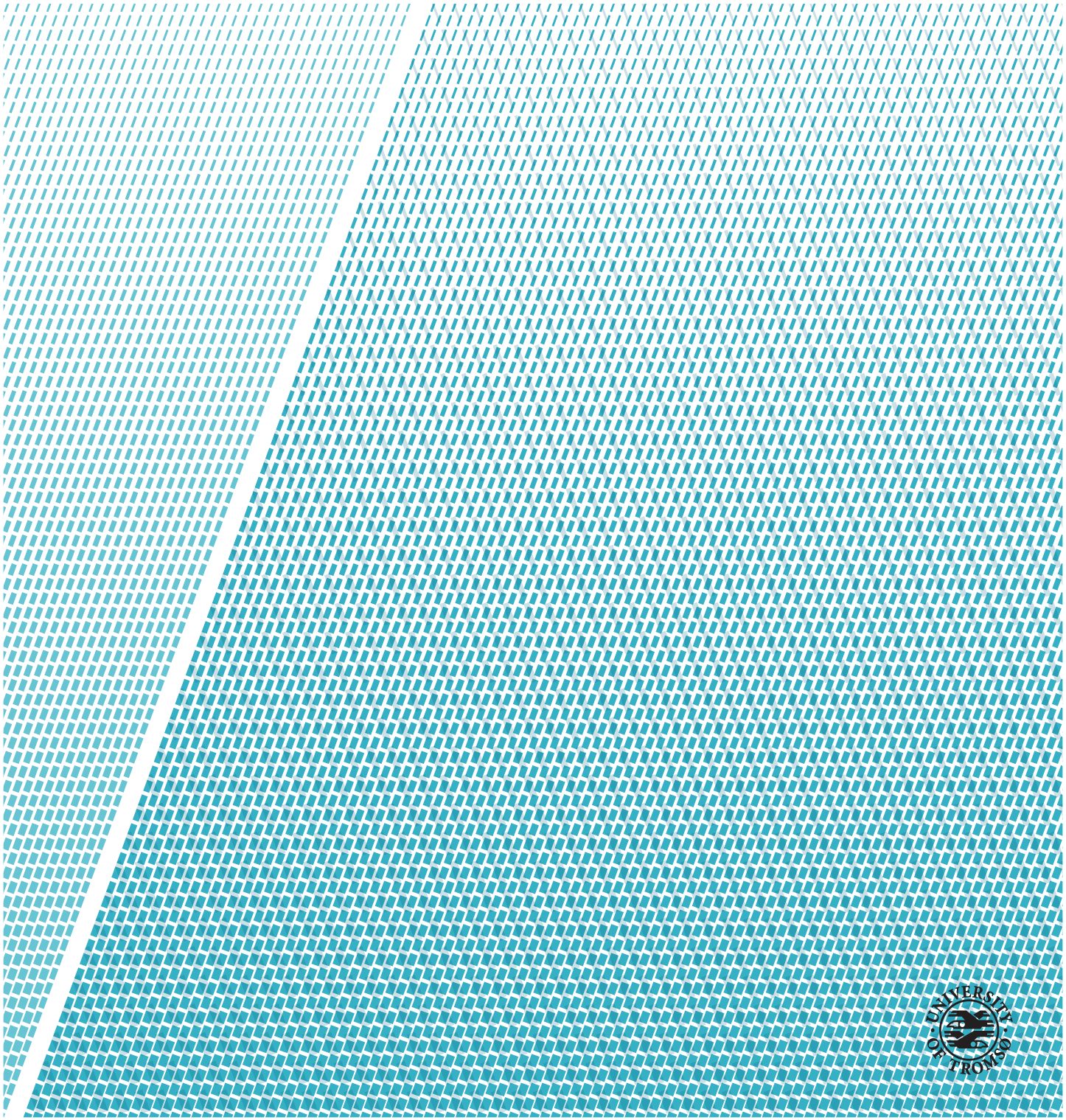
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology
Department of Computer Science

From Physical to Virtual Sensors (PVS)

Camilla Stormoen

INF-3983 Capstone Project in Computer Science . . . December 2017



Abstract

W3 Whats wrong with the world? / motivation 1-3 setninger

Architecture - 1-3 setninger

Design- 1-3 setninger

Implementation - 1-3 setninger

Experiments - 1-3 setninger

Results - 1-3 setninger

Lessons learned/main conclusion - 1-3 setninger

Kutt heller etterpaa

This dissertation present/describe ...

Contents

List of Figures

1 Introduction

The Arctic tundra in the far northern hemisphere is challenged by climate changes in the world today and is one of the ecosystems that are most affected by these changes [10]. The *Climate-ecological Observatory for Arctic Tundra - COAT* is a long-term research project developed by five Fram Center[1] institutions. Their goal is to create robust observation systems which enable documentation and understanding of climate change impacts on the Arctic tundra ecosystems. In autumn 2015, COAT was granted substantial funding to establish research infrastructure which allowed them to start up a research infrastructure during 2016-2020 [10].

Every year, COAT deploys several camera traps in eastern Finnmark, Norway for roughly one month during the winter. The main purpose of these cameras is to study predator populations, with especially a focus on the redlisted arctic fox which is the most endagered mammal species in Norway [9]. The camera traps are set up to take a time-lapsed photo every fifth minute during day and night which adds up to over 300 000 images per year [11]. Collecting this high volume of images gives Big Data challenges in the ecology field.

Hvordan komme inn på virtuelle sensorer? Og til "this project will develop an abstraction" eller noe lignende...

