkci robota je zobrazeno, kolik času uběhlo a kolik času zbývá do dokončení zadaného úkolu. Použity jsou také pro softwarový debouncing popsaný výše. Tyto funkce jsou řešeny tak, aby neblokovaly další operace. Použita je funkce *millis()*, která počítá množství uběhlých milisekund od spuštění. Příklad použití je možné vidět na následujícím kódu.

```
int interval = 3000; // interval po jehoz uplynuti se vykona telo podminky
unsigned long currentMili = millis(); // ulozeni ubehlych milisekund do promenne

if(currentMili - previousMili > interval) // prekroci-li pocitani zadany interval
{
  previousMili = currentMili; // predchozi ubehly cas se stava soucasnym
  //Funkce//
}
```

Výpis 3: Ukázka neblokujícího zpoždění

Z kódu vyplývá, že podmínka se vykoná jen v okamžiku, je-li počet uběhlých milisekund větší, než interval. Uběhlý čas je poté přičten k současným milisekundám a znovu se čeká zadaný interval.

Místo této funkce se běžně používá příkaz *delay(milliseconds)*, ten ovšem na zadanou dobu zcela zablokuje vykonávání kódu, což by v mém případě způsobilo zastavení funkcí časovače a dalších funkcí nutných pro chod robota.

2.2.2 Pohyb motorů

Jako další jsem implementoval funkce pro pohyb krokových motorů, ty jsou dva, pro horizontální a vertikální pohyb. Ke komunikaci s řadiči pro pohyb motorů využívám veřejně dostupnou knihovnu autora Mike McCauleyho *[AccelStepper]*[4]. V těchto funkcích bylo potřeba myslet na několik věcí.

Jako první se přepočítávají stupně na kroky. Při nastavování parametrů panoramata, které chci vyfotit, je potřeba zadávat úhel záběru v horizontálním a vertikálním směru. Tím si definuji oblast, která bude ve výsledném panoramatu zachycena. Tento úhel se zadává ve stupních. Aby se motor otočil o zvolený úhel, je třeba stupně převést na kroky. To se provede přes následující vztah:

$$kroky = \frac{stupne}{minimalni\ krok};\ kde\ minimalni\ krok = 0.01125^{\circ} \tag{1}$$

Minimální krok byl spočítán ze známých hodnot základního kroku motoru (1.8°), mikrokroků (32 na jeden základní krok), které umožňuje řídící obvod motoru a velikosti zpřevodování, které je pětinásobné.

$$min\_vel\_kroku = (1.8/32)/5 \tag{2}$$

Poté bylo třeba nastavit maximální rychlost otáčení a rychlost zrychlení. Tyto dva parametry jsou velice důležité pro plynulý chod robota, a to tak, aby nedocházelo k

nechtěným vibracím, které by mohly mít za následek rozmazání focených snímků. Na toto nastavení využívám funkce knihovny *[AccelStepper]*[4], konkrétně *setMaxSpeed()* a *setAcceleration()*. Hodnoty těchto funkcí jsou relativní a liší se v závislosti na použitých motorech, řadičích motorů, napětí.

Při každém zavolání funkce k pohybu motoru se musí zavolat *setCurrentPosition(0)* a vynulovat tak uloženou pozici hřídele motoru. Pak se podle potřeby změní směr, kterým se má hřídel otáčet záměnou znaménka před proměnou *steps* ve funkci *moveTo(steps)*. Tyto operace je možné vidět na následujícím kusu kódu:

```
motorH.setMaxSpeed(10000); // nastaveni maximalni rychlosti
motorH.setAcceleration(4000); // nastaveni miry zrychleni a zpomaleni
motorH.setCurrentPosition(0); // vynulovani relativni pozice hridele

switch (direction)
{
   case 0: motorH.moveTo(-steps); // pohyb ve smeru hodinovych rucicek
   break;
   case 1: motorH.moveTo(steps); // pohyb v proti smeru hodinovych rucicek
   break;
   default: break;
}

while(motorH.distanceToGo() != 0) // bude se vykonavat, dokud nedorazime na misto
{
   Timers(); // vykonani funkce Casovace
   motorH.run(); // pohyb motoru o jeden krok
}
```

Výpis 4: Vykonání pohybu motoru

Tento způsob vykonání pohybu je zvolen s ohledem na to, aby neblokoval funkce časovače. Smyčka *while* v každé iteraci pohne motorem jen o jeden krok, je tak možné během pohybu volat funkci *Timers()*, která na displeji vykresluje uběhlý a zbývající čas.

2.2.3 Panoramatické výpočty

K správnému fungování motorizované hlavy při focení panoramata je potřeba vykonat několik výpočtů. Ty zajistí správné vykonání pohybu tak, aby došlo k vyfocení té částí, která je nastavena a zajistí, aby snímky byly překrývány o danou hodnotu vyjádřenou v procentech.

Nejdříve je nutné provést výpočet horizontálního a vertikálního úhlu. Úhlu, o který se robot posune před vyfocením každého snímku. Na vstupu je potřeba znát dvě proměnné, ohniskovou vzdálenost přepočtenou na velikost kinofilmového políčka (standardní číslo udávané výrobcem objektivů v milimetrech) a míru překryvu mezi jednotlivými snímky v procentech.

$$uhel_h(rad) = 2 \cdot arctg\left(\frac{12}{ohn.\,vzdalenost}\right) - \left(\frac{prehryv_h}{100} \cdot 2 \cdot arctg\left(\frac{12}{ohn.\,vzdalenost}\right)\right) \tag{3}$$

Výpočet se provádí vůči polovině velikosti standardního kinofilmového políčka, které má rozměry 36x24 mm, proto konstanta 12. Protože je fotoaparát připevněn na robotovi v portrétním módu, tedy na výšku, tak se pro horizontální úhel počítá s polovinou výšky kinofilmového políčka. Výpočet vertikálního úhlu je totožný jen s tím rozdílem, že místo poloviny výšky kinofilmového políčka se použije polovina šířky, tedy 18.

