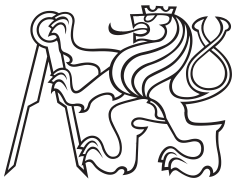
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra měření

# Meteorologická stanice pro autonomní provoz bezpilotních prostředků

Marek Šach

Vedoucí práce: Ing. Tomáš Meiser
Studijní program: Otevřená Informatika
Studijní obor: Počítačové systémy
Květen 2018

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šach** Jméno: **Marek** Osobní číslo: **457109**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačové systémy**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Meteorologická stanice pro autonomní provoz bezpilotních prostředků**

Název bakalářské práce anglicky:

**Meteorological Station for Autonomous Operation of Unmanned Aerial Vehicles**

Pokyny pro vypracování:

1. Prostudujte dokumentaci pro cílovou výpočetní jednotku a proveďte rešerši dostupných meteorologických senzorů.
2. Vyviňte vhodné rozhraní pro získávání dat ze zvolených senzorů a implementujte algoritmy pro stabilní zpracování získaných dat.
3. Proveďte rešerši známých postupů pro predikci vývoje počasí na základě změny dostupných veličin a implementujte zvolený algoritmus
4. Proveďte rešerši existujících otevřených meteorologických sítí a zdrojů. Navrhněte rozhraní pro získávání dat z těchto sítí.
5. Získejte data z měření v reálném nasazení jednotky a zhodnoťte výsledky měření a predikce.
6. Zpracujte dokumentaci projektu po softwarové a hardwarové stránce a návrh implementace řešení pro provoz autonomních bezpilotních prostředků.

Seznam doporučené literatury:

[1] Marwedel, Peter: Embedded system design. Vol. 1. New York: Springer, 2006.
[2] Lee, Edward Ashford, and Sanjit A. Seshia: Introduction to embedded systems: A cyber-physical systems approach. MIT Press, 2016.
[3] Douglas, C. K. M.: Local Weather Forecasting. Weather 12.9 (1957): 269-274.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Tomáš Meiser, katedra počítačů FEL**

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

Datum zadání bakalářské práce: **12.01.2018** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce:
**do konce letního semestru 2018/2019**

| Ing. Tomáš Meiser<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | prof. Ing. Pavel Ripka, CSc.<br>podpis děkana(ky) |
|---|---|---|

## III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

# Poděkování

Děkuji vedoucímu Ing. T. Meiserovi za spolupráci a vedení při tvorbě této práce.

# 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, 24. května 2018

# Abstrakt

Cílem této bakalářské práce je navrhnout a implementovat meteorologickou stanici pro podporu budoucího plně samostatného provozu autonomních bezpilotních prostředků (UAV).

Řešení sestává z volby vhodných hardwarových a softwarových prostředků, návrhu systému, jeho implementace a také jeho testování. Výsledkem práce je funkční meteorologická stanice, která umí pomocí senzorů měřit základní meteorologická data, ukládat je v databázi a posléze je poskytnout jak ve formě vhodné pro strojové zpracování, tak v podobě čitelné pro člověka.

**Klíčová slova:** meteorologická stanice, Raspberry Pi, počasí, IoT, senzory, GUI

**Vedoucí práce:** Ing. Tomáš Meiser

# Abstract

The aim of this bachelor thesis is to design and implement a meteorological station to support future fully autonomous UAV[1] operation.

The solution consists of choosing suitable hardware and software tools, designing the system, implementation and also testing. The result of this project is a functioning meteorological station, which is able to measure basic meteorological data using sensors, store it in a database and then provide it in a form suitable for machine processing, as well as in human readable form.

**Keywords:** meteorological station, Raspberry Pi, weather, IoT, sensors, GUI

**Title translation:** Meteorological Station for Autonomous Operation of Unmanned Aerial Vehicles

[1] unmanned aerial vehicles

# Obsah

# Obrázky



# Tabulky

# Kapitola 1
# Úvod

## 1.1 Základní popis

V současné době můžeme pozorovat značný pokrok v oblasti autonomních bezpilotních prostředků. Ty mohou mít v praxi velice různorodé využití. Stejně jako u jakýchkoliv jiných letounů je však jejich provozuschopnost do značné míry ovlivněna počasím. Aby autonomní letoun mohl spolehlivě a bezpečně plnit svůj úkol, potřebuje znát aktuální stav počasí a ideálně také relativně přesnou krátkodobou předpověď vývoje počasí v lokalitě, kde bude tento prostředek operovat.

K poskytování aktuálních informací o počasí v konkrétní lokalitě poslouží právě meteorologická stanice vytvořená v rámci této bakalářské práce. Pomocí senzorů bude tato meteostanice schopna získávat meteorologická data. Tato data bude umět následně zpracovat, uchovat a letounu poskytnout.

## 1.2 Cíle práce

Cílem této práce je postupně provést a popsat jednotlivé kroky, které vedou k vytvoření meteostanice.

Jádrem celé meteorologické stanice bude počítač Raspberry Pi 3, model B. Prvním úkolem bude hardwarový návrh stanice sestávající z výběru vhodných senzorů a návrhu jejich zapojení. Současně bude třeba provést softwarový návrh celého systému a zvolit vhodné prostředky k jejich implementaci.

Práce je členěna do několika kapitol. Nejprve zmíním obecný návrh celého systému po hardwarové i softwarové stránce, včetně volby technologií, které jsem využil. Dále se v této práci budu podrobněji věnovat zpracování dat z jednotlivých senzorů a popíši testy, které jsem s jednotlivými senzory provedl. Následně uvedu podrobněji detaily implementace jednotlivých modulů zajišťujících softwarovou funkcionalitu meteostanice. Nakonec zhodnotím práci meteostanice nasazené v praxi.

# Kapitola 2
# Návrh řešení

## 2.1 Výběr senzorů

Základní senzory, které jsem vybíral, jsou teploměr, vlhkoměr, barometr a luxmetr (senzor svítivosti). Mnoho takových dostupných senzorů umí naměřená data poskytovat již v digitální podobě přes běžná komunikační rozhraní (např. I2C, SPI, 1-wire). Raspberry Pi 3 B disponuje mimo jiné periferiemi pro komunikaci přes rozhraní SPI, I2C i 1-wire, což bylo vhodné při výběru senzorů zohlednit a následně s výhodou využít.

Po průzkumu dostupných senzorů jsem vybral senzory následující:

- **DHT22**
  Senzor pro měření teploty a relativní vlhkosti, taktéž označovaný jako AM2302. Teplotu senzor měří v rozsahu -40 až 80 °C s přesností ±0.5 °C. Relativní vlhkost měří v rozsahu 0 až 100% s chybou ±2%. Naměřená data umí převést a poskytnout již v digitální podobě, pro komunikaci pak využívá vlastní 1-wire interface. [6]

- **BMP280**
  Senzor tlaku a teploty, podporuje komunikaci přes rozhraní I2C. Rozsahem senzoru je 300 až 1100hPa a maximální odchylka ±1hPa. Při teplotě vzduchu 25 °C je odchylka pouze ±0.12hPa (odchylka se zvětšuje s rozdílem teploty od 25 °C). Senzor podporuje komunikaci přes I2C i SPI, ve své implementaci jsem využil rozhraní I2C pro maximální jednoduchost zapojení. [7]

- **TSL2561**
  Luxmetr TSL2561, stejně jako barometr BMP280, podporuje komunikaci přes rozhraní I2C. Výhodnou vlastností tohoto senzoru je fakt, že oproti mnohým jiným luxmetrům využívá k měření dvě fotodiody. Jedna je citlivá na obojí, infračervené i viditelné světlo, zatímco druhá je citlivá jen na infračervené světlo. Díky možnosti odečíst infračervenou složku tento senzor tedy umožní získat informaci o osvětlení ve viditelném spektru. To u luxmetrů disponujících pouze jednou fotodiodou není možné, právě kvůli přítomnosti infračervené složky v jimi měřeném osvětlení. [2]

**Obrázek 2.1:** Rozebraný srážkoměr, zdroj obrázku: [5]

Dalšími třemi senzory, které jsem využil, jsou srážkoměr, anemometr a korouhvička. Konkrétně jsem zvolil senzory, které se prodávají jako náhradní díly k meteostanici WH1080 [5]. Oproti doposud zmíněným senzorům tyto senzory již nebudou komunikovat přes standardní komunikační rozhraní, nýbrž pomocí jednotlivých tiků, případně analogového signálu.

Prvním senzorem z této skupiny je **srážkoměr**. Zvolený typ funguje na principu dvou vaniček, připevněných na společné ose. Když prší, srážky se zachycují do jedné ze dvou vaniček. Když se ve vaničce nashromáždí definované množství vody, vanička se převáží, vyprázdní a začne se plnit druhá vanička. Během tohoto převážení projde magnet kolem jazýčkového spínače, čímž se vygeneruje tik, který je možné na Raspberry Pi zachytit a zpracovat na libovolném GPIO pinu. Každý takový tik tedy reprezentuje definované množství zachycených srážek. Vnitřek srážkoměru (převažující se vaničky) je zobrazen na obrázku 2.1.

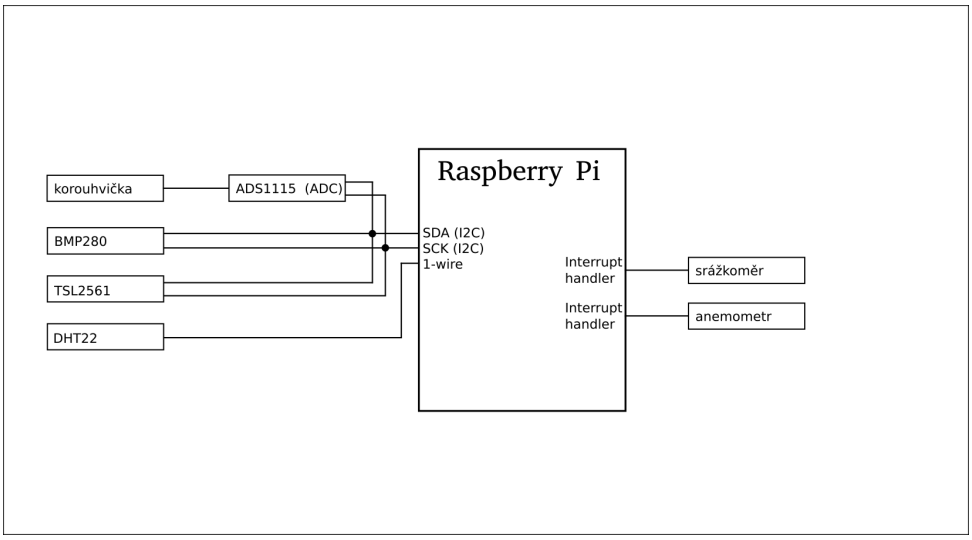
**Anemometr**, sloužící k měření rychlosti větru, sestává ze statické a pohyblivé části. Na pohyblivé části jsou přimontovány lopatky se zakončením ve tvaru duté polokoule, díky kterým se za větrných podmínek začne pohyblivá část anemometru otáčet. Během jedné otáčky anemometru způsobí průchody magnetu kolem jazýčkového spínače dva tiky, které lze na vstupním pinu Raspberry Pi zpracovat.

Posledním senzorem je **korouhvička**, sloužící k určení směru větru. Jako jediný z použitých senzorů bude korouhvička reprezentovat naměřená data pomocí analogového signálu. Korouhvička, stejně jako anemometr, sestává z

otáčivé části (která se otáčí v závislosti na směru větru) a nepohyblivé části. V nepohyblivé části je 8 rezistorů o různém odporu. Podle natočení pohyblivé části korouhvičky dojde k zapojení jednoho, případně dvou sousedních odporů do vnitřního obvodu korouhvičky, což způsobí různý úbytek výstupního napětí senzoru pro různá natočení korouhvičky.

Jelikož Raspberry Pi nemá zabudovaný analogově digitální převodník, je v případě korouhvičky třeba využít převodník externí. Ten je schopen data z korouhvičky zpracovat a dále je poskytnout již prostřednictvím standardního digitálního komunikačního rozhraní.

Z dostupných AD převodníků jsem zvolil model ADS1115. Zpracovanou napěťovou úroveň umí převést na 16-bitové číslo, které dále poskytuje přes rozhraní I2C [4].