In this project we will describe a system that simplifies biologists need to look

1. `http://www.framsenteret.no/english`

at images from different locations in the Arctic tundra in one view...

Assume biologists want to search for image of a specific animal from different locations in the Arctic Tundra. A virtual sensor would locate data from the data storage and then return those images that matches to the biologists request...

1.1 Motivation

The motivation behind this project is that no single sensor may cover the sensing needs, and that sensing needs can change rapidly over time. Consequently, there is a need for sensor fusion, and allow for combining sensors at different computers.

This project will develop an abstraction for virtual sensors, and do a prototype of the abstraction on a set of computers with physical sensors. The purpose is to provide for a more powerful and flexible sensor in the COAT monitoring of the arctic tundra. As such, a fox feeding station is the usage domain to be used for the prototype.

1.2 Contributions

The dissertation makes the following contributions:

- An introduction to virtual sensors.
- A description of data set preparation from the camera traps in the Arctic tundra.
- An implementation and description of a system that fuse raw data from different physical sensors into one virtual sensor.
- An evaluation of the system.

1.3 Assumptions

AVGRENSE, VIKTIG!! Something about motivation and stuff

1.4 Limitations

AVGRENSE, VIKTIG!!

- Mention focus on camera-sensors/data, and not other sensors?!
- Hvorfor fokus på bilder - forenkle, begrense, starte et sted
- Limiting search for animals - "predefined" what user can search

2

Related Work

2.1 Virtual Sensors

A virtual sensor is a constructed sensor in contrary to a physical sensor [1]. They are used in place of the real sensors where they read real physical sensor data and calculate the outputs by using some processing models.

Previous research have focused on simple in-network data aggregation techniques and the sensor networks are often represented as a database. Two examples of such approaches are TinyDB [6] and Cougar [7], which enable applications to have a central point (base station) and create routing trees to funnel replies back to this root. The main focus on these approaches are operating intelligent in-network aggregation and routing to reduce the overall energy cost. In both examples, the data aggregation is specified using an SQL-like language. However, queries cannot be used to merge different data types, only homogeneous data aggregation is achievable. Their virtual sensors approach is offering a simple interface and heterogeneous in-network data aggregation. Our virtual sensors have no SQL-like language or a database to store the aggregated data. Our work also fuses physical sensor data which is already in a data storage, and not consuming directly from the physical sensors.

A virtual node[2] is a development of a set of physical sensors that a programmer can interact with as a single sensor. Their virtual nodes are implemented using TinyOS [3] and is specified using logical neighborhoods [4][5]. The nodes

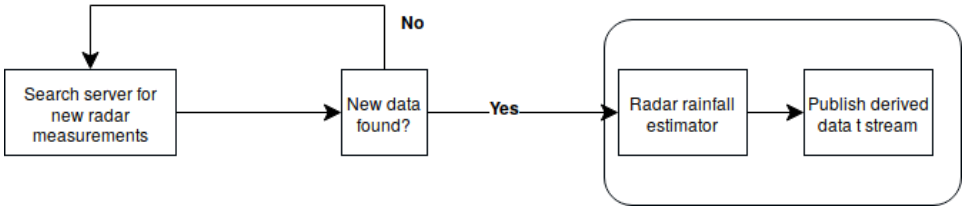


Figure 2.1: Figure illustrating the virtual rainfall sensor.

in the logical neighborhood are specified based on their characteristics by the programmer. Our approach of virtual sensors are not specified by the use of logical neighborhoods. There are no communication between the virtual nodes.

It is a prototype of a virtual sensor system for environmental observation with real-time customization of physical sensor data[8].

The users can leverage deployed virtual sensor types, interactively creating and sharing new customized virtual sensors at required locations and with parameters best suited to the researcher's purpose.

The system can provide point-averaged radar-rainfall products at either temporal resolution of the radar or as a temporal average of a fixed time period, which is illustrated in Figure 2.1 In contrast, our virtual sensors system is running on-demand, not in real-time. **OGSÅ ANDRE ULIKHETER**

Dice?? [12]

FROM PAPER: We presented DICE (Distributed Invariant CheckEr), a system for WSN-based distributed monitoring of global invariants in physical processes. DICE provides a declarative language to specify invariants and a runtime support enabling efficient monitoring of their violations.

3

The Dataset

The dataset is provided by COAT and contains over 1.6 millions of pictures taken from 2011 to 2015 by their camera traps stationed in the Arctic Tundra in Finnmark, Norway. The wildlife camera traps from Reconyx are placed at six different areas in the Arctic Tundra: Nordkynn, Ifjord, Komag, Nyborg, Stjernevann and Gaissene. The dataset contains pictures during night- and daytime. The cameras have infrared flash so the cameras can take pictures at nighttime. These pictures are without color while the pictures taken at daytime are with color, as shown in Figure 3.1.

3.1 Dataset Directory Structure

The pictures in the dataset is structured in directories and folders. The dataset is first divided into different years going from 2011 to 2015. Each of these folders are then again divided into the areas of the 5 camera traps in the Arctic Tundra. Inside each of these folders specified by the area, they contain folders from each site inside the specific area. Figure 3.2 shows a cropped screenshot of the directories.

NOTES: (In total 1976 directories, men alle er ikke i bruk/relevant)

(a) Picture of ravens at daytime.

(b) Picture of an arctic fox at nighttime.

Figure 3.1: Images from the COAT dataset showing how a picture can contain colors or greyscale.

3.2 Image Annotations

COAT provided annotations stored in excel-files with information about all of the images in the dataset that biologist and ecologists working at the COAT project have detected. The biologist have used these annotations to see what animal was present at a specific time and to see statistics. The biologist have not made these annotations to go back in time and look at the images, and the dataset is then not made for...