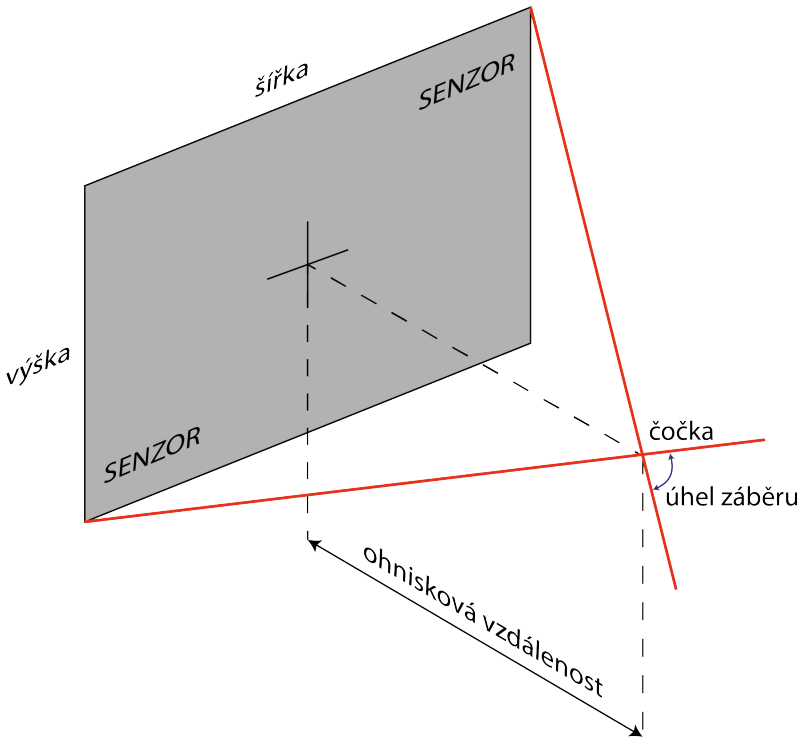
Protože výsledek vyšel v radiánech, tak jej převedeme na stupně.

$$uhel_h\_kinofilm(stupne) = \left(\frac{uhel_h(rad) \cdot 4068}{71}\right) \tag{4}$$

$$uhel_h(stupne) = \left(\frac{uhel_h\_kinofilm(stupne)}{1.6}\right) \tag{5}$$

V prvním vztahu odpovídá podíl $\frac{4068}{71}$ jednomu radiánu a lze jej také vyjádřit jako $\frac{180}{\pi}$. Přesnost vyjádřená celočíselným podílem je zcela dostatečná pro účely převodu. Konstanta 1.6 je pak tzv. crop faktor snímače použitého fotoaparátu. Je to číslo, které vyjadřuje kolikrát je snímač menší, oproti kinofilmovému políčku.

Následující obrázek demonstruje, který úhel je počítán.



Obrázek 5: Úhel zabírající objektiv

Po těchto výpočtech je možné zjistit, kolik bude potřeba vyfotit snímků k vytvoření panoramatu, dle zvoleného celkového úhlu záběru. Zvlášť se spočítá počet snímků pro horizontální pohyb a zvlášť pro vertikální. Tak je zjištěn počet potřebných řad a sloupců. K zjištění celkového počtu se pak oba dílčí výpočty vynásobí.

$$snimku_h = \frac{celkovyUhel_h}{uhel_h} \tag{6}$$

$$snimku_v = \frac{celkovyUhel_v}{uhel_v} \tag{7}$$

$$celkoveSnimku = snimku_h \cdot snimku_v \tag{8}$$

Protože výsledek nemusí být celé číslo, oba dílčí výpočty se proto zaokrouhlují vždy nahoru. V panoramatické fotografii platí pravidlo, vyfotit raději více snímků a ořezávat, než aby nějaký snímek chyběl.

Jako poslední kalkulace se provede zjištění času, který je potřeba k vyfotografování všech snímků.

$$cas(s) = (casPredSnimkem(s) + casPoSnimku(s)) \cdot pocetSnimku \tag{9}$$

casPredSnimkem a *casPoSnimku* jsou proměnné, které se nastavují v uživatelském rozhraní před samotným spuštěním operace focení. Říkají, jak dlouho se bude po vykonání pohybu motorů čekat, než bude snímek pořízen a kolik času se bude čekat po pořízení snímku před dalším pohybem motorů.