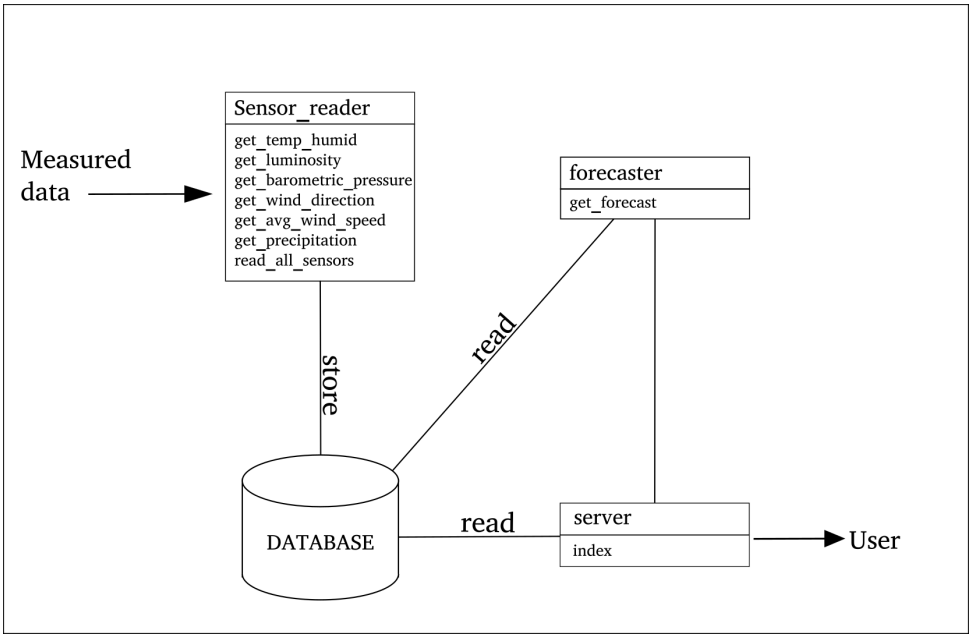



**Obrázek 2.2:** HW schéma

## 2.2 SW návrh

Softwarové řešení meteostanice sestává z několika modulů. Prvním je modul umožňující čtení a zpracování dat ze senzorů. Dále je třeba data ukládat. K tomuto účelu jsem navrhl a vytvořil databázi. Dalším krokem je příprava rozhraní, přes které bude možno ke zpracovaným datům přistupovat. Zde jsem jednak s výhodou využil databázového serveru, který vzdálený přístup umožňuje ze své podstaty. Pro interpretaci dat způsobem, který je čitelnější pro člověka, jsem navíc vytvořil webový server a na něm webovou stránku, která vizualizuje vybraná data. Posledním modulem tohoto projektu je jednoduchý tvůrce předpovědi počasí na základě lokálně měřených dat. Schéma softwarového návrhu je vizualizováno na obrázku 2.3

Pro implementaci jednotlivých modulů jsem zvolil programovací jazyk Python, kód je psán pro interpret verze 2.7. Důvodem volby jazyka Python byla jeho univerzálnost a snadná využitelnost pro implementaci všech




**Obrázek 2.3:** SW schéma

požadovaných modulů.

Při implementaci modulu pro sběr dat jsem s výhodou využil širokou knihovní podporu pro práci se vstupně výstupními piny, periferiemi, či dokonce pro komunikaci přímo s konkrétním typem senzoru. Pro tvorbu webového serveru existuje v Pythonu několik frameworků, které lze s výhodou využít. Jeho vhodnost pro implementaci webového serveru dokládá i jeho hojné využití v praxi. A pro propojení s databázovým serverem pro Python samozřejmě opět existuje knihovní podpora, která toto propojení umožní.

Celkově jsem tedy Python vyhodnotil jako komplexní nástroj, který umožní elegantní implementaci veškeré požadované funkcionality.

Pro tvorbu databáze jsem zvolil MySQL server, který lze na zařízení s Linuxovým operačním systémem (tedy i Raspberry Pi, na kterém běží OS Raspbian) snadno nainstalovat a využít.

# Kapitola 3

# Zpracování dat ze senzorů

V této kapitole se budu podrobněji věnovat zpracování dat z jednotlivých senzorů. Nejprve popíši hardwarový návrh stanice a uvedu schéma zapojení. Následně podrobněji vysvětlím princip zpracování dat z jednotlivých senzorů a představím testy, které jsem s jednotlivými senzory provedl.

## 3.1 Schéma zapojení

Jak jsem již podrobněji popsal v kapitole *Návrh řešení* v sekci *Výběr senzorů* 2.1, ke svému řešení jsem zvolil tři senzory komunikující přes digitální rozhraní (teploměr/vlhkoměr DHT22 přes 1-wire, TSL2561 a barometr BMP280 přes I2C), dále korouhvičku připojenou přes AD převodník taktéž využívající I2C rozhraní a nakonec srážkoměr s anemometrem generující jednotlivé tiky. Zjednodušené schéma návrhu bylo naznačeno na obrázku 2.2.

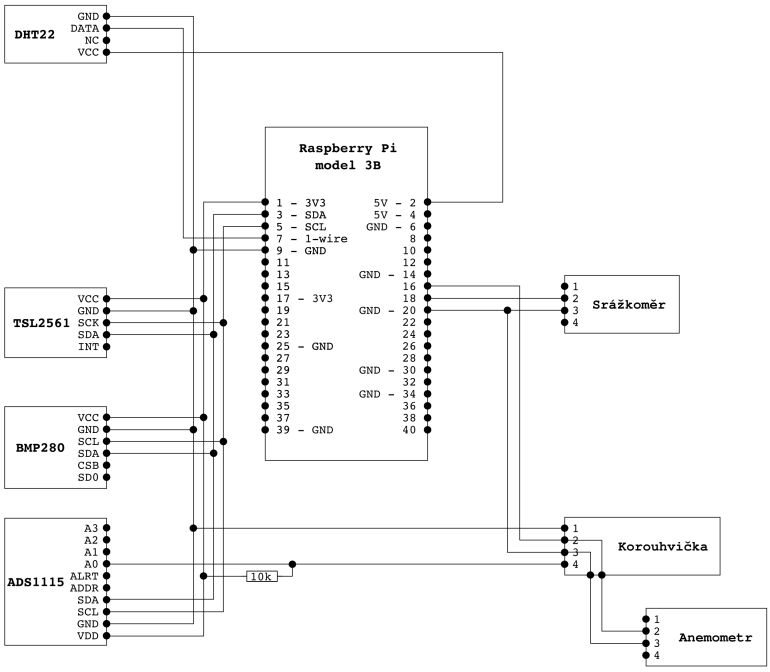
Při návrhu zapojení je nezbytné brát na zřetel specifické parametry zvolených modulů, především povolené napájecí napětí. Překročení mezních hodnot může způsobit nevratné poškození použitých modulů. Výpis základních informací z dokumentací k jednotlivým použitým senzorům a AD převodníku je uveden v tabulce 3.1.

| Součástka | min $U_{CC}$ [V] | max $U_{CC}$ [V] | typické $U_{CC}$ [V] | Komunikační rozhraní |
|---|---|---|---|---|
| DHT22 | 3,3 | 6 | 5 | 1-wire (vlastní) |
| TSL2561 | 2,7 | 3,6 | 3 | I2C |
| BMP280 | 1,71 | 3,6 | 1,8 | I2C, SPI |
| ADS1115 | 2 | 5,5 | - | I2C |

**Tabulka 3.1:** Specifikace použitých digitálních součástek

Moduly komunikující přes I2C nebo 1-wire je jednoduché zapojit, stačí je připojit k odpovídajícím pinům dané periferie na Raspberry Pi. Jediné, co je třeba provést obezřetně, je volba napěťových úrovní komunikačních linek. Tu je třeba volit s ohledem na hodnoty v tabulce 3.1.

Anemometr a srážkoměr generují tiky pomocí jazýčkového spínače. Generované tiky každého z těchto dvou senzorů jsou čteny jedním GPIO pinem. Při jejich zapojení je klíčové zajistit, aby nedošlo k překročení maximálních povolených hodnot napětí a proudu na pinech Raspberry Pi. Nedodržení těchto




**Obrázek 3.1:** Schéma zapojení

omezení by mohlo vést k nevratnému poškození Raspberry Pi. Abych dodržel maximální povolené hodnoty napětí a proudu na pinech Raspberry Pi, využil jsem jejich interní pull-up rezistor, který pin přivede ke správnému napětí pro úroveň high a při sepnutí jazýčkového spínače (propojením příslušného pinu se zemí) zajistí, že proud na pinu nepřekročí povolenou hodnotu.

V případě korouhvičky potřebujeme měřit úbytek napětí, které způsobí její interní rezistory. Pro tento účel musím korouhvičku zapojit do série s dalším rezistorem a vytvořit takto napěťový dělič. Při takovém zapojení se bude úbytek napětí na korouhvičce měnit v závislosti na velikosti vnitřního odporu korouhvičky. Tento úbytek je již měřitelný AD převodníkem, který bude připojen na I2C sběrnici.

Při dodržení popsaných postupů jsem vytvořil finální schéma zapojení, které je uvedeno na obrázku 3.1

## 3.2 Zpracování přerušení

Data naměřená anemometrem a srážkoměrem jsou reprezentována pomocí jednotlivých tiků. Prvním tématem, kterému se tedy budu věnovat, bude otázka, jakým způsobem detekovat a zpracovávat data ve formě samostatných tiků.

Jednou možností by bylo nakonfigurovat pin jako běžný vstupní pin a v nekonečné smyčce se na něj neustále dotazovat. Tento přístup je ovšem v případě implementace meteostanice nevhodný. Prvním nedostatkem je fakt,

že neustálé dotazování se v nekonečné smyčce by bylo značně výpočetně náročné a výrazně by se tím plýtvalo výpočetním výkonem počítače.

Druhým, a mnohem vážnějším, problémem je fakt, že při tomto přístupu může být tik snadno přehlédnut. Dotazování se na vstupní pin totiž reálně neprobíhá (nebo alespoň nemusí probíhat) nepřetržitě. Je nutné mít na paměti, že na Raspberry Pi běží operační systém. Na něm může běžet mnoho procesů současně, podporuje také vícevláknové aplikace. To znamená, že pokud cyklus s dotazováním se na pin poběží v samostatném vlákně, pravděpodobně se v běhu bude muset střídat jinými vlákny programu a dalšími procesy, které na Raspberry právě běží. Pokud tik proběhne v době, kdy dotazování neběží (běží právě jiné procesy/vlákna), bude přehlédnut.

Zvýšená spotřeba výpočetního výkonu je v případě implementace meteostanice pouze nešikovná, nicméně není zásadním problémem. Naproti tomu nezaznamenání některých tiků je problém zásadní, kvůli kterému není přístup s dotazováním pro meteostanici použitelný.

V praxi mohou existovat dvě okolnosti, při kterých je přístup s dotazováním korektně použitelný. První možností je případ, kdy jsou sledované tiky velmi dlouhé. U takového systému lze předpokládat, že bude tik dost dlouhý na to, aby nestihl celý proběhnou v době, kdy je vlákno s dotazovacím cyklem operačním systémem uspáno. Druhou možností jsou jednoúčelové aplikace, při jejichž implementaci stačí použití mikrokontroleru, typicky bez operačního systému, na kterém nebude probíhat nic jiného, než pouze a jedině dotazování (a případně rychlé zpracování zaznamenaného tiku). Takový mikrokontroler nebude přehlížet ani krátké tiky, nicméně bude výrazně složitější na něm implementovat komplexnější aplikaci.

Přístupem, který je obecně elegantnější a pro mou meteostanici dokonce jediný korektní, je využití přerušení generovaných vzestupnou, případně sestupnou, hranou. Počítač v této chvíli nemusí dělat nic, dokud na daném vstupním pinu nepřijde vzestupná (sestupná) hrana. Ta sama vyvolá přerušení, které vynutí vykonání metody, která bude sloužit k řádnému zpracování informace o detekovaném tiku. Využití přerušení tedy eliminuje všechny nedostatky, které s sebou neslo dotazování.

Pro jazyk Python existuje knihovna pro práci se vstupně výstupními piny (`RPi.GPIO`), která mimo jiné umožňuje generovat a zpracovávat přerušení při detekované vzestupné i sestupné hraně na příslušném pinu. Tuto knihovnu ve své implementaci využiji.

### 3.2.1 Testování

Jako první krok při práci s knihovnou `RPi.GPIO` a jejími funkcemi pro generování přerušení jsem testoval, zdali s její pomocí bude možné registrovat tiky velice krátké a také tiky přicházející ve velmi malých časových rozestupech.