The excel-files are structured in their own folder and each file is named as a combination of the camera traps area, it's year and the site. An example is "*fotoboks2011_nordkynn_nordkynn.2011.xlsx*".

NOTES: Ta med hvordan/hva slags metadata som ligger i excel-filen? Evt i bildene.

The dataset contains the images metadata and animal classification, but did not contain any filenames or a filepath. We then had to find a method to find out which images correspond to the metadata in the excel-files. Fortunately, each image had metadata stored in Exchangeable Image File Format (EXIF). We recursively traverse the directories where images are located and read date-time from the metadata and store it in a new CSV-file (Comma-separated Values). This was quite a time-consuming task because of the number of images to process.

NOTES: Forklare mer hvordan dette er gjort i design/implementasjon. Skal jeg si noe som ".. explained in details in Chapter/Section ...??

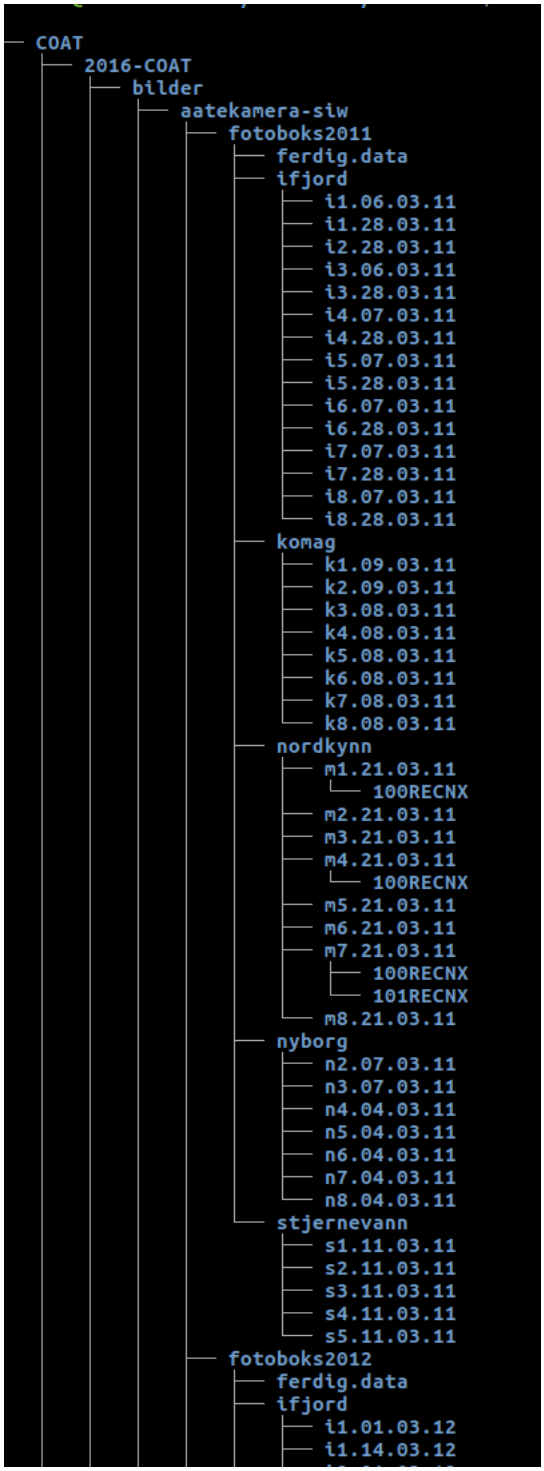


Figure 3.2: Figure showing a cropped screenshot of the structure of the directories.

4

Architecture

This chapter describes the architecture of the system. The components in the system are: physical sensors, raw data, analytics and classified data, fusion of data, fused data, virtual sensors and the user interface. The architecture of the system is presented in Figure 4.1. The arrows indicates the dataflow between each component in the system.

4.1 Physical Sensors

The physical sensors produce raw data. The raw data consists of images from different bait-camera sensors. The data also contains metadata about each image, as described in Chapter 3.

4.2 Raw Data, Analytics and Labeled Data

The raw data contains images from the physical sensors and annotations in excel-files containing metadata about each image alongside labeled data, as described in Chapter 3.