2.2.4 Spoušť

Bez implementace funkce *Spoust* by nevznikl jediný snímek. Panoramatický robot je připojen k fotoaparátu na dálkovou spoušť, která funguje na jednoduchém principu spínače, je-li obvod uzavřen, pořídí se snímek. Pro ochranu fotoaparátu je spínač řešen přes optoizolátor, čímž se minimalizuje možnost průniku nežádoucího napětí do obvodu dálkové spouště fotoaparátu. Aby se snímek pořídil, musí být obvod uzavřen nejméně na 300 ms, proto musely být použity neblokující zpoždění, které neovlivní funkce časovačů. Byl použit totožný princip implementace, jako ve Výpisu 3. Samotné sepnutí spouště je pak realizováno standardní funkci knihovny *[Arduino]*.

```
digitalWrite (shutterPin, HIGH); // nastavi vystup pinu na HIGH = log(1)

digitalWrite (shutterPin, LOW); // nastavi vystup na LOW = log(0)
```

Výpis 5: Sepnutí spouště

2.2.5 Nastaveni zorného úhlu pohybem

Před samotným započetím procesu focení je potřeba nastavit úhel záběru, který bude panoramatický snímek pokrývat. To je možné udělat dvěma způsoby, buď se přímo nastaví ve stupních horizontální a vertikální úhel, a nebo se použije funkce nastavení zorného úhlu pohybem. To se provede tak, že s již připevněným fotoaparátem na motorizované hlavě se pomocí tlačítek pohybuje panoramatickou hlavou do krajních částí chtěného zorného úhlu. Robot nejprve požádá, aby se pozice přesunula do pravého horního rohu, poté levého horního a nakonec do levého spodního, tím je možno odečíst potřebný horizontální a vertikální úhel a přesně si vymezit oblast, která bude zachycena na snímku.

Princip fungování je následující. V prvním potvrzeném bodě, tedy pravém horním rohu, se nastaví horizontální a vertikální úhel na nulu. Protože je známa velikost jednoho kroku motoru, tak je možno spočítat, kolik takových kroků bylo uděláno při cestě do levého horního rohu, tím je získán horizontální uhel ve stupních. Stejný princip se využije při pohybu do levého dolního rohu.

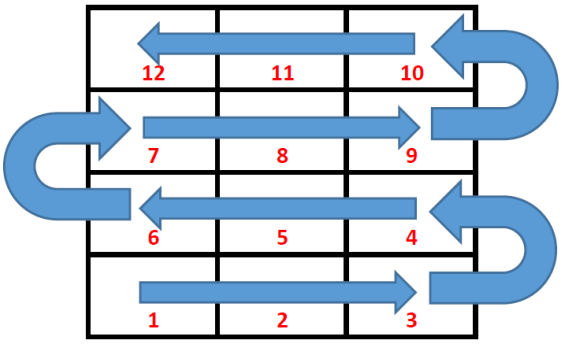
2.2.6 Fotografování

Tato funkce zajistí samotné vyfotografování panoramatu. Využívá ke svému fungování výše popsané funkce.

Po definování potřebných proměnných, se provede funkce panoramatických výpočtů, která dodá potřebné informace o počtu potřebných snímků a úhlu, o který se před každým snímkem motor posune.

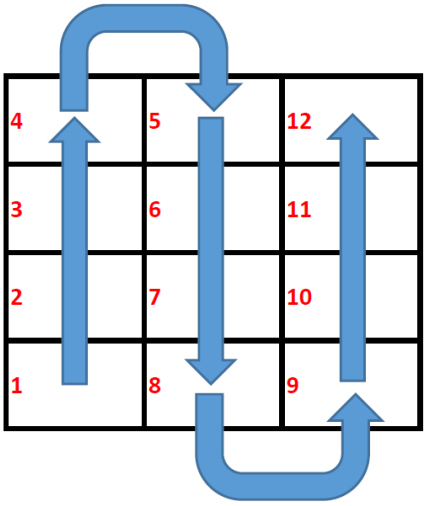
Při implementaci muselo být myšleno na to, že způsobů, jak vyfotit matici snímků, je několik a každý se hodí na jinou fotografovanou scénu. Proto jsou do programu zaneseny dva základní způsoby, které mohou být v uživatelském rozhraní nastaveny:

- **focení po řadách** - pro menší panoramata s délkou focení pod 15 minut. Při focení po řadách na sebe budou stíny, které focené objekty vrhají, přirozeně navazovat.



Obrázek 6: Focení po řadách

- **focení po sloupcích** - hodí se při použití objektivů s velkým ohniskem a dlouhou dobou focení. Při použití tohoto způsobu nejsou tolik patrné změny ve světelných podmínkách napříč panoramatem a pohybu Slunce po obloze.



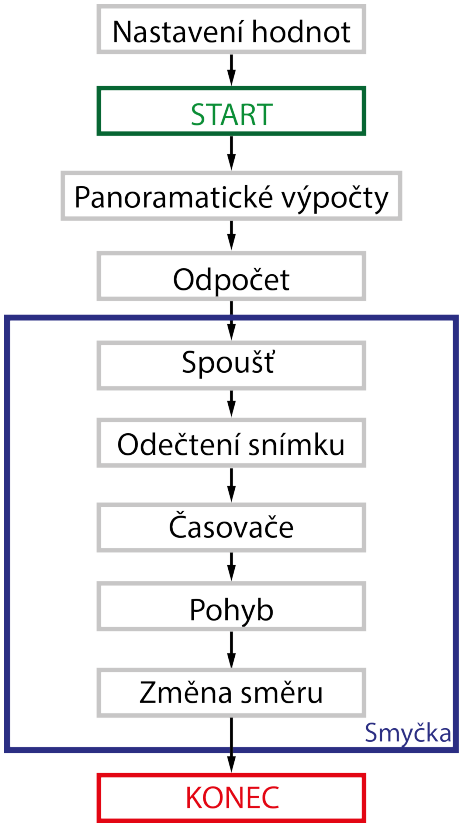
Obrázek 7: Focení po sloupcích

Realizace v programu probíhala pomocí *switch a case*, kde v návěstích bylo definováno, jakým způsobem bude probíhat pohyb.