Vytvořil jsem tedy jednoduchý test, ve kterém jsem využil dva navzájem vodivě propojené piny. Jeden výstupní pin uměle generoval tiky a druhý pin detekoval vzestupné hrany. Nakonec program porovnal počet vygenerovaných tiků a počet detekovaných vzestupných hran. Podle shody těchto dvou číselných hodnot test určoval spolehlivost detekce hran, tedy detekce příchozího

| perioda [s] | vygenerováno tiků | detekováno tiků | výskyt chyb |
|---|---|---|---|
| 0.05 | 1000 | 1000 | ne |
| 0.045 | 1000 | 1000 | ne |
| 0.04 | 1000 | 1000 | ne |
| 0.035 | 1000 | 1000 | ne |
| 0.03 | 1000 | 1000 | ne |
| 0.025 | 1000 | 1000 | ne |
| 0.02 | 1000 | 1000 | ne |
| 0.015 | 1000 | 1000 | ne |
| 0.01 | 1000 | 1000 | ne |
| 0.005 | 1000 | 1000 | ne |
| 0.0045 | 1000 | 1000 | ne |
| 0.004 | 1000 | 1000 | ne |
| 0.0035 | 1000 | 1000 | ne |
| 0.003 | 1000 | 1000 | ne |
| 0.0025 | 1000 | 1000 | ne |
| 0.002 | 1000 | 1000 | ne |
| 0.0015 | 1000 | 998 | ano |
| 0.001 | 1000 | 999 | ano |
| 0.0005 | 1000 | 1000 | ne |

**Tabulka 3.2:** Spolehlivost detekce tiků v závislosti na jejich periodě

tiku.

První pozitivní zjištění bylo, že program dokáže bezchybně zpracovávat i velice krátké tiky, včetně těch úplně nejkratších, které jsem byl schopen generovat. Na výstupním pinu jsem jedním příkazem nastavil logickou hodnotu 1 a následujícím příkazem zpět logickou hodnotu 0, bez jakékoliv uměle přidané prodlevy. Teprve po vygenerování tiku jsem generátor na chvíli uspal, další tik tedy generoval až po krátké pauze. Jelikož se dá předpokládat, že tiky přirozeně generované na senzoru budou řádově mnohem delší, než ty umělé softwarové, mohu vyvodit závěr, že zpracování prakticky libovolně dlouhého tiku bude fungovat spolehlivě.

Dále bylo potřeba zjistit, v jakých periodách po sobě mohou tiky přicházet, aby byl systém schopen tyto tiky spolehlivě zpracovávat. Na tuto otázku neexistuje jednoznačná odpověď, protože především záleží na délce procedury, jejíž vykonání je přerušením vyvoláno (angl. interrupt service routine, krátce ISR) - musí se stihnout vykonat celá, než přijde další přerušení. Kvůli této okolnosti je rozumnou praxí psát ISR krátké a výpočetně jednoduché.

Provedl jsem tedy výše popsaný test s nejkratším možným tikem a proměnlivou periodou tiků. V rámci tohoto testu se v ISR pouze inkrementoval čítač zaznamenaných náběžných hran, nic víc. Při této implementaci se chyby (nezaznamenané náběžné hrany) začaly objevovat až při periodě nižší, než 2 milisekundy, pro větší periody se systém choval bezchybně. Výsledky testu jsou zachyceny v tabulce 3.2.

Samozřejmě výsledek tohoto testu nelze zobecnit, vždy bude záležet na

konkrétní implementaci ISR, nicméně se dá předpokládat, že tiky generované srážkoměrem i anemometrem budou mít řádově větší periodu, než jednotky milisekund. Lze tedy proto předpokládat, že ani v tomto směru nebude s detekcí náběžných hran při vývoji a aplikaci meteostanice problém.

## 3.3 Teploměr a vlhkoměr DHT22

Senzor DHT22 poskytuje měřená data již v digitální podobě, pro komunikaci využívá vlastní 1-wire interface. Komunikace přes toto rozhraní probíhá tak, že na dotázání pošle senzor odpověď sestávající z 40 bitů a obsahuje naměřená data z obou senzorů. Prvních 16 bitů představuje relativní vlhkost vzduchu, dalších 16 bitů teplotu a posledních 8 bitů je kontrolní součet. [6]

Pro Raspberry Pi existuje knihovna `Adafruit_DHT`, která komunikaci přes toto rozhraní implementuje. Při vývoji meteostanice jsem tuto knihovnu využil.

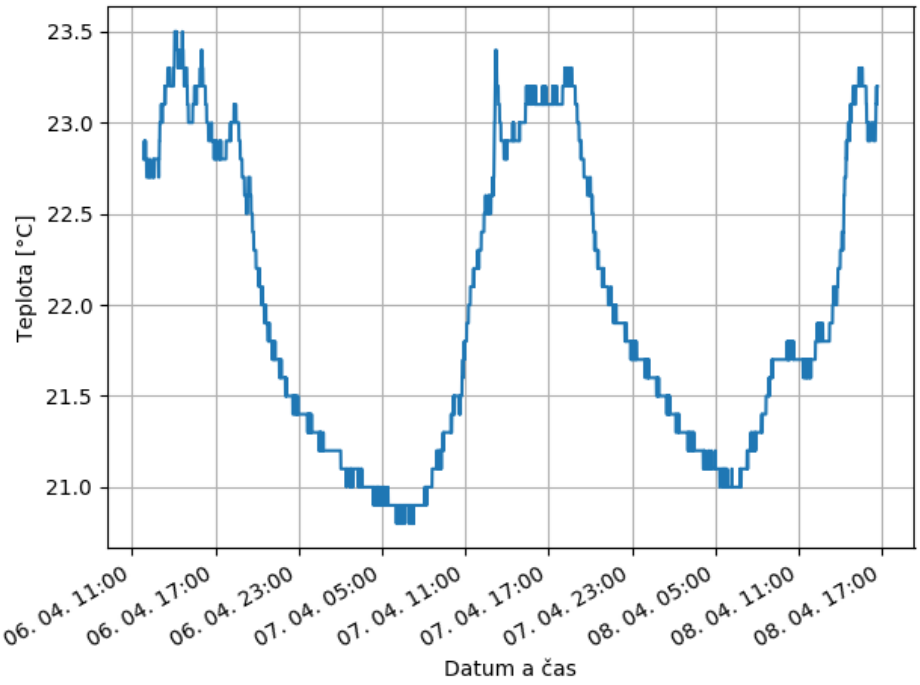
Pro otestování senzoru jsem provedl experiment, při kterém jsem nechal senzor po dobu dvou dní každou minutu změřit teplotu a vlhkost. Experiment jsem vykonával v bytovém prostoru v Praze, kde se typicky teplota nepohybuje mimo interval 20-25°C a vlhkost se běžně drží v intervalu 30-70%. V případě, že naměřené hodnoty budou odpovídat tomuto standardu, bude možné předpokládat, že senzor funguje správně.

Měření jsem provedl v období 6. - 8. dubna 2018. Výsledky měření jsou zaznamenány v grafech 3.2 a 3.3. Průběh teploty odpovídá běžnému dennímu cyklu a obecně se naměřené hodnoty neliší od předpokládaných výsledků. To dokládá, že senzor zřejmě poskytuje validní informace a funguje správně.

## 3.4 Luxmetr TSL2561

Luxmetr TSL2561 využívá komunikačního rozhraní I2C. Jelikož senzor sestává ze dvou fotodiod (jedna je citlivá na infračervené světlo, druhá na obojí, infračervené i viditelné světelné spektrum), má v sobě také integrované dva AD převodníky (pro každou fotodiodu jeden) a data díky tomu může poskytovat již v digitální podobě. Data z jednotlivých diod (respektive převodníků) lze číst samostatně. Přečtená data z AD převodníků nicméně ještě nejsou hodnoty uvedené v jednotkách lux, je to pouze hodnota z registru, který svým číselným rozsahem pokrývá spektrum citlivosti příslušné fotodiody. Na hodnoty vyjádřené v základních jednotkách intenzity osvětlení (lux) je tedy třeba tato čísla ještě přepočítat. Jak čtení, tak zpracování dat (včetně převodu na jednotky lux) z tohoto typu luxmetru je již implementováno ve veřejně dostupné knihovně `tsl2561`, kterou jsem ve své implementaci s výhodou využil.

Ačkoliv v konečném návrhu je k meteostanici připojen pouze jeden luxmetr, při vývoji jsem měl k dispozici tyto luxmetry dva, čehož jsem využil při testování. Díky hardwarově nastavitelné I2C adrese jsem mohl mít oba připojené na stejné I2C sběrnici současně.



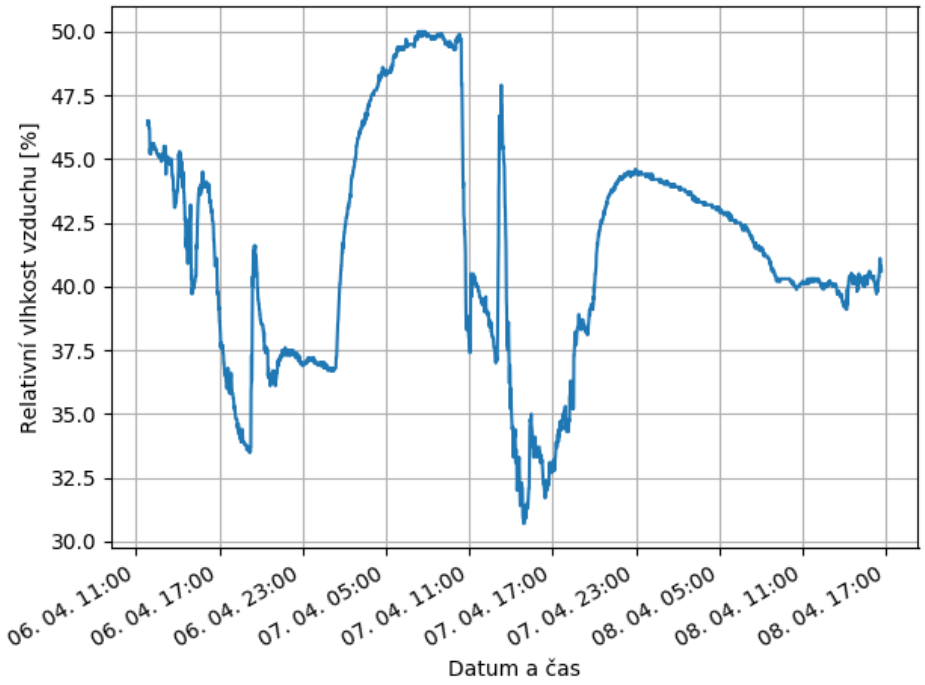

**Obrázek 3.2:** Testování teploměru

Funkčnost obou senzorů jsem otestoval v rámci stejného experimentu, jako senzor DHT22. Měření tedy opět probíhalo v období 6. - 8. dubna 2018 v bytovém prostoru v Praze. Jeden luxmetr jsem přilepil na okno, odstínil od zbytku místnosti a měřil venkovní osvětlení. Druhý luxmetr pak měřil intenzitu osvětlení uvnitř místnosti. V obou případech jsem zaznamenával pouze hodnoty osvětlení ve viditelném spektru.

Jelikož je okno orientováno přibližně severozápadním směrem, dalo se předpokládat, že nejvyšší hodnoty intenzity osvětlení budou naměřeny v podvečer, krátce před západem slunce. Zároveň se kvůli severozápadní orientaci okna, které je navíc částečně zastíněno stromem, daly očekávat relativně nízké měřené hodnoty.

Výsledky měření jsou vykresleny v grafu 3.4. Data odpovídají běžnému dennímu cyklu, navíc potvrzují předpoklad, že největší intenzita osvětlení bude naměřena v podvečer, kvůli zmíněné severozápadní orientaci okna. Zajímavostí naměřených dat je také fakt, že poslední den měření jsou zaznamenané hodnoty značně proměnlivé v průběhu času. Tento jev lze vysvětlit tím, že bylo tento den polojasno, slunce často zacházelo za mraky a posléze opět vycházelo, což způsobilo proměnlivou intenzitu osvětlení.

V tomto experimentu se opět nevyskytly žádné anomálie, které by byly v rozporu s předpokládaným výsledkem. Na podkladě provedeného pokusu lze tedy opět předpokládat, že senzory fungují správně a poskytují validní informace.




**Obrázek 3.3:** Testování vlhkoměru

## 3.5 Barometr BMP280

Barometr BMP280 podporuje komunikaci přes digitální rozhraní I2C i SPI [7]. Ve svém řešení jsem se rozhodl využít rozhraní I2C, díky jeho jednoduššímu zapojení. Komunikace se senzorem přes rozhraní I2C je navíc již implementována ve veřejně dostupném modulu `bme280.py` [3]. Tento modul jsem při implementaci meteostanice využil.