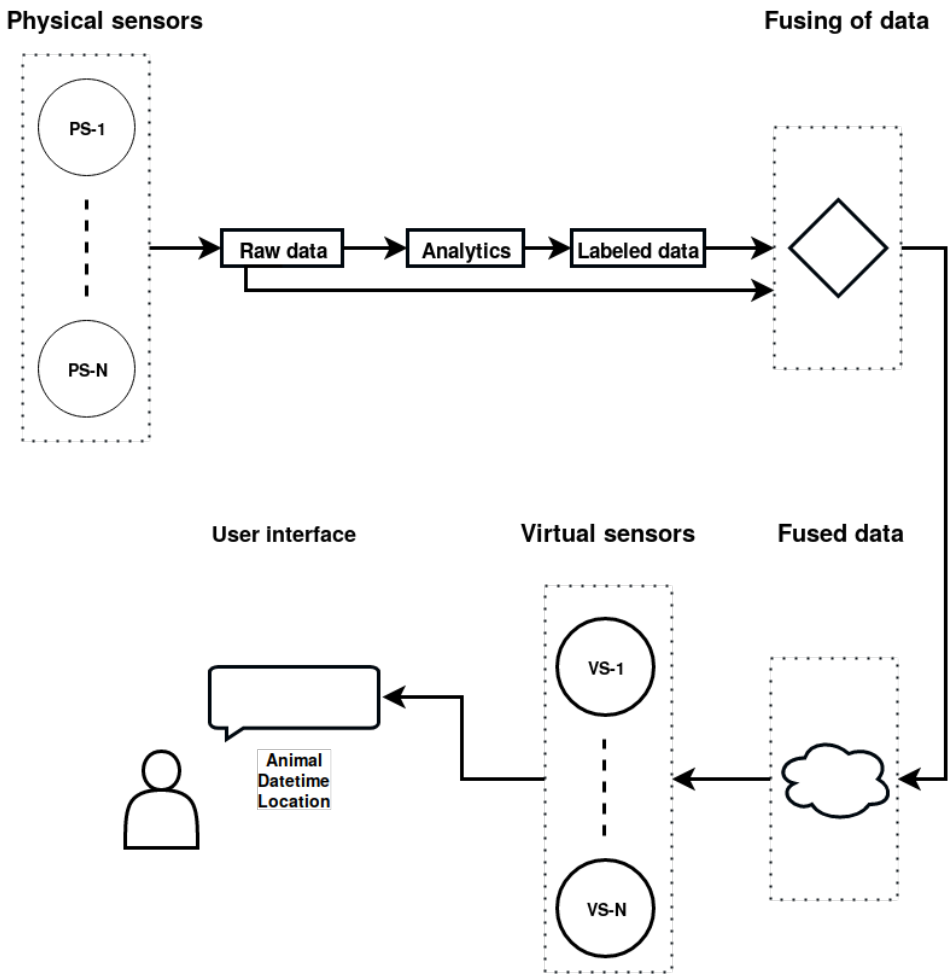


Figure 4.1: Figure showing the system architecture.

4.3 Fusing Of Data

Get analytics and classified data from raw data (excel) and the raw-data and fuse/aggregate correlated/corresponding(?) data.

4.4 Fused Data

The fused data retrieves it's data and on demand. The fused data is the physical sensors location combined with necessary metadata such as date-time, location, site, year, what kind of animal there as in the image and also how many animals there was/**and animal classification**.

4.5 Virtual Sensors

The virtual sensors are divided into multiple sensors where each virtual sensor represents a different animal that scientists are interested in, like ravens, red foxes, golden eagles or polar foxes.

NOTES: UTDYP DE VIRTUELLE SENSORENE These virtual sensors have the same functionalities...?

The virtual sensors offers interaction between the fused data and the user interface. A user tells the user interface, described in Section 4.6, a task and forwards the task to the desired virtual sensor. The virtual sensor will then retrieve its results from the fused data.

4.6 User Interface

A user wants to retrieve fused data about animals. The user interacts with the virtual sensor user interface. This takes care of the interaction with the virtual sensor. When a user gives a command, the desired virtual sensor retrieve data from the fused data as described in Section 4.5, and presents the result back to the user.

NOTES: utdype query språk - utdype mer i design/implementation?

5 Design

In this chapter we will look at the design of the system and present the design of each component of the architecture. Figure 5.1 shows the design of the system. The arrows indicates the communication lines and dataflow between each component in the system. The dotted arrows show how the data storage is structured with files and raw data.

5.1 Data Storage

The data storage contains raw data from the physical camera trap sensors and excel-files containing annotations for each raw data, as described in Chapter 3.

5.2 Sensor Fusion

As mentioned earlier in Chapter 3, the dataset did not contain any filename or a filepath. It only contained image metadata such as a pictures location, date-time, site, year and animal classification. We then had to find a method to find out which images correspond to which annotation in the excel-sheet.