Před započetím focení je možné nastavit tzv. zpoždění. Než začne samotný proces, odpočítá se zvolený počet sekund. To se hodí například pro ustálení vibrací při manipulaci se zařízením. Pro vykreslování číslic při odpočtu využívám veřejně dostupnou knihovnu *[Phi-super-font]*[5] autora Dr. Johna Liu, která je mnou upravena pro možnost využití sběrnice I^2C pro komunikaci s displejem.

Protože fotografování probíhá střídavě, po vyfocení první řady/sloupce se změní směr otáčení a fotí se v opačném směru, je třeba tuto operaci změny otáčení motoru provést. Zvolil jsem příkazy *switch a case*, kdy rozhodujícím atributem je to, zda-li se právě fotí sudá, či lichá řada/sloupec. V návěstích se pak jen změní směr otáčení motoru.

Po provedení pohybu se vyčká nastavitelné množství času, poté se vyfotí snímek a opět čeká nastavitelné množství času. Tento proces se opakuje do té doby, než jsou vyfoceny všechny snímky. Focení panoramata je poté dokončeno. Během focení jsou také na displej vypsány informativní položky o době uběhlé od začátku focení a zbývající čas. Toto obstarává dříve popsaná funkce časovače. Dále počet vyfocených a zbývajících snímků. Ty se zajišťují inkrementací, respektive dekrementací daných proměnných při každém vyfocení snímku. Na obrázku 8 je ukázka operací potřebných k úspěšnému vyfocení panoramatu.



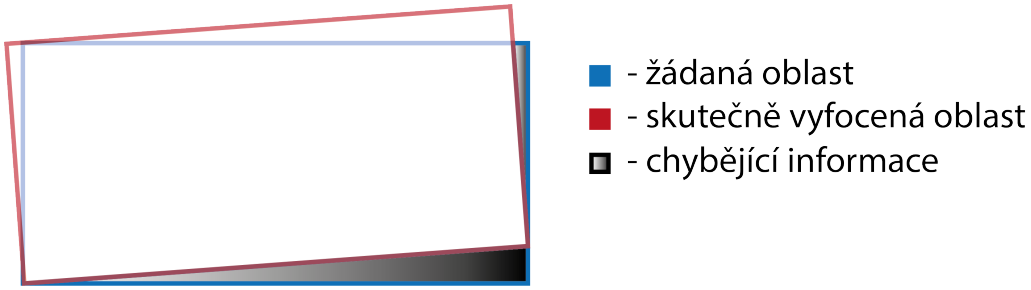
Obrázek 8: Posloupnost volaných operací

2.3 Pomocné a další funkce

Zde se budu zabývat funkcemi, které buď přímo souvisí s hlavním úkolem motorizované panoramatické hlavy, nebo slouží jako doplňková, informativní věc.

2.3.1 Akcelerometr

Je-li foceno panorama, je velice důležité, aby snímky byly foceny vodorovně s horizontem. Budovy či stromy by neměly tzv. padat. Pokud by snímky nebyly vyfoceny v souladu s tímto pravidlem, tak při spojování snímků v počítači do jediného panoramatu by muselo dojít k narovnání, k čemuž by byla využita softwarová interpolace, která by nezanedbatelně snížila obrazovou kvalitu, především ostrost. Dalším problémem, ke kterému by mohlo dojít, je chybějící informace oproti původně vybrané oblasti chtěné na výsledném snímku, jak je vidět na obrázku 9.



Obrázek 9: Chybějící informace při špatném ustavení zařízení

.

Pro správné ustavení zařízení do roviny jsem proto implementoval funkci, která práci usnadní. Ke svému chodu využívá akcelerometr ADXL345, který dokáže detekovat kromě gravitačního zrychlení i úroveň náklonu. Pro komunikaci s čipem přes I^2C sběrnici využívám knihovnu dodávanou se softwarem Arduino s názvem *[Wire]* a pro využívání funkcí veřejně dostupnou knihovnu *[ADXL345]*[6] společnosti *Love Electronics*.

Po spuštění funkce je navázáno spojení s akcelerometrem a ve smyčce jsou čteny surová data tak, jak je čip posílá. Tyto data jsou ovlivněna šumem, který zapříčiní náhlé kolísání hodnot. Proto, aby data byla prakticky využitelná, je nutné následky působení šumu co nejvíce eliminovat. Jako první řešení jsem použil průměrování příchozích hodnot, což sice šum po dostatku průměrovaných hodnot dobře eliminovalo, ale zobrazování hodnot trpělo neúnosným zpožděním oproti skutečné poloze senzoru. Bylo proto nutné přijít s jiným řešením. Po provedení výzkumu jsem nakonec zvolil tento vzorec:

$$y_t(stupne) = \alpha \cdot x_t - (1 - \alpha) \cdot y_{t-1} \tag{10}$$

Kde y_t je filtrovaný signál, y_{t-1} předchozí filtrovaný signál, x_t jsou surová data ze senzoru a α proměnná, která určuje míru vyhlazení signálu. Čtení a filtrování signálu je potřeba provádět pro všechny tři osy, tedy x y a z.

Nyní je již možné data převést na srozumitelnou informaci v podobě míry náklonu v osách x a y vůči směru působení zemské gravitace. Pro osu x je potřeba následující převodní vztah:

$$x(stupne) = arctan\left(\frac{-G_x}{G_z}\right) \tag{11}$$

Kde $-G_x$ je filtrovaná informace pro osu x a G_z filtrovaná informace pro osu z. Pro y pak platí:

$$y(stupne) = arctan\left(\frac{G_y}{\sqrt{G_x^2 + G_z^2}}\right) \tag{12}$$

Protože senzor je jen výjimečně již od výroby zkalibrován tak, aby měřil přesně, je zapotřebí provést dodatečnou korekci. To jsem provedl pomocí vodní váhy, kdy bylo potřeba v obou osách odečíst 2°. Poté je již možné informace zobrazit na displeji, kdy je

žádoucí, aby hodnoty v obou osách byly 0. Pro implementaci výše popsané funkcionality jsem využil zdroje [1].