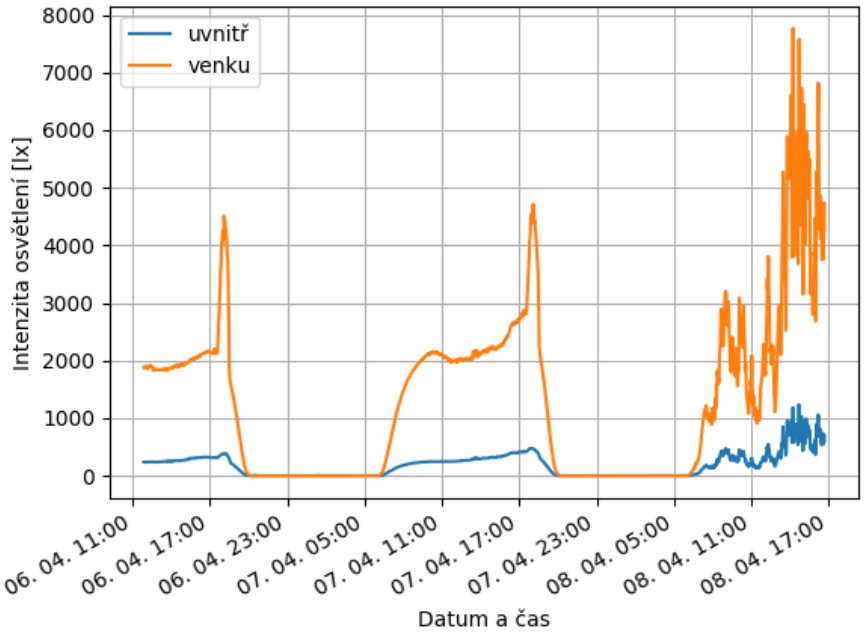
Atmosférický tlak je však veličinou proměnlivou vzhledem k nadmořské výšce. Proto je konvencí měřený tlak přepočítávat ze změřené hodnoty na hodnotu odpovídajícího tlaku vzduchu ve výšce hladiny moře [1]. Přepočet je v praxi naštěstí jednoduchý. Spočívá pouze v přičtení konstanty, která je příslušná k lokalitě a nadmořské výšce, v níž použitý barometr skutečně měří.

### 3.5.1 Testování barometru

Před započetím testu bylo třeba nejprve objevit zmiňovanou konstantu, která se bude k měřenému tlaku přičítat, aby se tlak přepočetl na odpovídající hodnotu ve výšce hladiny moře.

Jednoduchý, avšak účinný způsob, jak tuto konstantu najít, je využití jiného, již správně nastaveného referenčního barometru. Rozumné řešení je tedy vyhledat nejbližší možnou meteostanici k místu nasazení barometru a data, která tato meteostanice poskytuje, využít jako referenční.

Testování jsem prováděl v Praze ve Strašnicích, jako referenci jsem využil




**Obrázek 3.4:** Testování luxmetrů

data dostupná na url http://www.meteostanice-praha.eu/ (stránka navštívena 13. 5. 2018). Důvodem volby tohoto zdroje byl fakt, že poskytuje data z meteostanice, která je umístěna přibližně 1,1km daleko od místa, kde jsem test prováděl já a zároveň se výrazně neliší v nadmořské výšce (můj senzor byl přibližně 220m a referenční zhruba 250m nad mořem). Pro zjištění nadmořské výšky jsem použil podklady dostupné na serveru www.mapy.cz. Atmosférický tlak v místě referenční meteostanice by se tedy měl blížit tlaku v lokalitě, kde jsem test prováděl.

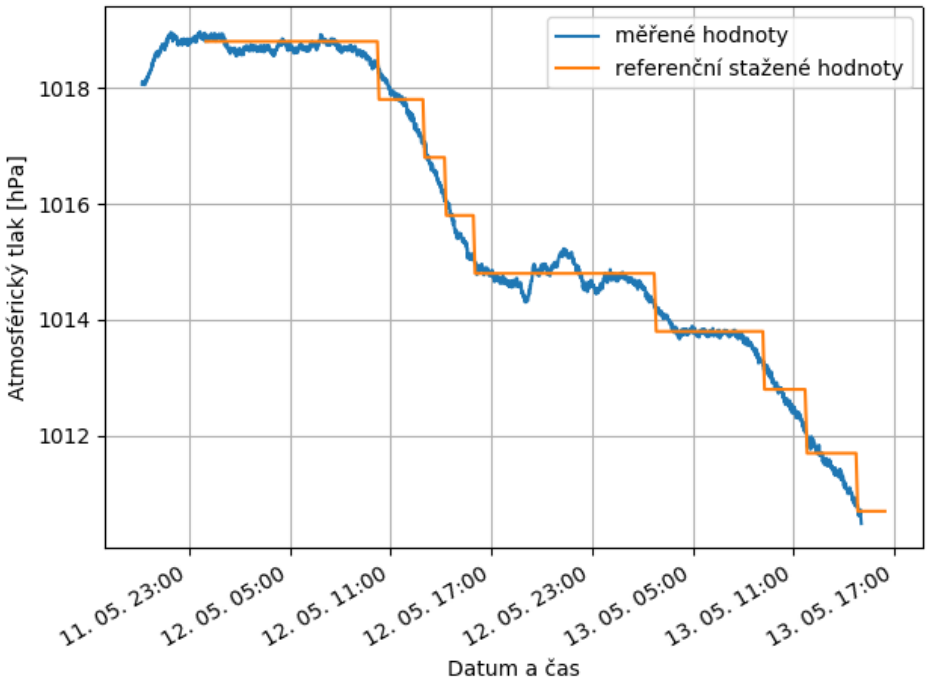
Nejprve jsem tedy zjistil rozdíl mezi hodnotami uvedenými na referenčním serveru a hodnotami, které poskytoval můj barometr. Rozdíl mezi nimi byl 25,3hPa. Tuto hodnotu jsem tedy použil jako konstantu, o kterou jsem při následném testu povýšil své měřené hodnoty.

Velice důležité je však brát na vědomí fakt, že po přemístění barometru do jiné lokality je vhodné provést kalibraci znovu.

Po úvodním nastavení jsem provedl jednoduchý test, abych ověřil správnou funkčnost barometru. Test jsem uskutečnil tak, že jsem nechal senzor přibližně jeden a půl dne měřit a získaná data ukládal. Naměřená data jsem následně porovnal s daty referenčními, získanými z meteostanice, kterou jsem použil jako referenční při úvodním nastavení. V případě správného fungování barometru by si mnou měřená data měla být s daty získanými z referenčního serveru velmi blízká.

Mnou naměřená data a referenční data jsou vizualizována v grafu 3.5.

Jak je z grafu vidět, měřená a referenční data jsou prakticky totožná, což




**Obrázek 3.5:** Test barometru

podporuje hypotézu, že barometr funguje správně.

## 3.6 Srážkoměr

Srážkoměr poskytuje informaci o zachyceném úhrnu kapalných srážek ve formě jednotlivých tiků. Jak bylo již řečeno, funguje na principu dvou vaniček na společné ose. Zachycenými srážkami se vždy plní právě jedna z nich a při zachycení daného objemu srážek se vanička převáží, vyprázdní a během převážení vygeneruje průchodem magnetu kolem jazýčkového spínače tik. Jeden vyvedený vodič je připojen ke vstupnímu pinu s připojeným pull-up rezistorem, jazýčkový spínač případně propojí vstupní pin se zemí, tik je tedy reprezentován poklesem napětí na úroveň *low*, respektive sestupnou hranou. Po převážení vaniček a vygenerování tiku se začne plnit druhá vanička a celý proces se opakuje.

### 3.6.1 Kalibrace srážkoměru

Pro přesnost měření úhrnu srážek je klíčovou informací, jakému množství zachycených srážek odpovídá jeden tik. V dokumentaci ke srážkoměru je uvedeno, že jeden tik odpovídá úhrnu srážek o výšce vodního sloupce 0.2794mm [8].

Správnost tohoto údaje jsem se rozhodl prověřit experimentem. Do srážkoměru jsem naléval postupně pomalu daný objem vody a přitom jsem sledoval,

| | Počet tiků | | |
|---|---|---|---|
| Množství vody [ml] | Pokus č. 1 | Pokus č. 2 | Pokus č. 3 |
| 250 | 115 | 110 | 118 |
| 300 | 142 | 133 | 144 |
| 350 | 168 | 161 | 166 |
| 400 | 192 | 186 | 191 |
| 450 | 217 | 211 | 216 |
| 500 | 240 | 237 | 241 |

**Tabulka 3.3:** Kalibrace srážkoměru

kolik tiků bude tímto množstvím vody vygenerováno. Ze zjištěného počtu tiků při daném množství vody jsem posléze odhadl objem vody odpovídající jednomu tiku. Posledním krokem byl přepočet objemu vody na výšku vodního sloupce, s ohledem na velikost plochy, na níž budou srážky zachycovány, a porovnání tohoto čísla s hodnotou uvedenou v dokumentaci.

Měření jsem provedl celkem třikrát. Vždy jsem do srážkoměru nalil nejprve 250ml vody, následně jsem přiléval po 50ml až do celkového množství 500ml vody. Mezi každým přilitím (tedy při každých dalších 50ml) jsem zaznamenal počet vygenerovaných tiků. Naměřené hodnoty jsou uvedeny v tabulce 3.3.

Z tabulky je vidět, že rozdíl v počtu tiků mezi jednotlivými pokusy neroste úměrně s množstvím vody. Z toho důvodu jsem při odhadu množství vody odpovídající jednomu tiku bral v úvahu záznamy s největším množstvím vody, tedy s 500ml (u nich by měla být chyba připadající na jeden tik nejmenší).

Spočítal jsem tedy průměrné množství vygenerovaných tiků 500ml vody $n$.

$$n = \frac{240 + 237 + 241}{3} = 239,\overline{3}$$

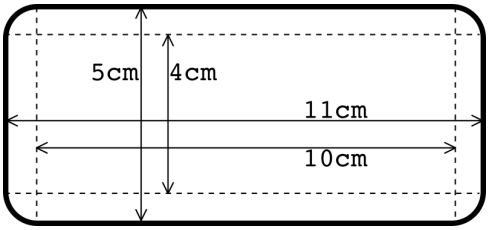
Z toho lze snadno spočítat objem vody odpovídající jednomu tiku $V$. Použitý objem vody (500ml) vydělím průměrným počtem tiků, které toto množství vody vygeneruje ($n$).

$$V = \frac{500}{n} = \frac{500}{239,\overline{3}} = 2,0891\text{ml}$$

V posledním kroku výpočtu bylo třeba přepočítat objem vody, který vygeneruje jeden tik, na odpovídající výšku vodního sloupce, ve kterém se úhrn srážek typicky zaznamenává. Objem vody lze při tomto přepočtu vyjádřit jako součin velikosti plochy, na níž srážkoměr srážky zachycuje, vynásobený hledanou výškou vodního sloupce.

Při pohledu shora má srážkoměr tvar obdélníka se zaoblenými rohy, přesné rozměry jsou naznačeny ve schématu 3.6. Plochu $S$ tohoto útvaru lze spočítat například rovnicí

$$S = 5 \cdot 10 + 2\left(0,5 \cdot 4 + 2 \cdot \frac{\pi \cdot 0,5^2}{4}\right) = 54,7854\text{cm}^2$$




**Obrázek 3.6:** Srážkoměr - pohled shora

Hledaná výška vodního sloupce $h$ odpovídající jednomu tiku se pak spočítá jako objem vody dělený plochou, na níž byl zachycen (v rovnici využijeme toho, že $1\text{cm}^3 = 1\text{ml}$).

$$h = \frac{V}{S} = \frac{2,0891\text{cm}^3}{54,7854\text{cm}^2} = 0,038133\text{cm} = 0,38133\text{mm}$$

Tento výsledek je poměrně zajímavý, protože se relativně výrazně liší od informace uváděné v dokumentaci, že jednomu tiku odpovídá 0.2794mm srážek.

Není jednoduché odhadnout, v důsledku čeho tento rozdíl vznikl. Jedna možnost je taková, že výrobce měření prováděl nepřesně. Druhou variantou je možnost, že se jednotlivé vyrobené kusy tohoto srážkoměru mohou v tomto parametru výrazně lišit. Tuto teorii podporuje fakt, že je srážkoměr poměrně citlivý (pro převážení vaničky stačí přibližně 2ml vody). Pokud je mezi dvěma vyrobenými kusy nepatrný mechanický rozdíl, pro který u jednoho srážkoměru stačí trochu méně či více vody na převážení vaničky a vygenerování tiku, může mít tato skutečnost významný vliv na kalibraci daného srážkoměru. Tuto hypotézu však nemohu ani potvrdit, ani vyvrátit, k tomu by bylo potřeba mít k dispozici víc kusů tohoto srážkoměru a otestovat kalibraci každého jednotlivého kusu zvlášť. Já ovšem disponuji pouze jedním kusem, proto takový test nemohu provést.

Nejdůležitějším závěrem, který si z tohoto experimentu však odnesu, je zjištěný převodní vztah mezi počtem tiků a výškou vodního sloupce měřených srážek. Při implementaci své meteostanice použiji rozhodně hodnotu odhadnutou během tohoto experimentu, nikoliv nepřesnou hodnotu uvedenou v dokumentaci.

## 3.7 Anemometr

Anemometr reprezentuje měřená data, obdobně jako srážkoměr, pomocí tiků. Senzor sestává ze statické a pohyblivé části.