We recursively traverse the directories where pictures are located and read

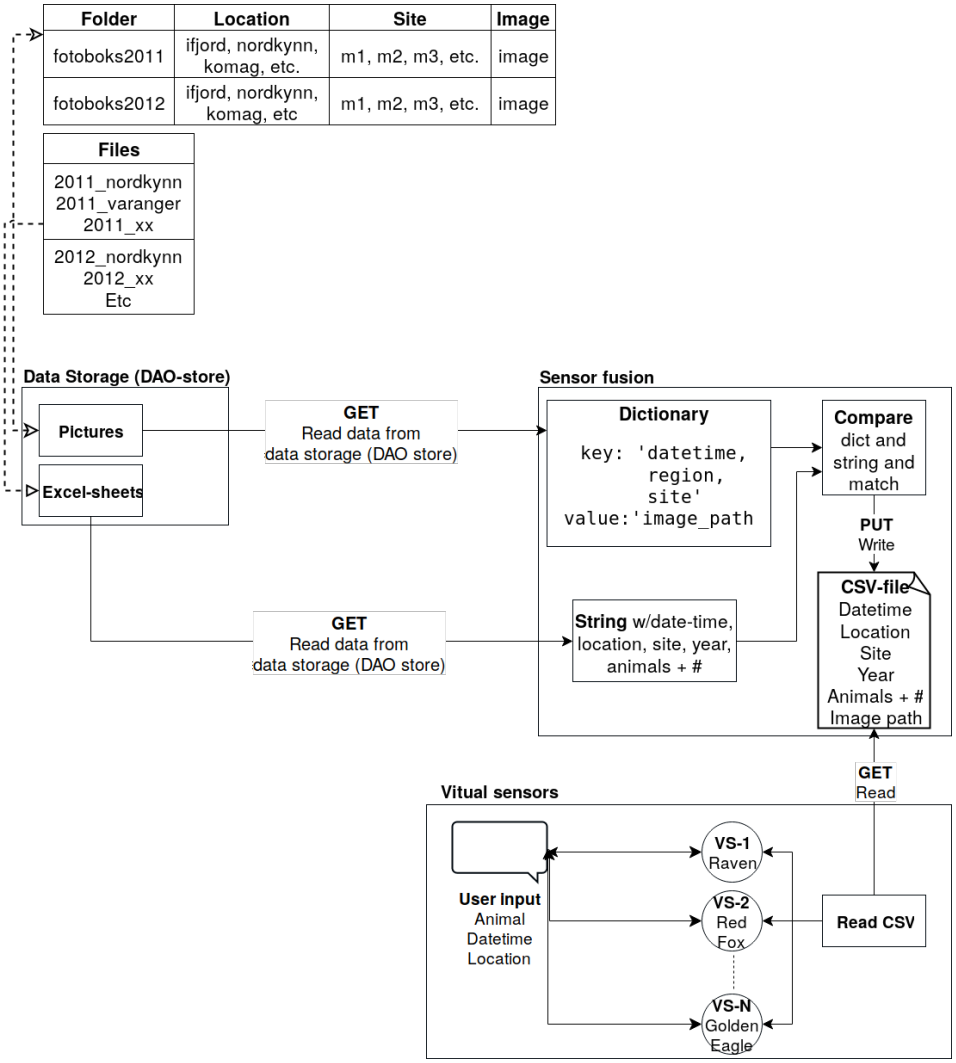


Figure 5.1: Figure showing the system design.

date and time from the metadata for each picture along with the annotation in the excel-sheet. The images and annotations that matched, is stored in a Python dictionary, like a key-value store, with date-time as key and image-path as value. Images and annotations that matches is stored in a new CSV-file. As mentioned earlier, was this quite a time-consuming task because of the numbers of images to process and compare.

5.3 Virtual Sensors

5.3.1 Virtual Sensors

The virtual sensors are divided into multiple sensors depending on animals. The respective virtual sensor must handle events from the user. Assume a user wants to find pictures of a red fox at a specific time. The user interface makes the respective virtual sensor aware of the task. The virtual sensor will then go to a list of all red foxes that are located from the fused data and search for the specific request. If the virtual sensor find pictures that are similar to the request from the user, one and one picture is visualized on the computer screen to the user.

The virtual sensors in this approach are functions in a sequential **system/program**. They could have been threads in a parallel program or the best solution would be if the virtual sensors where independent subprocesses because that would not affected other virtual sensors when a virtual sensor is added or removed from the system, as well as if one virtual sensor crashes. *Er dette mer diskusjonsmateriale? Eller ha det med her også?*

5.3.2 User Application

The user-application is the connection between the user and the virtual sensors. It gets input from the user and sends a request to the virtual sensor based on its input. The user-applications goal is to make an interface that is easy to use and to show images that the user want to see. The main menu is shown in Listing 5.1. The user only need to specify the following 3 parameters for the virtual sensor:

- **Animal:** The user can choose to see 3 different animals; raven, red fox and the golden eagle. The user can also chose to see pictures with no animals in it. The reason that the user only can chose between these 4 categories is that these are the only virtual sensors that are implemented in this version of the system.

- **Date-time:** If the user wants a specific date-time the picture(s) was taken, he/she can specify the date and time in this section. If the user leave this field blank, all dates will included when the system search for pictures.

- **Location:** At last, the user can chose if he/she wants the picture to be located anywhere specific. The only valid arguments in this field is Nordkynn, Komag, Nyborg or Stjernevann since these are the only names in the CSV-file in the fused data.

Listing 5.1: Main menu

```
**** WELCOME! ****
******************

Write what you want to see
(Raven, RedFox, GoldenEagle):

Write when you want to see (blank is all dates):
E.g: 2011:03:24 or 2011:04:24 10:10:00

Where do you want the animal to be located
(nordkynn, varanger: komag, nyborg, stjernevann):
```

6 Implementation

6.1 Overview

This chapter will elaborate on how we implemented the system, general implementation requirements, issues and choices. We will first look at some libraries used in this implementation, then we will take a look at the data storage in Section 6.2 and the sensor fusion in Section 6.3. At last, we will describe the virtual sensors and the user application in Section 6.4.

The system is implemented and written in the high-level programming language Python 2.7[1]. **This was a practical ...choice/language because ... reason 1 (kjent med det fra før av), reason 2(object orientert), reason 3...(offers many/different (open-source) libraries)**. The libraries are mentioned below.

Exchangeable Image File Format - EXIF

Exchangeable Image File Format (EXIF) is a standard that specifies the formats for images, sound, scanners and other systems recorded by digital cameras. This standard consists of image information of the EXIF image file or the EXIF audio file.

1. https://www.python.org/

Python have a module called EXIF 2.1.2[2], to extract the metadata(image information from the jpg-files.

Pandas Dataframe

Pandas[3] is an open source Python Data Analysis Library which provides high-performance and easy-to-use data structures. It is a NUMFocus[4] sponsored project which will help ensure success of the development of pandas as a open-source project in world class. A dataframe[5] is a two-dimensional, dynamic tabular data structure with labeled axes (rows and columns), just like an excel-file.

OpenCV

OpenCV V2.4.9.1[6] is an open-source library which provides the functionality for loading and preprocessing images.

6.2 Data Storage

The data storage contains raw data and prepossessed images that are structured and sorted by animals. Unfortunately, these preprossesed images don't consists of any image metadata so we don't have any information of when the image is taken. Therefore, the solution is to traverse the whole data storage and find the raw data with metadata and compare these to the excel-files containing information like date, time, location and site.