2.3.2 Paměťová karta, XML soubory

Protože skládání stovek, či tisíců fotografií do jediného panoramatu může být pro počítač velice náročný a zdlouhavý úkol, je rozumné hledat řešení, jak tuto náročnost snížit. Když do programu na tvorbu panoramat nahrajeme snímky, tak ten pomocí algoritmů hledá společné znaky mezi všemi snímky, tedy i těmi, které spolu nemají nic společného. To značně zvyšuje dobu potřebnou pro správné složení. Je ovšem možné programu říct, kde má každý snímek v matici fotografií pozici a tím zabránit slepému hledání společných znaků napříč všemi snímky.

Při každém zmáčknutí spouště se uloží relativní x a y souřadnice pozic motorů, vůči startovnímu bodu, kterému je přiřazena pozice 0,0. U každé fotografie se takto zapíšou souřadnice do XML souboru na paměťovou SD kartu. Ke komunikaci s modulem čtečky paměťových SD karet používám knihovnu *[SdFat]*[7] autora *Williama Greimana*. Forma, v které jsou souřadnice uloženy, mi ukládá otevřený standart *PapyWizard*, který je podporován většinou programů, zabývajících se panoramatickou fotografií. Při začátku focení se zapíše do konečného XML souboru, pojmenovaném číslem, které udává, o kolikáté panorama se jedná, kdy je tato informace uložena v EEPROM paměti a je tedy uchována i po odpojení napájení, nejdříve hlavička souboru. V té jsou uloženy informace o použitém ohnisku objektivu, začátku a konci focení, velikosti senzoru fotoaparátu a další pomocné informace. Poté se zapisují data s pozicemi jednotlivých fotek a nakonec neměnné ukončení souboru.

Pokud je takto soubor zapsán, je možné jej načíst v programu pro tvorbu panoramat a ten seřadí fotky podle vepsaných souřadnic. Má to ještě jednu velkou výhodu. Je nad možnosti dnešního software najít společné body mezi fotografiemi zachycujících oblohu. Protože je ale známa přesná pozice ve snímku, není nutné, aby tyto body mezi snímky byly nalezeny. Odpadne tak nutnost složitého dokreslování oblohy v grafických editorech, což značně ušetří čas.

2.3.3 Datum a Čas

Zařízení obsahuje modul reálného času DS1307, který si díky vlastní baterii uchovává informaci i po odpojení hlavního zdroje napájení. Pro komunikaci s modulem bylo využíváno funkcí veřejně dostupné knihovny *[RTClib]*[8] společnosti *JeeLabs*.

V motorizované panoramatické hlavě nejsou funkce času a data klíčové, slouží tak spíše jako informativní služba. Informace o čase jsou zapisovány do XML souboru, díky čemuž má člověk přehled, kdy bylo jaké panorama pořízeno. Čas a datum je pak možno také vyvolat přímo z menu uživatelského rozhraní. Seřízení údajů v tuto chvíli probíhá při kompilaci a nahrávání programu do jednotky Arduino. V budoucím vývoji se počítá s využitím informace z GPS signálu.

2.3.4 Teploměr

Jedná se o informativní službu dostupnou z uživatelského rozhraní. Využit je číslicový digitální teploměr, k jehož komunikaci se využívá sběrnice One Wire, která je dostupná přes standardně dodávanou knihovnu k jednotce Arduino, *[Wire]* autora *Nicholase Zambettiho* a knihovny *[DallasTemperature]*[9]. Informace je získávána ve stupních Celsia a je přímo zobrazována na displeji.

Větší, než informativní hodnotu, má informace o teplotě v Astro-fotografii, kde je teplota přímo úměrná množství šumu zaznamenaném na fotografii. Pomocí odečtu teploty je tak možné upravit nastavení fotoaparátu k omezení šumu na co nejmenší úroveň.

2.3.5 Baterie

Protože celé zařízení je napájeno akumulátorem, je nutné mít informaci o tom, jak je baterie nabitá. K tomuto účelu se dá využít jeden z analogových pinů na Arduinu. Na tomto pinu je možné nastavit velikost referenčního napětí na 5, 3.3 nebo 1.1V. To umožní si zvolit jemnost rozlišení odečítané hodnoty napětí baterie. Zvolil jsem hodnotu 1.1V. Protože je ovšem napětí nabité baterie přibližně 10.5V, bylo nutné pomocí napěťového děliče velikost napětí snížit pod 1.1V správným zvolením hodnot odporů. Pro výpočet hodnot platí:

$$U_1(V) = U \cdot \frac{R_1}{R1 + R2} \tag{13}$$

$$U_2(V) = U \cdot \frac{R_2}{R1 + R2} \tag{14}$$

Ze vztahů vypočítáme poměr, ve kterém musí být odporyR_1 a R_2 pro dosažení chtěného výstupního napětí.