Větrné počasí způsobí otáčení pohyblivé části a během jedné otáčky anemometru způsobí průchody magnetu kolem jazýčkového spínače dva tiky, které lze na vstupním pinu Raspberry Pi zpracovat. Obdobně jako v případě srážkoměru je vstupní pin na Raspberry Pi připojen na pull-up rezistor a tik je představován krátkým propojením pinu se zemí.

Při softwarovém zpracování tiků vycházím z předpokladu, že při běžných provozních podmínkách bude rychlost větru přímo úměrná rychlosti otáčení anemometru. Tento předpoklad je podpořen převodním vztahem uvedeným v dokumentaci k senzoru [8]. Pro určení úhlové rychlosti, tedy množství otáček za jednotku času, je klíčové u zachycených tiků uchovat informaci, kdy byly vygenerovány.

Ve své implementaci k tomuto účelu využívám frontu (o dané maximální velikosti), která pro každý zachycený tik uloží jeho časové razítko. Dále jsem vytvořil metodu, která spočítá průměrnou úhlovou rychlost anemometru za danou jednotku času, respektive za posledních $n$ sekund. Princip fungování této metody spočívá v iterování přes frontu a zjištění, kolik tiků bylo vygenerováno ve zkoumaném časovém intervalu. Úhlovou rychlost anemometru následně metoda spočítá jako podíl počtu vygenerovaných tiků a délky zkoumaného časového intervalu.

Nakonec je třeba úhlovou rychlost anemometru přepočítat na skutečnou rychlost větru. Díky předpokladu, že při běžných provozních podmínkách je rychlost otáčení anemometru přímo úměrná rychlosti větru, stačilo k tomuto přepočtu nalézt pouze vhodnou lineární funkci, pomocí které se přepočet provede.

### 3.7.1 Kalibrace anemometru

Vztah pro přepočet úhlové rychlosti anemometru na rychlost větru je uveden v dokumentaci k senzoru, dle dokumentace odpovídá 1 tik (tedy půl otáčky) za 1 sekundu rychlosti větru 2,4km/h. Po převedení tedy 1 otáčka za sekundu odpovídá rychlosti větru $0,\overline{3}$m/s. Pokud bych tedy vycházel z dat uvedených v dokumentaci, mohl bych převod úhlové rychlosti otáčení anemometru $\omega$ na rychlost větru $v$ formalizovat funkcí

$$v = 0,\overline{3}\omega$$

Informace uvedené v dokumentaci jsem opět ověřil vlastním měřením. Pro testování jsem měl k dispozici domácí ventilátor se třemi různými rychlostmi, a dále druhý anemometr, který byl již zkalibrovaný, tudíž jsem jej mohl využít jako referenční.

Pro každou ze tří nastavitelných rychlostí na ventilátoru jsem v různých vzdálenostech od něj měřil referenčním anemometrem skutečnou rychlost větru a následně i úhlovou rychlost otáčení mého anemometru. Na základě takto získaných dat jsem následně odhadl převodní funkci z rychlosti otáčení mého anemometru na skutečnou rychlost větru. Data, která jsem naměřil, jsou uvedena v tabulce 3.4.

Data z tohoto pokusu lze vnímat jako body ve dvourozměrném prostoru, jejichž souřadnice jsou uspořádané dvojice [rychlost otáčení anemometru, skutečná rychlost větru]. Proložením těchto bodů přímkou jsem odhadl funkci pro přepočet úhlové rychlosti anemometru na rychlost větru.

Pro proložení bodů přímkou jsem použil knihovní funkci programovacího prostředí Matlab `fit`. Nalezená přímka má směrnici 0,6589 a je posunuta

| | Rychlostní stupeň ventilátoru | | | | | |
|---|---|---|---|---|---|---|
| | 1 | | 2 | | 3 | |
| vzdálenost | reference | otáčky | reference | otáčky | reference | otáčky |
| 20 | 2,2 | 2,7 | 4 | 2,9 | 4,2 | 3,1 |
| 40 | 2,2 | 2,6 | 2 | 2,7 | 3 | 2,9 |
| 60 | 2,3 | 2,2 | 2 | 2,4 | 2,8 | 2,5 |
| 80 | 2 | 1,6 | 2 | 1,5 | 2,3 | 1,8 |
| 100 | 1,9 | 1 | 2 | 1,2 | 2,2 | 1,3 |
| 120 | 1,7 | 0,7 | 1,9 | 0,8 | 2,1 | 1,1 |
| 140 | 1,6 | 0,4 | 1,8 | 0,8 | 1,9 | 0,8 |
| 160 | 1,5 | 0,4 | 1,6 | 0,6 | 1,7 | 0,6 |
| 180 | 1,3 | 0,1 | 1,5 | 0,5 | 1,6 | 0,4 |
| 200 | 1,2 | 0 | 1,3 | 0,4 | 1,6 | 0,4 |
| 220 | 1 | 0 | 1,2 | 0,1 | 1,5 | 0,6 |
| 240 | 0,9 | 0 | 1,2 | 0 | 1,4 | 0,4 |
| 260 | 1 | 0 | 1,2 | 0 | 1,4 | 0,5 |
| 280 | 0,9 | 0 | 1,1 | 0 | 1,2 | 0,4 |
| 300 | 0 | 0 | 1,1 | 0 | 1 | 0,3 |

vzdálenost - vzdálenost od ventilátoru [cm],
reference - skutečná rychlost větru měřená referenčním anemometrem [m/s],
otáčky - rychlost otáčení anemometru, který kalibruji [otáčky/s]

**Tabulka 3.4:** Kalibrace anemometru

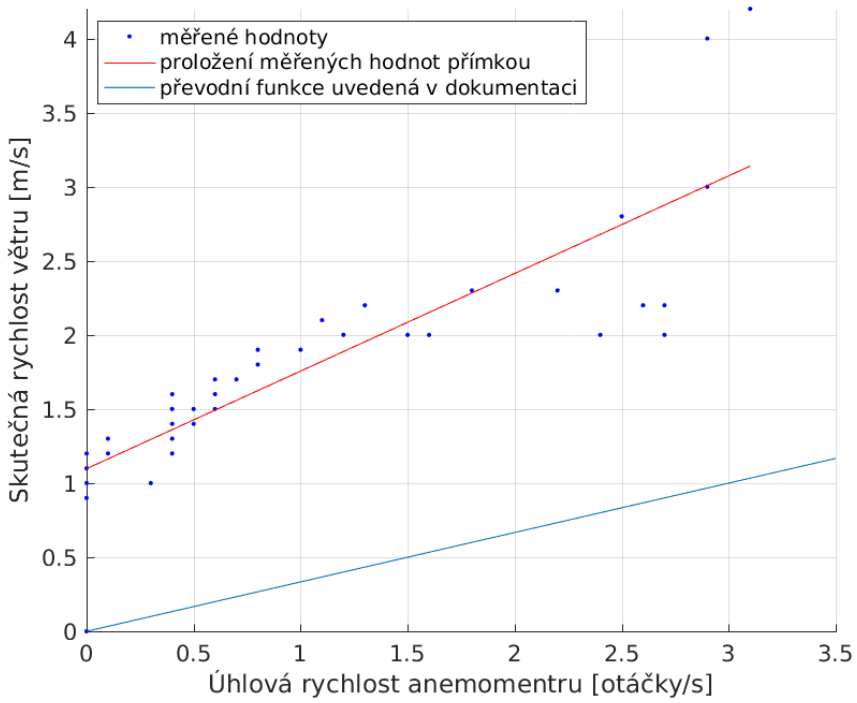
o konstantu 1,097. Vizualizace naměřených bodů a jejich proložení je znázorněno v grafu 3.7.

Při odhadu funkce pro převod rychlosti otáčení anemometru na rychlost větru musím ještě uvážit fakt, že velmi slabý vítr nemusí anemometr kvůli svým mechanickým vlastnostem vůbec zaznamenat. Naměřené hodnoty naznačují, že při rychlosti větru kolem 1 m/s a méně se anemometr vůbec neroztočí, pro tento rozsah se tedy jeví jako netečný. V případě přímky, která prokládá naměřené body, je toto netečné pásmo reprezentováno posunem o konstantu. Existenci tohoto netečného pásma je tedy třeba brát v úvahu při formalizaci hledané převodní funkce a pokud se anemometr netočí vůbec, je vhodnější tento stav považovat za bezvětří.

Hledanou funkci pro převod úhlové rychlosti otáčení anemometru $\omega$ na rychlost větru $v$ jsem tedy formalizoval následujícím způsobem:

$$v = \begin{cases} 0,6589\omega + 1,097 & \text{pro } \omega > 0, \\ 0 & \text{jinak} \end{cases}$$

Zajímavým závěrem tohoto měření je fakt, že mnou odhadnutý převodní vztah je výrazně odlišný od toho, který je uvedený v dokumentaci. Rozdíl je dobře viditelný v grafu 3.7. Na vztahu v dokumentaci je navíc podezřelé, že z něj plynoucí lineární funkce není posunutá o konstantu, tedy neuvažuje netečné pásmo anemometru, které mnou odhadnutá funkce reflektuje. Z toho




**Obrázek 3.7:** Kalibrace anemometru

důvodu se převodní vztah v dokumentaci jeví jako výrazněji zjednodušený, nežli vztah mnou odhadnutý.

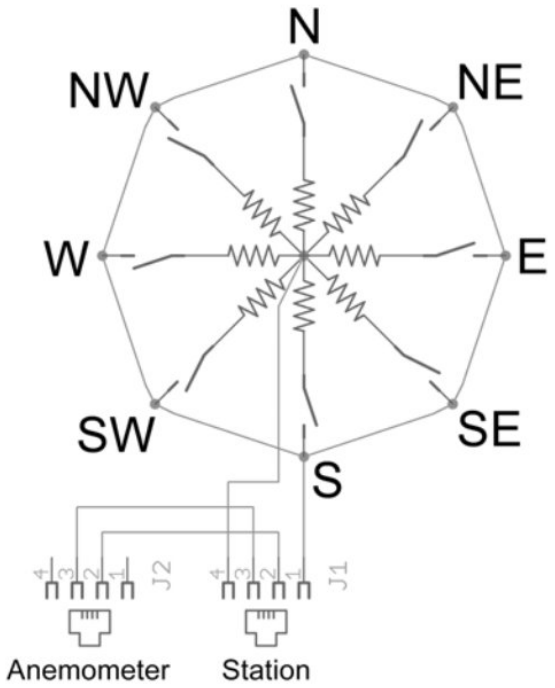
Vzhledem k výsledkům mého pokusu jsem tedy při implementaci nevyužil převodní vztah uvedený v dokumentaci, místo něj jsem použil funkci odhadnutou na základě mnou naměřených dat.

## 3.8 Korouhvička

Korouhvička je jediný z použitých senzorů, který poskytuje data v analogové podobě. V sobě má zabudovaných osm rezistorů o různě velkém odporu. Podle toho, jak je natočena, tak do svého vnitřního obvodu připojí vždy jeden nebo paralelně dva sousední rezistory. Díky tomu může odpor, který vnitřní obvod korouhvičky představuje, nabývat šestnácti různých hodnot, přičemž každá hodnota představuje jeden daný směr. Schéma vnitřního obvodu korouhvičky je naznačeno na obrázku 3.8.

Jak je vidět ve schématu 3.1, korouhvička je zapojena jedním vyvedeným vodičem přímo na zem, druhým přes rezistor o odporu $10k\Omega$ k napájecímu napětí $3,3V$. Dohromady tyto součástky tvoří napěťový dělič. Jelikož je odpor na korouhvičce proměnlivý, je na ní proměnlivý i úbytek napětí. Ten lze měřit a podle jeho hodnoty lze určit, jakým směrem je korouhvička natočena.

Jelikož velikosti vnitřních odporů v korouhvičce jsou známé, lze snadno spočítat, jaký úbytek napětí způsobí daný odpor na korouhvičce. Výpočet se




**Obrázek 3.8:** Vnitřní obvod korouhvičky, zdroj: [5]

provede následujícím způsobem.

Z druhého Kirchhoffova zákona víme, že součet úbytků napětí na uzavřené smyčce musí být roven nule. V našem případě to znamená, že součet úbytků napětí na korouhvičce $U_k$ a na rezistoru $U_r$ musí být roven napájecímu napětí $U_{cc}$.

$$U_{cc} = U_k + U_r$$

Z tohoto vztahu lze vyjádřit úbytek napětí na korouhvičce $U_k$.

$$U_k = U_{cc} - U_r$$

Z Ohmova zákona vyplývá vztah $U = RI$, díky kterému můžeme předchozí vztah upravit

$$U_k = U_{cc} - R_r I$$

Pro vypočtení proudu v obvodu (opět aplikací Ohmova zákona) je třeba znát kromě napájecího napětí $U_{cc}$ ještě celkový odpor $R$ na takovém obvodu. Jelikož jsou korouhvička a rezistor zapojeny sériově, stačí jejich odpory pouze sečíst.

$$I = \frac{U_{cc}}{R} = \frac{U_{cc}}{R_r + R_k}$$

Nyní již stačí dosadit a získáme vzorec pro výpočet úbytku napětí na korouhvičce.

$$U_k = U_{cc} - \frac{R_r U_{cc}}{R_r + R_k} = U_{cc}\left(1 - \frac{R_r}{R_r + R_k}\right) = 3,3 \cdot \left(1 - \frac{10000}{10000 + R_k}\right)$$

Tím je vyřešen problém s převodem úhlu natočení korouhvičky na odpovídající měřitelný úbytek napětí. Druhým krokem, který je třeba vyřešit, je implementace samotného čtení napěťové úrovně.