6.3 Sensor Fusion

6.3.1 Raw Data Information Extraction

The raw data in the data storage is traversed by walking the OS directories. If the traversed file ends with JPG, the image information in the EXIF is being

2. https://pypi.python.org/pypi/ExifRead
3. http://pandas.pydata.org/
4. https://www.numfocus.org/open-source-projects/
5. https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html
6. https://opencv-python-tutroals.readthedocs.io/en/latest/

extracted and the datetime value is read. This datetime along with the images site, region and image path is stored in a Python dictionary, where the datetime, site and region is key and the image path is the value. This was as mentioned earlier, a time-consuming task because of the 1.6 millions of pictures that was being processed.

During implementation we came across a problem with the folders containing white-space and UTF-8 character. An example of a folder is *'/../Bilder til institutt for informatikk/åtekamera (Siw)'*. This makes it difficult to extract images due to the reason that the new CSV-file had white-space between each characters. This is described further in Section 6.3.3. Our solution was to change the folder-names to not contain whitespace and UTF-8 characters.

6.3.2 Excel files Extraction

The excel files contains annotations for each image. To read excel files, we use Pandas Dataframe, described in Section 6.1. From the excel files, we extract information like date, time, animals, region, site and year. The necessary information about each animal is stored in a string.

During implementation we came across a problem with inconsistency between cells in the same row. The problem was that the datetime cells information was inconsistent with the information in cells day, month, year, hours and minutes, shown in Figure 6.1. Our solution was to alter and concatenate the correct values to make a new datetime.

Another problem was that the cells containing column-names was different. One example is that month was written "*moth*" in some excel files and "*month*" in others. Because of time constraints, we only used the excel files containing the word "moth" since this was the first excel-file to be extracted and our intent was to get images to visualize.

A different problem was that some column-header names was written with a big letter like "*Time*" in some excel files and other contained "*time*". Our solution

23	3	2011	7	40	0	m7.11	T	3/22/2011
23	3	2011	7	50	0	m7.11	T	3/22/2011
23	3	2011	8	0	0	m7.11	T	3/22/2011
23	3	2011	8	10	0	m7.11	T	3/22/2011
23	3	2011	8	20	0	m7.11	T	3/22/2011
23	3	2011	8	30	0	m7.11	T	3/22/2011
23	3	2011	8	40	0	m7.11	T	3/22/2011
23	3	2011	8	50	0	m7.11	T	3/22/2011

Figure 6.1: Shows inconsistency between cells in rows in excel file.

was to make every column header into lowercases using Pandas Dataframe function, *pandas.Series.str.lower()*.

One more problem arose when some of the excel-file contained table-headers with UTF-8 characters, as seen in Figure 6.2. These characters won't be interpreted by the system and the table-headers will not be readable.

6.3.3 Comparison of data

The extracted metadata from the images and the annotations in the excel files that matches, is stored in a CSV-file (Comma-separated Values). We chose to store the fused data in a new CSV-file because of the large number of images to process. It will simplify search in the future since it will be faster to search through the fused data in the CSV instead of searching through the raw data and compare them with the annotations. This is not the best choice, however it is a introduction to a solution.

NOTES: folder-path inneholdt mellomrom og æøå, gjorde det vanskelig å hente ut bilder når pathen krøllet seg... brukte litt tid på å prøve meg fram, endte med å snakke med JM om å endre folder-navn uten mellomrom og æøå.. Snakke om det i compare når jeg leser fra csv-filen ettersom det er i csv-filen det bugger seg når jeg skal lese ut path når den inneholder space og æøå?

As described in Section 6.3.1, the folder-names contained white-spaces and UTF-8 characters.

6.4 Virtual Sensors

6.4.1 Virtual Sensors

The virtual sensors are implemented as functions, not threads or independent subprocesses, as mentioned in Section 5.3.1. The reason for this choice is because ... **..start somewhere and eventually lack of time/time constraints and wanted to show something instead of nothing..**

fjellrev	havÃ¸rn	jerv	kongeÃ¸rn	krÃ¥ke

Figure 6.2: Shows table-headers with unsupported UTF-8.

We also chose to focus on virtual sensors for three different animals and images without any animals in it. The three animals were the red fox, ravens and the golden eagle. We made this choice because these were the animals that were **most present/appeared the most** in the annotations in the excel files that were extracted.

The OpenCV library is used to visualize images to the user. The library read the images it is given and display these on the screen to the user. The reason for choosing this visualization library was because of experience with this in previous projects and also that our focus was not primarily on which library to use when visualizing images.

6.4.2 User application

The user application is implemented as an interface in the terminal that communicates with the virtual sensors.

There is currently not possible for more than one user to be connected in the same application at a time. Because of lack of time, we have not thought about implementing this. Our focus has been making a prototype of the user interface to be easy to use for biologists/users and to show images on the users request.

7 Evaluation

This chapter describes the experimental setup and metrics used to evaluate the implemented system.

7.1 Experimental Setup

All experiements were done on a Lenovo ThinkCenter with an Intel® Core™ i5-6400T CPU @ 2.20GHz × 4, Intel® HD Graphics 530 (Skylake GT2), 15,6 GiB memory and 503 GB disk. It ran on Ubuntu 17.04 64-bit with gcc V6.3.0 compiler and Python V2.7.13.

7.2 Experimental Design

We decided to measure execution time on the five different experiments. We measured execution time on the new csv-file that contained only fused data from Varanger and the new CSV-file that contained fused data from Varanger and Nordkynn. We also ran three experiments where we tried to run the comparison between images and annotations concurrently with 4, 100 and 500 processes by using a multiprocessing library in Python that supports spawning processes using an API similar to the threading module.