Implementaci kódu pak bude vypadat následovně:

```
value = analogRead(analogInput); // cteni dat z analogoveho pinu

vout = (value * 1.1) / 1023.0; // prepocet vuci referencnimu napeti a rozliseni pinu
vin = vout / (R2/(R1+R2)); // zjisteni napeti baterie
```

Výpis 6: Odečítání napětí baterie

V druhém řádku vloženého kódu jsou data získané z analogového pinu násobena referenčním napětím o hodnotě 1.1V. Tento mezi výpočet je poté vydělen číslem 1023. To říká, že data na analogovém pinu je možno číst s 10 bitovým rozlišením. Teoreticky tedy bude možné zjistit napětí baterie s přesností na 1 mV. V poslední části kódu se vydělí *vout* napětím U_2 jak je uvedeno ve vzorci výše. Teď už se jen výsledek podělí číslem 10 a zůstane napětí baterie ve Voltech.

Protože většině uživatelů nemusí zobrazené napětí říkat nic o stavu baterie, převádím pomocí funkce map(value, fromLow, fromHigh, toLow, toHigh) napětí na procenta. Do této funkce jsem dosadil přibližné maximální napětí, kterým baterie disponuje. Protože přesně vím, při kterém napětí přestává fungovat řadič krokového motoru, tak bylo minimální napětí rovno této hodnotě.

2.3.6 Reproduktor

Jelikož robot pracuje při práci zcela automaticky a není nutné jej neustále kontrolovat, bylo potřebné implementovat funkci jednoduché zvukové signalizace při ukončení činnosti. Využit je piezo reproduktor s rezonančním kmitočtem 4.1 kHz.

Aby mohl být zvuk reprodukován, tak je využita možnost mikrokontroleru tvořit PWM signál. U Arduina pracuje defalutně na frekvenci okolo 500Hz, což poslouží k tvorbě tónů v slyšitelném spektru.

K reprodukci přijatelně přesných tónů je potřeba nejdříve zjistit, kolik mikrosekund musí být u každého tónu signál na log(1). K tomu lze využít následující vzorec:

$$doba(mikrosekund) = \frac{perioda}{2} \tag{15}$$

kde

$$perioda(mikrosekund) = \frac{1}{frekvence} \tag{16}$$

Frekvence jednotlivých tónů jsou obecně známé, je tak možné spočítat délku v mikrosekundách na log(1) pro každý tón. Pro jednu oktávu, tedy 8 tónů je výsledek následující:

nota	$frekvence(Hz)$	$perioda(\mu s)$	$dobaNaLog(1)(\mu s)$
c	261	3830	1915
d	294	3400	1700
e	329	3038	1519
f	349	2864	1432
g	392	2550	1275
a	440	2272	1136
h	493	2028	1014
C	523	1912	956

Tabulka 1: Definice jednotlivých tónů

Následně je potřeba funkcí, které dle výše vypočtených parametrů přehrají dané tóny, přidají rytmus a vytvoří tak melodii.

Protože jsou známé časy, po které musí být pin na log(1), implementace je přímočará. V tomto kusu kódu je možné vidět funkci pro přehrání tónu se zvolenou délkou znění. Různá délka zvuku se realizuje vícenásobným opakováním přehrávání tónu.

```
for (long i = 0; i < duration * 1000L; i += tone * 2)
{
    digitalWrite (speakerPin, HIGH); // log(1)
   delayMicroseconds(tone); // mame log(1) po zvolenou dobu
    digitalWrite (speakerPin, LOW); // log(0)
   delayMicroseconds(tone); // mame log(0) po zvolenou dobu
 }
```

Výpis 7: Přehrání tónu s nastavitelnou délkou

Chceme-li přehrát více tónů za sebou, zapíšeme je do *char* pole *notes[]*. *For* smyčka pak toto pole projde, přečte jednotlivé tóny a předá je funkci pro přehrání tónu. Kromě samotných tónů je také možné zapisovat rytmus pomocí číselných hodnot. Tyto hodnoty definují délku znění tónů a funkci pro přehrávání je předávána proměnou *duration*. Pro implementaci této funkcionality jsem využil zdroje [2].

2.4 Astro-fotografie

Díky přesnosti instalovaných krokových motorů, je možné robota využít i k astro-fotografii.

Jedná se o velmi náročný druh fotografie, kdy je potřeba vyfotit objekty se světelností mnohonásobně menší, než je při běžné fotografii. Aby se tak zachytilo dostatečné množství dat, je nutné na fotoaparátech používat dlouhé expozice. Tady ovšem nastává zásadní problém. Objekty na obloze se pohybují. Pohybují se po kružnici kolem středu nazývaném nebeský pól, který lze najít nedaleko hvězdy Polárka.

Aby bylo možné fotit s dlouhou expozicí a přitom na snímku zůstaly hvězdy bodové, je nutné jejich pohyb po obloze kompenzovat. To je schopen vykonat robot. Ten zjednodušeně řečeno kopíruje pohyb hvězd a tím kompenzuje jejich pohyb.

Jako první je potřeba robota nastavit do správné pozice. Nejprve jej nasměrujeme k severu. V tuto chvíli je to prováděno pomocí ručního kompasu, v budoucím vývoji se počítá s využitím jiných technologií. Směřuje-li hlava k severu, je nutné nastavit velikost náklonu tak, aby kolmice na vertikální osu otáčení směřovala k nebeskému pólu. Tento úhel je roven zeměpisné šířce. V tuto chvíli je hodnota nastavena fixně, v budoucnu bude využit signál GPS. K správnému nasměrování pomáhá akcelerometr, kdy je žádoucí robotem pohnout tak, aby se na displeji zobrazila nula. Pokud je zeměpisná šířka rovna $50\,^\circ$ a robotická hlava se nachází v horizontální rovině, zobrazuje se na displeji číslo 50. Jakmile je robotem směrováno k nebeskému pólu, číslo se zmenšuje dokud nedosáhne nuly.