Raspberry Pi nemá zabudovaný AD převodník, použil jsem proto převodník externí. Zvolil jsem převodník ADS1115/ADS1105, který podporuje komunikaci přes rozhraní I2C. Při čtení z tohoto převodníku je třeba zvolit rozsah, v němž se může pohybovat hodnota měřeného napětí na vstupu. S ohledem na hodnoty, které může korouhvička při mém zapojení poskytovat, se z nabízených možností nejvíce hodí rozsah $\pm 4.096V$.

Převodník naměřené analogové hodnoty reprezentuje 16-bitovým číslem kódovaném ve dvojkovém doplňku. Kladná napětí v rámci zvolené škály tedy reprezentuje hexadecimálními čísly 0 až 0x7FFF, tedy dekadickými 0 až 32767 [4]. Při mnou zvolené škále $\pm 4.096V$ to tedy znamená, že $0V$ reprezentuje převodník kódem 0, napětí $4,096V$ pak reprezentuje kódem 32767 (kód roste lineárně s napětím). Přepočet napěťové úrovně $U$ na příslušný kód $x$ lze tedy vyjádřit vztahem.

$$x = \frac{32768}{4,096} \cdot U = 8000 \cdot U$$

S aplikací uvedených výpočtů jsem tedy zjistil hodnoty, s jejichž znalostí již lze bez potíží implementovat program pro zpracování dat z korouhvičky a určení směru jejího natočení. Tato data jsou uvedena v tabulce 3.5.

Tato čísla jsou samozřejmě pouze teoreticky spočtená, u reálné aplikace je třeba počítat s možnými nepřesnostmi, například v případě velikosti odporu u použitých rezistorů. Proto při softwarovém zpracování nelze očekávat přesné číslo, které by odpovídalo některé ze spočtených hodnot uvedených v tabulce. Směr korouhvičky určuji pouze podle toho, kterému ze spočtených čísel je přečtená hodnota nejblíže.

Dále je nezbytné brát na vědomí, že při reálném nasazení za větrného počasí se korouhvička typicky výrazně komíhá ze strany na stranu. Přečíst ji tedy jen jednou a tuto hodnotu považovat za skutečný směr větru by bylo naivní a nepřesné. Výrazně přesnějším přístupem je změřit aktuální natočení vícekrát a z měřených hodnot vypočítat průměr.

Ve svém programu jsem tedy vytvořil samostatné vlákno, které v pravidelných intervalech čte aktuální natočení korouhvičky, které ukládá do fronty. Fronta má opět danou maximální velikost. Když je dosaženo maximální kapacity fronty, při uložení nového záznamu se nejstarší smaže. Při dotazu na aktuální směr větru vypočítám průměrnou hodnotu ze záznamů uložených v této frontě.

| Směr [°] | Vnitřní odpor [$k\Omega$] | Úbytek napětí [$V$] | Kód na ADC |
|---|---|---|---|
| 0 | 33 | 2.5325581395 | 20260.4651162791 |
| 22.5 | 6.57 | 1.3084490042 | 10467.592033796 |
| 45 | 8.2 | 1.4868131868 | 11894.5054945055 |
| 67.5 | 0.891 | 0.2699752089 | 2159.8016711046 |
| 90 | 1 | 0.3 | 2400 |
| 112.5 | 0.688 | 0.2124251497 | 1699.4011976048 |
| 135 | 2.2 | 0.5950819672 | 4760.6557377049 |
| 157.5 | 1.41 | 0.4078001753 | 3262.4014022787 |
| 180 | 3.9 | 0.9258992806 | 7407.1942446043 |
| 202.5 | 3.14 | 0.7885844749 | 6308.6757990868 |
| 225 | 16 | 2.0307692308 | 16246.1538461538 |
| 247.5 | 14.12 | 1.931840796 | 15454.7263681592 |
| 270 | 120 | 3.0461538462 | 24369.2307692308 |
| 292.5 | 42.12 | 2.6668457406 | 21334.7659247889 |
| 315 | 64.9 | 2.8594125501 | 22875.300400534 |
| 337.5 | 21.88 | 2.264868256 | 18118.9460476788 |

**Tabulka 3.5:** Korouhvička - důležité hodnoty

Směr větru je však reprezentován ve stupních, což je cyklická škála hodnot. To výpočet průměru lehce komplikuje, protože nestačí spočítat pouze aritmetický průměr. Výpočet jsem prováděl následujícím postupem.

Na počátku mám množinu $M$ s $n$ zaznamenanými úhly natočení korouhvičky $\alpha_i$

$$M = \{\alpha_1, \ldots, \alpha_n\}$$

Na každý úhel $\alpha_i$ posléze pohlížím jako na jednotkový vektor $\vec{v_i} = (x_i, y_i)$ o příslušném natočení. Souřadnice $(x_i, y_i)$ každého vektoru $\vec{v_i}$ určím pomocí goniometrických funkcí.

$$x_i = cos(\alpha_i)$$
$$y_i = sin(\alpha_i)$$

Následně spočítám ze všech vektorů $\vec{v_i}$ jeden průměrný vektor $\vec{v} = (x, y)$, jehož jednotlivé souřadnice jsou aritmetickými průměry příslušných souřadnic vektorů $\vec{v_i}$.

$$x = \frac{1}{n}\sum_{i=1}^{n} x_i$$
$$y = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Úhel natočení $\alpha$ průměrného vektoru $\vec{v} = (x, y)$ již odpovídá hledané průměrné hodnotě natočení korouhvičky. Hledaný úhel $\alpha$ určím opět pomocí goniometrických funkcí. Převodní funkce je následující.

$$\alpha = \begin{cases} 0° & \text{pro } x = 0 \text{ a } y > 0 \\ 180° & \text{pro } x = 0 \text{ a } y < 0 \\ \text{arctg}\left(\frac{y}{x}\right) & \text{pro } x > 0 \\ \text{arctg}\left(\frac{y}{x}\right) + 180° & \text{pro } x < 0 \end{cases}$$

Při výpočtu je třeba brát ohled na dvě úskalí funkce arcus tangens. První je, že arcus tangens má periodu 180°, zatímco korouhvička má rozsah 360°. Druhým je fakt, že pro úhly 0° a 180° není arcus tangens definován. Na oba tyto speciální případy beru při výpočtu zřetel. Další speciální případ nastane, když převodní funkce vrátí záporný úhel. To se může stát, jelikož arcus tangens má obor hodnot ±90°. V takovém případě stačí k výsledku přičíst 360° a výsledek se posune do požadovaného intervalu $\langle 0°, 360° \rangle$.

Posledním speciálním případem, který může nastat, je situace, že průměrný vektor bude nulový. Pro takový případ nelze uvedeným způsobem směr větru určit. Vzhledem k tomu, že průměr vypočítávám z velkého množství dat, je však vysoce pravděpodobné, že nulový průměrný vektor se při výpočtu bude objevovat jen velmi zřídka. Výjimečný výskyt takové anomálie při pravidelném sběru dat tedy nebude vadit.

# Kapitola 4

# Softwarové řešení

Jak bylo již dříve zmíněno, softwarové řešení meteostanice sestává z několika modulů. Prvním je modul sloužící ke komunikaci se senzory, který umí načíst a zpracovat naměřená data. K ukládání dat slouží databáze. Dále bylo třeba rozmyslet a vytvořit vhodné rozhraní, které umožní přístup k měřeným datům. Posledním modulem je jednoduchý generátor předpovědi počasí. Schéma softwarového návrhu je vizualizováno na obrázku 2.3. V této kapitole rozeberu podrobněji jednotlivé složky softwarového řešení meteostanice.

## 4.1 Databáze

Pro ukládání měřených dat je vhodné vytvořit databázi. Navrhl jsem proto jednoduchou databázi o jedné tabulce, která k ukládání meteorologických dat poslouží. Návrh této tabulky je zobrazen zde: 4.1.

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| record_id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| date_time | datetime | NO | | NULL | |
| temperature | float | YES | | NULL | |
| humidity | float unsigned | YES | | NULL | |
| luminosity | double unsigned | YES | | NULL | |
| air_pressure | double unsigned | YES | | NULL | |
| precipitation | int(10) unsigned | YES | | NULL | |
| wind_speed | float unsigned | YES | | NULL | |
| wind_direction | float unsigned | YES | | NULL | |

**Tabulka 4.1:** Návrh databáze

U většiny veličin stačí ukládat aktuálně změřenou hodnotu. Jediným speciálním případem je úhrn srážek, který je třeba vztáhnout na jednotku času. Tuto komplikaci jsem vyřešil tak, že do databáze ukládám vždy celkový úhrn srážek naměřený za dobu, která uplynula od zápisu posledního předchozího záznamu v databázi. Tento přístup hodnotím jako výhodný proto, že lze následně z databáze jednoduše získat úhrn srážek za libovolný zvolený časový úsek. Stačí vybrat řádky ve zvoleném časovém intervalu a spočítat součet úhrnů srážek ze všech vybraných řádků.

Co se konkrétní použité technologie týče, zvolil jsme databázový server MySQL, který jsem nainstaloval přímo na RaspberryPi. K němu jsem implementoval modul `db_manager.py`, který umožní přístup do databáze přímo z programu.

Ve své implementaci jsem samozřejmě počítal s tím, že se perspektivně může databázový server změnit. V případě, že by k takové změně došlo, stačí v modulu `db_manager.py` upravit některý z parametrů `host, database, user, password` a modul se bez potíží připojí k jiné databázi.

Velkou výhodou využití MySQL databázového serveru je fakt, že se k němu lze snadno připojit vzdáleně. Vezmeme-li v úvahu hlavní účel této práce, tedy poskytování dat pro provoz autonomních bezpilotních prostředků, lze do budoucna předpokládat, že budou tyto prostředky potřebovat přistupovat k datům automaticky. U MySQL serveru existuje mnoho prostředků, které umožní připojení k databázi z různých programovacích jazyků.

Takovéto řešení tedy kromě spolehlivé archivace dat zajistí také velice snadno použitelné rozhraní, přes které bude možné k datům přistupovat. Proto hodnotím využití MySQL serveru jako výhodné.

## 4.2 Webový server

Další užitečnou funkcionalitou softwarového vybavení meteostanice je webový server. Jelikož bude meteostanice při svém provozu připojena k internetu, je toto řešení, vedle přímého přístupu do databáze, druhou možností, jak přistupovat k uchovaným datům.

S výhodou jsem také opět využil možnosti programování v jazyce Python, který je pro tvorbu webového serveru vhodným nástrojem, především díky existenci mnoha knihoven a frameworků, které implementaci jednoduchého serveru značně usnadní. Jako základ svého serveru jsem konkrétně zvolil webový framework *Flask*, který poskytuje jednoduché API pro implementaci požadované funkcionality serveru.

Jelikož pro strojový přístup k datům je možné využít přímé napojení do databáze, při implementaci serveru jsem se především zaměřil na vizualizaci dat pro lidského uživatele. Vytvořil jsem tedy webovou stránku, kde je uveden aktuální stav počasí, grafy zobrazující třídenní historii vybraných meteorologických veličin a krátkodobou předpověď vyhodnocenou na základě lokálně měřených dat.

V případě budoucí potřeby by však díky použitému frameworku *Flask* rozšíření serveru o libovolnou funkcionalitu nebyl problém. Jádro celého serveru je totiž reprezentováno modulem, jehož metody jsou přiřazeny adresám URL, které se navzájem liší v parametru *path*. Stačilo by tedy implementovat požadovanou funkcionalitu v nové metodě a této metodě přiřadit nový, ještě neobsazený parametr *path* a pomocí něj definovat URL, na které lze vyžádat vykonání této metody.

### 4.2.1 Grafické webové rozhraní

Pro tvorbu webové aplikace s grafickým uživatelským rozhraním umožňuje *Flask* využití parametrizovatelných šablon, ze kterých se po dotázání dynamicky vytvoří požadovaná webová stránka, která se odešle zpět v HTTP odpovědi.

Základem tohoto konstruktu jsou dva oddělené moduly. Jedním je samotný server, zajišťující zpracování a zodpovězení HTTP požadavků, dynamické generování webové stránky a případnou další logiku aplikace (např. komunikaci s databází či jinými moduly v rámci meteostanice). Druhým samostatným modulem je parametrizovatelná šablona definující vzhled a případnou funkcionalitu konkrétní webové stránky.