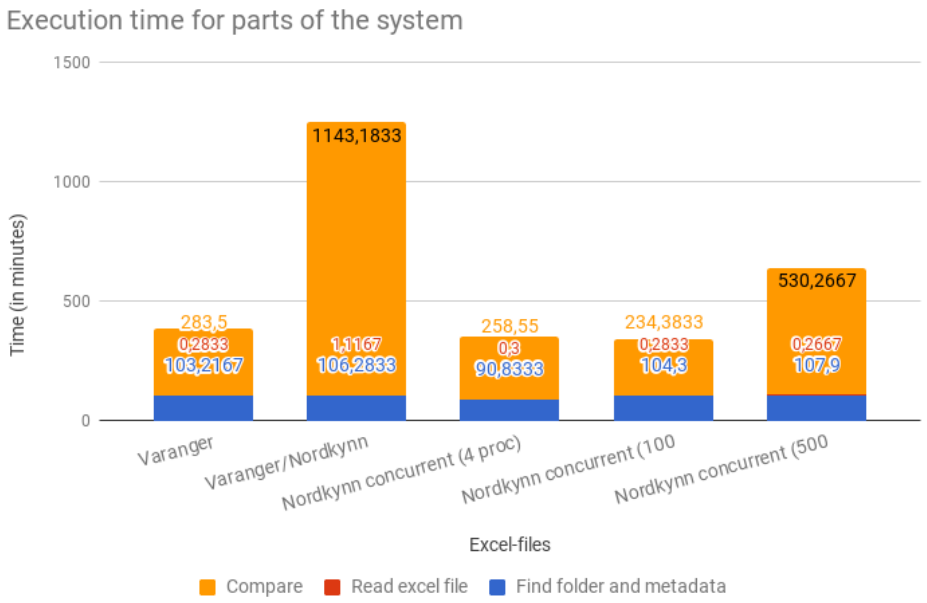


Figure 7.1: Figure showing time to execute the system (in minutes).

7.3 Results

In this section we will discuss the results of the testing described in the sections above...

What does the result say? Each experiment, result, meaning

7.3.1 Measure time to execute

Most time during finding folder and read metadata and comparing.. Read excel file is faaaast!!

Figure 7.1 shows the time to execute different parts of the system in minutes... We can see that the comparison between images and annotations is the part of the system that takes the longest time followed by the part that finds folders and read metadata from the raw data.

When only Varanger, same time for sequential" and concurrent.. Eneste som skiller seg ut er Varanger og Nordkynn sammen, men ikke så rart ettersom det er fra 2 lokasjoner og ikke en som de andre.. Mer data å sammenligne. Concurrent med 500 processer skiller seg også litt ut i

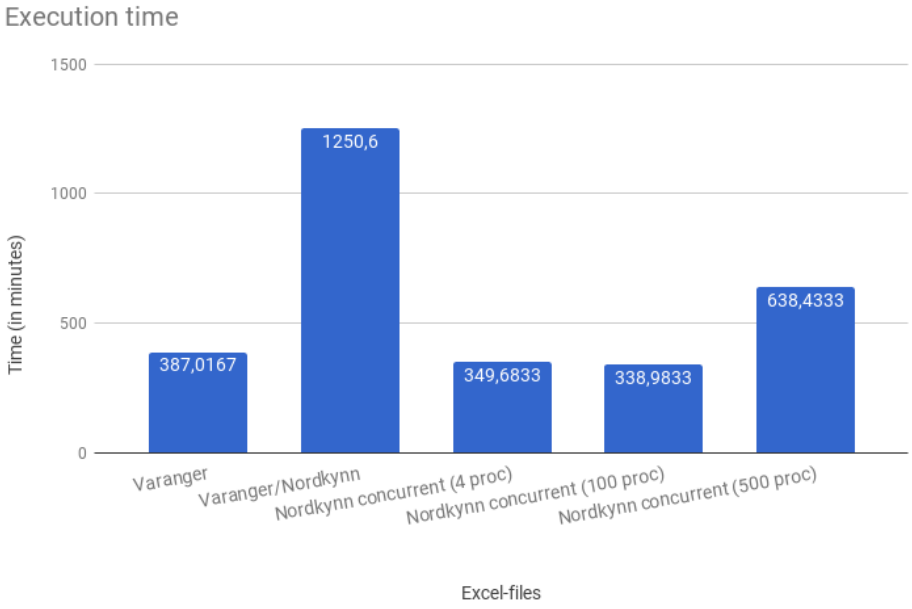


Figure 7.2: Figure showing time to execute the system (in minutes).

forhold til de andre concurrent, men tror dette har med overhead eller noe å gjøre..

Figure 7.2 show the time for the whole system to run. As we can see, ...

7.3.2 Measure CPU/memory??

Doing the experiments now..

8

Discussion

- Virtual sensors probably uavhengige prosesser, ikke threads ettersom man evt vil addere flere sensorer og unnga a starte alle sensorer på nytt igjen.. HVORFOR valgte jeg løsningen som jeg gjorde, og er dette er andre løsninger..

- Burde lage egne gorutiner/subprocesser/classer for hvert dyr. alle er en basic class med f. eks ID, TYPE, CAPABILITIES, HISTORIKK?, QUERIES/RAW DATA..

- Updates of "database" in background, not on demand -> future work?