Stejně jako při focení panoramat je možné před samotným začátkem nastavit odpočet. To probíhá pomocí stejné funkce. Po odpočtu dojde k simultánnímu pohybu obou motorů. Protože je známo, o jaký úhel na pomyslné kružnici se hvězdy pohnou, je možné spočítat pohyb pro jakoukoli dobu. Za 24 hodin je úhel $360\,^\circ$. Pro výpočet hodnoty úhlu, která bude hvězdami uražena za minutu, postupujeme následovně:

$$uhel(stupne) = \left(\frac{360}{24}\right)/60 \tag{17}$$

Za minutu urazí úhel $0.25\,^\circ$. Z praktických testů vyšlo najevo, že minimální úhel, o který se spolehlivě dokáží motory pohnout je $0.125\,^\circ$. Tedy úhel, které hvězdy na kružnici urazí za 30 sekund. Díky tomu je možné fotit objekty na širších ohniskových vzdálenostech.

Provede-li se pohyb motorů, čeká se 30 sekund a poté se cyklus opakuje. Před samotným spuštěním pohybu je možné nastavit, kolik se bude provádět fotografických expozic a jak dlouho bude expozice trvat. Tím se nastaví celková dobu focení. Proces je tedy zcela automatizovaný a není potřeba po každé expozici nového nastavování.

3 Testování v různých podmínkách

Protože se jedná o zařízení, kdy je nezbytně nutná spolehlivost, bylo potřebné provést řádné otestování v různých podmínkách. Není přípustné, aby při focení panoramatu, který se skládá z 1000 snímků, jeden chyběl. Program proto musí být odladěn pro spolehlivý chod a hardware musí být dostatečně chráněn před vnějšími vlivy.

3.1 Teplota

Okolní teplota může mít velký vliv na elektronická zařízení. Bývá příčinou chybných měření z čidel a také může vést k přehřátí elektroniky. Testy byly provedeny při různých teplotách s aktivovanou funkci focení panorama.

- **Velmi nízké teploty** - průměrná teplota $-15\,°C$. Měření probíhalo ve venkovních prostorách. Byla zjištěna značně pomalejší reakce při zobrazení informací na displeji, která ale na funkčnost neměla většího vlivu. Znatelnější potíže způsobila teplota vestavěnému akumulátoru, kdy se kapacita snížila o celých 22%. Díky naddimenzovanosti akumulátoru je ovšem možné i s takovým snížením kapacity vyfotit i velké panorama. Na práci ostatní elektroniky neměla teplota znatelný vliv.

- **Nízké teploty** - průměrná teplota $0\,°C$. Měření ve venkovních prostorách. Při této teplotě byla zjištěna jen nepatrně nižší reakce displeje. Kapacita akumulátoru se snížila o 4%. Další omezení nebyla zaznamenána.

- **Normální teploty** - průměrná teplota $15\,°C$. Při této teplotě pracoval robot zcela spolehlivě bez jakéhokoli omezení. Kapacita baterie se nelišila od referenční hodnoty.

- **Vysoké teploty** - průměrná teplota $30\,°C$. Foceno ve vnitřních prostorách. U kapacity baterie nebyl zaznamenán znatelný pokles. Při této teplotě byla měřena i teplota povrchu baterie, krokového motoru a mikrokontroleru Arduina. Teplota baterie se po 10 minutách ustálila na $34\,°C$, krokový motor vykazovat průměrnou teplotu $31.5\,°C$ a mikrokontroler $36\,°C$. Tyto měření byla vyhodnocena jako uspokojivá a podle zjištěných výsledků nebude třeba řešit dodatečné chlazení.

Jako referenční hodnota kapacity akumulátoru byla sada pěti focení při teplotě $15\,°C$, které probíhalo v uzavřené místnosti při plném nabití akumulátoru až do doby, kdy přestaly fungovat motory, což si děje při poklesu napětí pod 7.2 V. U každého z pěti testu byl zaznamenán počet vyfocených snímků a tyto hodnoty zprůměrovány.

Testování při rozličných teplotách ukázalo spolehlivost práce robota. Za celou dobu testování nebyla zaznamenána chyba řídícího software, či elektroniky.

3.2 Vlhkost

Vlhkost může mít jak krátkodobý vliv na elektroniku, tak i dlouhodobý. Při vysoké vlhkosti může dojít v krajním případě ke zkratu a zničení elektroniky. Hlavní vliv má

ovšem vlhkost z dlouhodobého hlediska, kdy je spolu s teplotou činitelem koroze a růstů plísní. Tyto dva jevy mohou mít zásadní vliv na funkčnost a jsou schopné zničit zařízení. Dlouhodobé působení na mé zařízení nejsem v této chvíli schopen komentovat, kvůli nedostatečnosti relevantních dat.

Otestován byl ovšem krátkodobý vliv ve velmi suchém a naopak velmi vlhkém prostředí.

- **velmi suché prostředí** - test probíhal v uzavřeném prostředí s uměle sníženou vlhkostí. V takovém prostředí je riziko výskytu statické elektřiny, který může poškodit elektronická zařízení, či způsobit nestabilitu vykonávání řídícího software. To ovšem nebylo potvrzeno a robot pracoval dle očekávání.
- **velmi vlhké prostředí** - robot byl testován ve venkovním prostředí při mírném dešti. Bylo vyfoceno testovací panorama o velikosti 400 snímků. Při procesu nebyl zaznamenán jakýkoli problém s funkčností. Neočekává se ovšem, že by zařízení mělo v těchto podmínkách běžně pracovat.