Šablona je v mém případě obyčejný soubor s webovou stránkou popsanou v jazyce HTML, doplněnou o funkcionalitu pomocí jazyka Javascript. V šabloně jsou však navíc ještě speciální značky indikující proměnlivý parametr (například konkrétní meteorologická data určená k vypsání/vykreslení grafem). V momentě, kdy server obdrží dotaz na tuto webovou stránku, dynamicky vygeneruje části stránek, kterými nahradí parametry v šabloně. Tuto vygenerovanou stránku nakonec server zašle v HTTP odpovědi.

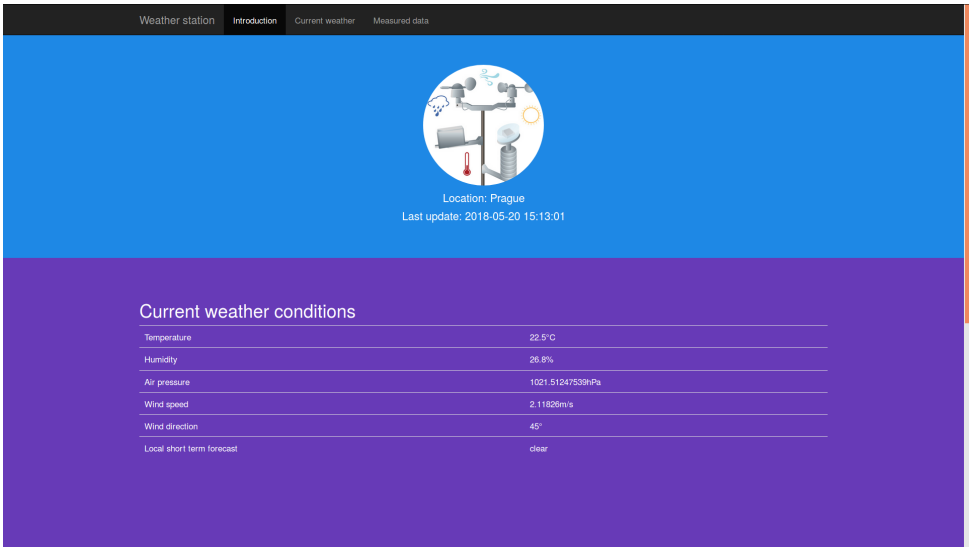
Co se struktury webu týče, všechny informace jsou shrnuty na jedné stránce, ta je strukturována do tří částí umístěných nad sebou. První je úvod poskytující informace, kdy bylo provedeno poslední měření a kde se meteostanice vyskytuje. Druhá část vypisuje zaznamenaná data z posledního měření, včetně jednoduché předpovědi určené na základě lokálně změřených hodnot. V poslední části jsou grafy vizualizující třídenní historii vybraných meteorologických veličin.

V současné době je při návrhu vzhledu samozřejmě nezbytné brát na zřetel velkou rozmanitost zařízení, na nichž může být stránka zobrazována, web je tedy třeba tvořit responzivně, aby byla stránka dobře čitelná na různých druzích zařízení. Pro splnění tohoto požadavku jsem pro formátování použil volně dostupné šablony a knihovny s kaskádovými styly *Bootstrap* a pro responzivní vykreslování grafů pak knihovnu *Chart.js*. Využití těchto dvou technologií zajistí responzivní vzhled webu a dobrou čitelnost na různých zařízeních.

Ukázky webové stránky jsou k nahlédnutí na snímcích obrazovky 4.1 a 4.2.

## 4.3 Předpovídání počasí

K předpovídání počasí lze přistoupit dvojím způsobem. První je výpočet na základě lokálně měřených dat, především atmosférického tlaku vzduchu. Snížený tlak, typicky méně než 1000hPa predikuje velkou oblačnost a deštivé počasí, zvýšený tlak (více než 1020hPa) naopak předpovídá jasno bez oblačnosti. Tento přístup využívají často domácí meteostanice, i já jsem ve své meteostanici takové vyhodnocení předpovědi implementoval. Uživateli ji webový server na mé meteostanici poskytuje dvojím způsobem. Prvním je výpis na webové stránce v tabulce informující o aktuálním stavu počasí. Druhou možností je získat pouze předpověď seriali-




**Obrázek 4.1:** Screenshot webové stránky - úvod a aktuální informace

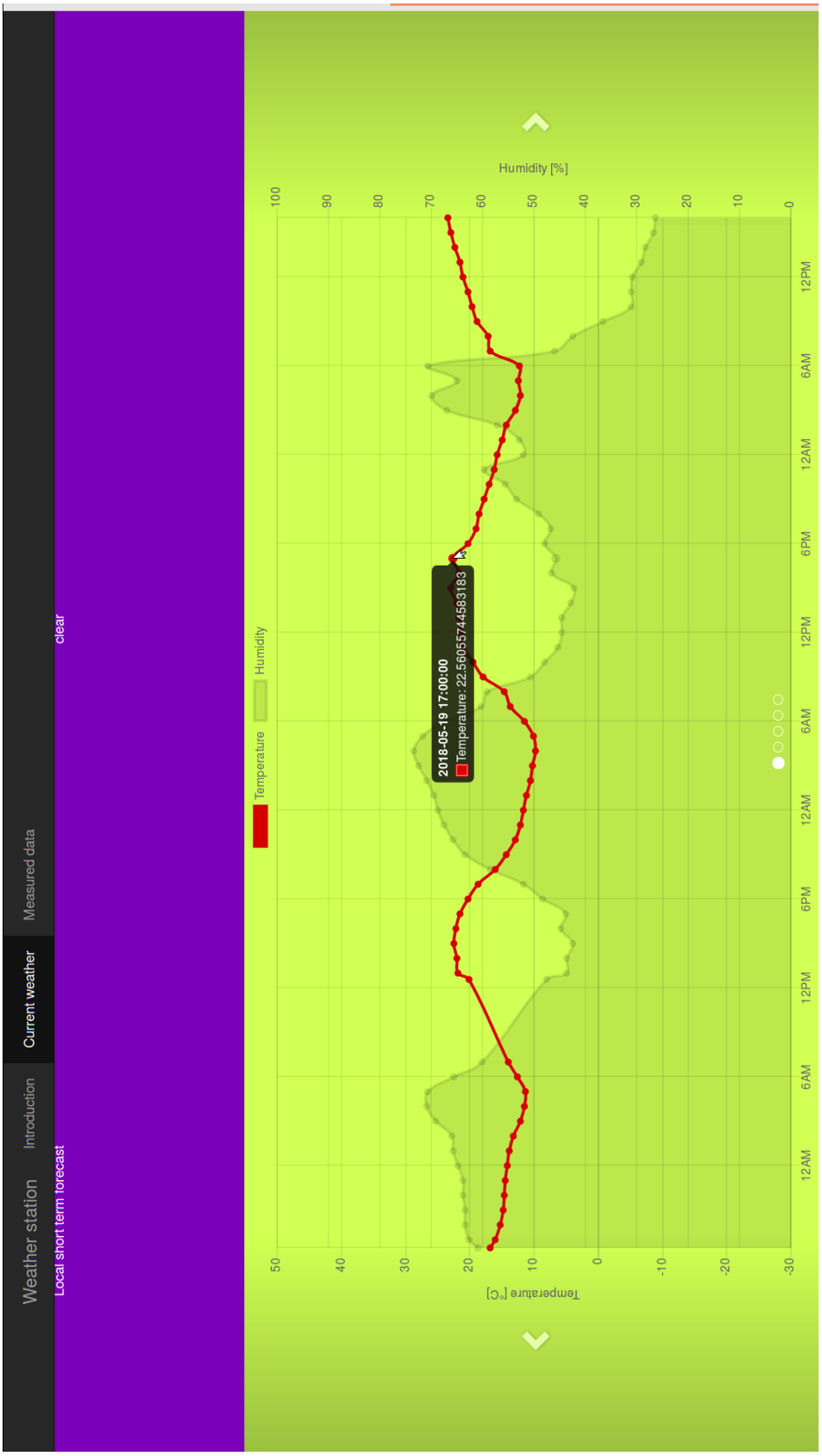
zovanou ve formátu JSON. Serializovaná předpověď je k dispozici na url `<ip-adreda>:<port>/json/forecast`. Po zaslání HTTP požadavku na tuto url vrátí server odpověď ve tvaru `{"forecast": <předpověď>}`. Předpověď může nabývat tří hodnot, `"clear"`, `"partly cloudy"` a `"rain"`.

Tento princip generování předpovědi má však dvě zásadní nevýhody. První nevýhodou je nemožnost tvorby dlouhodobé předpovědi, druhou je poměrně malá přesnost takové předpovědi.

Kvalitní předpovědi počasí se v praxi tvoří na základě velkého množství dat získaných z různých zdrojů a jsou vypočítávány pomocí složitých a výpočetně náročných postupů. Takovou předpověď by nebylo rozumné se pokoušet počítat samostatně. V případě potřeby je totiž možné takovou předpověď stáhnout z veřejně dostupných zdrojů, a to jak v grafické podobě (zobrazit jako webovou stránku), tak serializovanou, vhodnou pro strojové zpracování. Vlastní výpočet kvalitní předpovědi by tedy bylo značné plýtvání výpočetním výkonem s nejistým výsledkem.

Jedním serverem, který takové předpovědi poskytuje je například server *www.openweathermap.com*. Kromě grafického zobrazení předpovědi z něj lze získat předpověď serializovanou ve formátech JSON, XML nebo HTML, které jsou vhodné pro strojové zpracování.

Tento konkrétní server poskytuje možnost získávat serializovanou předpověď pouze registrovaným uživatelům (při registraci uživatel získá klíč zvaný `appid`, který musí poslat jako parametr HTTP požadavku). Pro získání předpovědi pak uživateli stačí zaslat HTTP požadavek obsahující parametry jako například appid, polohu, požadované jednotky a podobně. Server následně zašle předpověď serializovanou jako tělo HTTP odpovědi. Celé API je podrobně popsáno v dokumentaci dostupné online na `https://openweathermap.org/api` (stránka navštívena 17. 5. 2018).




**Obrázek 4.2:** Screenshot webové stránky - graf

# Kapitola 5

# Nasazení meteostanice

Posledním úkolem spojeným s tvorbou meteostanice bylo její reálné nasazení a následné zhodnocení výsledků její činnosti. V této kapitole tedy v krátkosti popíši konstrukci konečné zástavby meteostanice a zhodnotím výsledky měření při jejím reálném nasazení.

## 5.1 Konstrukce konečné zástavby

Před tím, než bylo možné meteostanici umístit a spustit, bylo nezbytné vytvořit její konečnou konstrukci. Některé náchylné součástky musí taková konstrukce ochránit před nepříznivými povětrnostními vlivy. Zároveň však tato konstrukce nesmí zabránit jednotlivým senzorům v možnosti správně měřit.

Anemometr, srážkoměr a korouhvička žádnou ochranu nepotřebují. Ty tedy stačilo připevnit na společný stojan, který byl součástí setu náhradních dílů k meteostanici *WH1080*[1].

Senzor teploty a relativní vzdušné vlhkosti (DHT22), luxmetr (TSL2561) a barometr (BMP280) bylo třeba ochránit před deštěm. K tomu posloužila krabička, která byla taktéž součástí zmíněného setu náhradních dílů k meteostanici *WH1080*. Krabička je válcového tvaru a sestává z jednotlivých prstenců. Mezi těmito prstenci je volný prostor umožňující volné proudění vzduchu. Díky tomu jsou teplota, vlhkost a tlak uvnitř krabičky stejné, jako v jejím okolí. Senzor teploty, vzdušné vlhkosti a barometr byly tedy uvnitř této krabičky. Luxmetr byl pak neprodyšně zatavený pod průhledným krytem z plexiskla na vršku této krabičky.

Samotné Raspberry Pi bylo uzavřeno ve vodotěsné krabičce. Zároveň v této krabičce byla také baterie, ze které bylo Raspberry Pi napájeno.

Pro účely testování jsem meteostanici umístil na střechu budovy Fakulty elektrotechnické ČVUT na Karlově náměstí. Hotová meteostanice připravená k měření je zachycena na fotografiích 5.1 a 5.2.

[1] Set náhradních dílů k meteostanici WH1080 sestávající jak ze srážkoměru, anemometru a korouhvičky, tak ze stojanu a příslušenství nutnému k sestavení tohoto setu. Součástí setu byla i válcová krabička sloužící jako ochrana před deštěm pro ostatní senzory

**Obrázek 5.1:** Meteostanice - panorama

## 5.2 Hodnocení měřených výsledků

Meteostanice měřila na střeše budovy Fakulty elektrotechnické ČVUT, na Karlově náměstí, v období 18. - 20. května 2018. Cílem pokusu bylo ověřit, že všechny senzory měří korektně a výsledky měření odpovídají skutečnému stavu počasí.