- hash-map instead of multiple lists with different animals. hash on animal and/or place

- parallel/concurrent program when doing metadata-collecting and reading from excel-files and writing to csv. Python - bad experience with threads/subprocess, golang probably better?

- Et script som går gjennom alle excel-filene som den ikke har lest og lagrer data eller noe sånt.. Bør gjøres i bakgrunnen/ikke on-demand.

- Drawback with Python - not that good at threading/parallel/concurrency.. Python is not good for multi-processor/multi-core work. Prøvde å bruke multiprocess-bibliotek med pools med processer.. Ble litt bedre, men ikke

noe spesielt.. Golang ville kanskje vært bedre mtp concurrency

- Forskjellig språk har focus på forskjellig ting og er gode forskjellige ting.. Uansett hvilket språk man velger pga en fordel, vil det mest sannsnylig være et annet språk som er bedre på en annenn ting du skal implementere.. som f. eks. Golang bedre concurrent enn Python..

This chapter discusses our approach, experience and why we chose the solution we ended up with...

8.1 Virtual Sensors

8.2 BBB

8.3 CCC

9 Conclusion

In this project, we have implemented a system...

Our experiments showed that ...

9.1 Contributions?

Har denne i introen også..

10

Future Work

- Concurrent lesing av excel-filer (alle sammen)
- Concurrent lesing av metadata og traversering..
- Concurrent sammneligning
- VS er subproseser/channels(?) i GO slik at man evt kan fjerne/legge til sensorer om man vil
- Gjøre i Golang?
- Concurrent VS: kan hente data fra flere sensorer eller kombinere flere VS.. Eks: ravn & rev eller ravn | rev.
- Query for å hente bilder mellom 2 tidspunkt
- Oppdatere/sjekek frequent, ikke som nå (on-demand).

11

Appendix?

readme, source code, dataset measurement RAW

Bibliography

[1] S. Kabadayi and A. Pridgen and C. Julien, *Virtual Sensors: Abstracting Data from Physical Sensors*, 2006, in *2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks(WoWMoM'06), 6 pp.-592*.

[2] Ciciriello, Pietro and Mottola, Luca and Picco, Gian Pietro, *Building Virtual Sensors and Actuators over Logical Neighborhoods*, 2006, in ACM, 19–24. *http://doi.acm.org/10.1145/1176866.1176870*.

[3] Hill, Jason and Szewczyk, Robert and Woo, Alec and Hollar, Seth and Culler, David and Pister, Kristofer, *System Architecture Directions for Networked Sensors*, 2000, in *ASPLOS-IX: Proc. of the 9 nt Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 93–104.

[4] Mottola, Luca and Picco, Gian Pietro, *Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks*, 2006, in *Distributed Computing in Sensor Systems: Second IEEE International Conference, DCOSS 2006, San Francisco, CA, USA, June 18-20, 2006. Proceedings*, pages 150–168. *https://doi.org/10.1007/11776178_10*.

[5] Mottola, Luca and Picco, Gian Pietro, *Programming Wireless Sensor Networks with Logical Neighborhoods*, 2006, in *Proceedings of the First International Conference on Integrated Internet Ad Hoc and Sensor Networks*, series = InterSense '06, pages 150–168. *http://doi.acm.org/10.1145/1142680.1142691*.

[6] Madden, Samuel R. and Franklin, Michael J. and Hellerstein, Joseph M. and Hong, Wei, *TinyDB: An Acquisitional Query Processing System for Sensor Networks*, March 2005, in *ACM Trans. Database Syst. Vol. 30, Nr. 1*, pages 122–173. *http://doi.acm.org/10.1145/1061318.1061322*.

[7] Yao, Yong and Gehrke, Johannes, *The Cougar Approach to In-network Query Processing in Sensor Networks*, September 2002, in *SIGMOD Rec., Vol. 31, Nr. 3*, pages 9–18. *http://doi.acm.org/10.1145/601858.601861*.

[8] David J. Hill and Yong Liu and Luigi Marini and Rob Kooper and Alejandro Rodriguez and Joe Futrelle and Barbara S. Minsker and James Myers and Terry McLaren, *A virtual sensor system for user-generated, real-time environmental data products*, 2011, in *Environmental Modelling & Software, Vol. 26, Nr. 12*, pages 1710 - 1724. *http://www.sciencedirect.com/science/article/pii/S1364815211001988*.

[9] Ims, R.A., Jepsen, J.U., Stien, A. & Yoccoz, N.G. 2013, *Science plan for COAT: Climate-ecological Observatory for Arctic Tundra. Fram Centre Report Series 1*, 2013, in *Fram Centre, Norway, 177 pages. http://www.framsenteret.no/getfile.php/2435814.1574.xyxruwywpp/FinalPDF_COAT.pdf*.

[10] Åshild Ø. Pedersen, A. Stien, E. Soininen, and R. A. Ims, *Climate-ecological observatory for arctic tundra-status 2016*, Mars 2016, in *Fram Forum 2016, pages 36-43.*

[11] S. Hamel, S. T. Killengreen, J.-A. Henden, N. E. Eide, L. Roed-Eriksen, R. A. Ims, and N. G. Yoccoz, *"Towards good practice guidance in using camera-traps in ecology: influence of sampling design on validity of ecological inferences*, 2013, in *Methods in Ecology and Evolution, vol. 4, no. 2. pp. 105-113*

[12] Gună, Ştefan and Mottola, Luca and Picco, Gian Pietro, *DICE: Monitoring Global Invariants with Wireless Sensor Networks*, June 2014, in *ACM, vol. 10, no. 4. pp. 54:1–54:34 http://doi.acm.org/10.1145/2509434*.