3.3 Prezentace na dni otevřených dveří

Nezanedbatelným testem zařízení byla účast na Dni otevřených dveří katedry informatiky. Při tomto dnu zařízení pracovalo s drobnými přestávkami 4 hodiny. Řídící software po celou dobu fungoval bez problému, byla zkoušena veškerá funkcionalita přístroje, fotoaparát během dne vyfotil 70 GB dat. Vliv na test také měli návštěvníci, kteří si zařízení sami mohli vyzkoušet, značně zatěžovali software zkoušením funkcí a samotnou stavbu robota.

3.4 Reálný provoz

Po těchto testech došlo i k plnému otestování v reálných podmínkách. Dne 10.2.2014 byl z hory Huk pořízen panoramatický snímek města Vrbna pod Pradědem s okolními kopci Hrubého Jeseníku. Chod hardware i software zařízení se ukázal jako bezproblémový. Během celého procesu focení nedošlo k žádným obtížím a všechny potřebné snímky byly vyfoceny tak, jak bylo plánováno. Zde je několik parametrů výsledného snímku:

- **Počet vyfocených snímků:** 1200
- **Délka focení:** 80 minut
- **Optické rozlišení snímku:** $198660 \times 37244 = 7.4$ Gpix
- **Horizontální úhel záběru:** $191\,^{\circ}$
- **Vertikální úhel záběru:** $32\,^{\circ}$
- **Velikost na disku (formát PSB):** 31.3 GB
- **Velikost pro web (formát JPEG):** 1.15 GB

- **Počet dílčích snímků pro webové prohlížení:** 38 144

Na samotný snímek je možné se podívat na <http://www.gpix.eu>

4 Závěr

Vývoj řídícího software motorizované panoramatické hlavy ukázal, že při přijatelných finančních nákladech lze vyvinout kvalitní zařízení převyšující svou kvalitou mnohem dražší, komerční produkty. Byla provedena implementace funkcí k pořízení kvalitních panoramatických snímků ve velmi vysokém rozlišení. Spolehlivost byla otestována v rozličných podmínkách a ověřena v reálném provozu.

Toto zařízení ovšem nabízí velký prostor pro další vývoj. Počítá se s vývojem následujících částí:

- **Časosběr** - umožní pořizovat časosběrná videa. Ty mohou například zaznamenávat stavbu výškové budovy, kdy za využití motorů bude sledována postupná stavba, či sledovat růst rostlin. Robot, s uživatelem zvolený časový interval, pořídí snímek. Také bude definováno, jaký druh pohybu má během procesu vykonat. Počítá se s možností složitějších pohybů, nadefinovaných předem v počítači. Po dokončení focení se snímky složí do plynulého videa.

- **Rozhraní mezi robotem a fotoaparátem** - tato implementace otevře obrovskou škálu dalších možností vývoje. Bude potřeba implementovat Picture Transfer Protocol (PTP), který umožní z prostředí robota ovládat veškerou funkcionalitu fotoaparátu. Bude možné například:

  - **kontrolovat, zda byl snímek vyfocen** - fotoaparát podá softwaru motorizované hlavy zprávu, že byl snímek vyfocen a uložen na kartu. Až po tomto potvrzení se motorizovaná hlava pohne k dalšímu snímku.
  - **optimalizovat rychlost focení** - díky zpětné vazbě o tom, jak je zaplněna vyrovnávací paměť fotoaparátu a znalostí parametrů focení (hlavně času závěrky) bude možné pořizovat panoramata ve vyšší rychlosti bez snížení kvality výstupu.
  - **fotit panorama s velkým dynamickým rozsahem** - místo jednoho snímku pro každou část záběru se jich vyfotí několik s odlišným nastavením fotoaparátu. Ve výsledku vznikne panorama s dynamickým rozsahem, kterého by klasickou metodou focení nebylo možné dosáhnout.

- **Ovládání z mobilního zařízení** - spolu s PTP protokolem umožní veškeré ovládání funkcí robota i fotoaparátu pomocí mobilního zařízení. Počítá se s komunikací přes Bluetooth protokol. V tuto chvíli je naplánován vývoj pro operační systém Android.

5 Reference

[1] Manthani, Anim Monilal URL: <http://theccontinuum.com/tag/adxl345/> [cit. 2014-03-01]

[2] Arduino, *Melody* URL: <http://www.arduino.cc/en/Tutorial/melody> [cit. 2014-03-01]

[3] Malpartida, Francisco URL: <http://arduino.cc/en/Reference/LiquidCrystal> [cit. 2014-03-06]

[4] McCauley, Mike URL: <http://www.airspayce.com/mikem/arduino/AccelStepper/> [cit. 2014-03-06]

[5] Liu, John URL: <http://liudr.wordpress.com/libraries/phi_super_font/> [cit. 2014-03-06]

[6] Love Electronics, *ADXL345* URL: <https://github.com/landis/saphe-black-box/tree/master/libraries/ADXL345> [cit. 2014-03-10]

[7] Greiman, William URL: <http://code.google.com/p/sdfatlib/> [cit. 2014-03-11]

[8] JeeLabs, *RTClib* URL: <https://github.com/adafruit/RTClib> [cit. 2014-03-11]

[9] Burton, Miles URL: <http://arduino.cc/en/reference/wire> [cit. 2014-03-11]

A Obsah přiloženého datového nosiče

- text bakalářské práce
- zdrojový kód řídícího software motorizované panoramatické hlavy

B Obrazová příloha

Obrázek 10: Stavba robota