Pro zhodnocení měřených dat jsem je porovnal s referenčními hodnotami. Jako referenční jsem využil data, která jsou dostupná na serveru `https://www.wunderground.com/history/` (stránka navštívena 21. 5. 2018). Konkrétně jsem využil data měřená meteostanicí, která se nachází na letišti Praha - Kbely. Vzhledem k odlišné lokalitě mé a referenční meteostanice šlo předpokládat rozdíly mezi změřenými a referenčními hodnotami. Rozdíly by však neměly být zásadní.

Po provedení zkušebního měření mohu konstatovat, že většina senzorů na mé meteostanici změřila podobná data, jako meteostanice referenční. Příkladem uvádím grafy s porovnáním mnou změřené a referenční teploty (5.3), relativní vzdušné vlhkosti (5.4) a atmosférického tlaku (5.5).

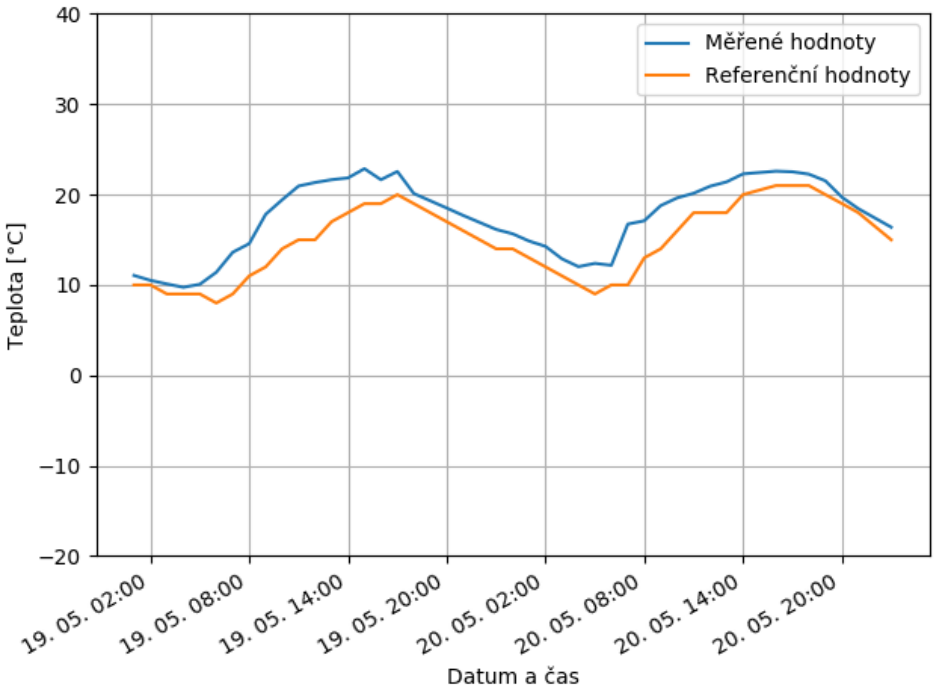
Rozdíly mezi mým a referenčním měřením jsou sice patrné, nicméně vzhledem ke zmíněné odlišné lokalitě obou meteostanic nejsou překvapivé. Meteostanice jsou od sebe navzájem vzdáleny přibližně 10 km. Má meteostanice je navíc umístěna v centru Prahy, zatímco referenční na jejím okraji. Odlišnost prostředí může mít taktéž vliv na měřené meteorologické veličiny. Tímto testem tedy nelze objektivně posoudit přesnost senzorů. Lze však pozorovat, jestli výsledky měření obsahují výrazné a nečekané anomálie, které by indikovaly špatnou funkčnost senzoru.

**Obrázek 5.2:** Meteostanice - detail

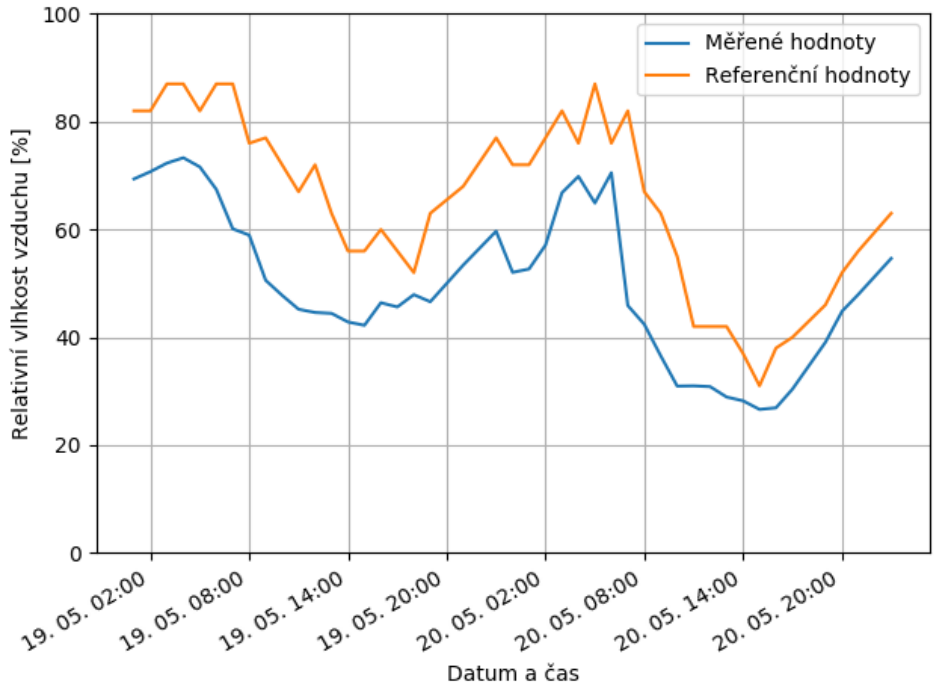
U většiny senzorů jsem nepozoroval žádné anomálie. Jediný senzor, u kterého jsem prokázal jeho omezenou funkčnost, je luxmetr. Ten byl bohužel při vystavení slunečnímu svitu za jasného dne saturován. Jelikož byly oba dny, kdy jsem měření prováděl, slunečné, byl luxmetr, až na krátké výjimky, přibližně mezi 9:00 - 17:00 nepřetržitě v saturovaném stavu. Validní data tedy měřil jen ráno, v podvečer a samozřejmě v noci.

Senzor, který jsem tímto testem nemohl prověřit, je srážkoměr. Důvodem je fakt, že během dvou dnů, kdy jsem test prováděl, v místě měření nepršelo.

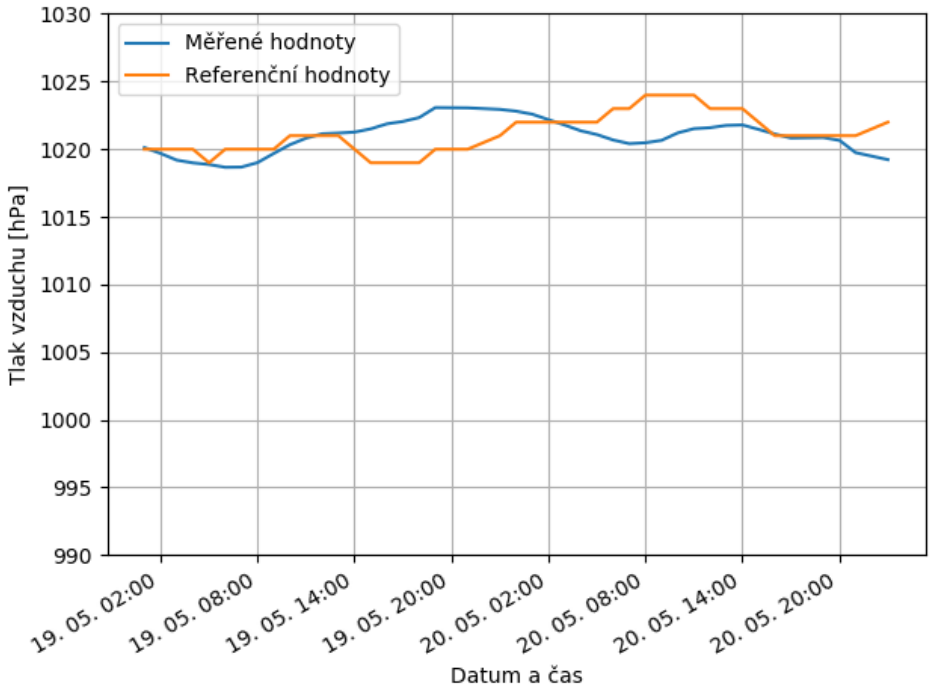
Objevným výsledkem tohoto testu je tedy zjištění, že luxmetr je nespolehlivý při vysoké intenzitě osvětlení. Kromě luxmetru jsem však u žádného jiného senzoru nepozoroval anomálie, které by indikovaly jeho špatnou funkčnost.




**Obrázek 5.3:** Test - měření teploty




**Obrázek 5.4:** Test - měření relativní vzdušné vlhkosti




**Obrázek 5.5:** Test - měření tlaku vzduchu

# Kapitola 6

# Závěr

Cílem práce bylo navrhnout, vyvinout a otestovat meteorologickou stanici na platformě Raspberry Pi. Tento cíl práce se podařilo splnit. Ze strany hardwarového návrhu a zpracování dat testy naznačují, že vše, kromě problematického luxmetru, funguje správně. Po softwarové stránce projektu se podařilo implementovat veškerou požadovanou funkcionalitu, od zpracování dat, přes databázi k ukládání dat, až po vytvoření rozhraní pro přístup k výsledkům měření, včetně vyhodnocení jednoduché předpovědi na základě aktuálního atmosférického tlaku.

Meteostanice je navíc vytvořena takovým způsobem, že ji lze snadno rozšířit o další funkcionalitu. V případě potřeby by nebyl problém připojit další senzory, měřící další meteorologické veličiny. Současně je možné velice snadno rozšířit webový server. Přidáním jedné metody lze doplnit schopnost zpracovat a vyhodnotit prakticky libovolné požadavky, které se v budoucnu budou jevit jako užitečné. Jelikož je účelem této meteostanice posloužit budoucímu plně samostatnému provozu autonomních bezpilotních prostředků, jejichž účel a specifické potřeby nejsou předem známy, vidím rozšiřitelnost meteostanice jako její důležitou vlastnost.

Jako velice zajímavý bod této práce hodnotím samotný výběr platformy Raspberry Pi spolu s programovacím jazykem Python. Tyto technologie se ukázaly jako velice mocný a univerzální nástroj pro tvorbu vestavěných systémů. Největší sílu v kombinaci těchto technologií vidím v jejich všestrannosti. Díky dobré knihovní podpoře jazyka Python totiž umožňují jak snadné programování na nízké úrovni (obsluhu vstupně výstupních pinů, komunikaci přes periferie), tak velice elegantní implementaci vysokoúrovňové softwarové nástavby (v mém případě implementace webového serveru a propojení s MySQL databází). Možnost nainstalovat MySQL server přímo na Raspberry Pi bylo při vývoji taktéž velmi výhodné. Meteostanice vytvořená v rámci této práce tedy byla dobrým příkladem, jak pomocí zvolených nástrojů elegantně vybudovat zařízení s poměrně komplexní funkcionalitou.

# Literatura


[1] Atmosférický tlak. *meteocentrum.cz*. Dostupné z `https://www.meteocentrum.cz/zajimavosti/encyklopedie/atmosfericky-tlak`, získáno 14. 5. 2018.

[2] TSL2560, TSL2561 Light-to-Digital Converter. *Data Sheet, TAOS Inc., TAOS059N*, březen 2009.

[3] Matt Hawkins. Using the BME280 I2C Temperature and Pressure Sensor in Python. July 2016. Dostupné z `https://www.raspberrypi-spy.co.uk/2016/07/using-bme280-i2c-temperature-pressure-sensor-in-python/`, získáno 13. 5. 2018.

[4] Texas Instruments. Ultra-small, low-power, 16-bit analog-to-digital converter with internal reference. Říjen 2009.

[5] Jiří Jaša. Meteostanice – měření srážek, rychlosti a směru větru. *Vodnický Blog*, listopad 2017. Dostupné z `https://www.vodnici.net/wiki/meteostanice-mereni-srazek-rychlosti-a-smeru-vetru/`, získáno 6. 4. 2018.

[6] Thomas Liu. Digital relative humidity & temperature sensor AM2302/DHT22. 2012. Dostupné z `https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf`, získáno 6. 4. 2018.

[7] Bosch Sensortec. Digital pressure sensor. Květen 2015. Dostupné z `https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf`, získáno 6. 4. 2018.

[8] Argent Data Systems. Weather Sensor Assembly p/n 80422. Dostupné z `https://cdn.sparkfun.com/assets/8/4/c/d/6/Weather_Sensor_Assembly_Updated.pdf`, získáno 14. 4. 2018.