TALLINN UNIVERSITY OF TECHNOLOGY
SCHOOL OF ENGINEERING
Department of Electrical Power Engineering and Mechatronics

HYPERSPECTRAL IMAGING WITH JETSON TX2

HÜPERSPEKTRAAL-PILDITEHNIKA JETSON TX2 ABIL

MASTER THESIS

Student: Jalal Sadigli
Student code: 184359MAHM

Supervisor: Dhanushka Chamara Liyanage, Engineer
Co-supervisor: Mart Tamre ,professor

Tallinn 2021

AUTHOR'S DECLARATION

Hereby I declare that I have written this thesis independently.
No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"21" December 2020

Author: signed digitally

Thesis is in accordance with terms and requirements

"....." ..... 20….

Supervisor: ….....
/signature/

Accepted for defence

"....."..... 20… .

Chairman of theses defence commission: .....
/name and signature/

Non-exclusive Licence for Publication and Reproduction of Graduation Thesis[1]

I, Jalal Sadigli (date of birth: 31.08.1996) hereby

1. grant Tallinn University of Technology (TalTech) a non-exclusive license for my thesis HYPERSPECTRAL IMAGING WITH JETSON TX2, supervised by Dhanushka Chamara Liyanage,

1.1 reproduced for the purposes of preservation and electronic publication, incl. to be entered in the digital collection of TalTech library until expiry of the term of copyright;

1.2 published via the web of TalTech, incl. to be entered in the digital collection of TalTech library until expiry of the term of copyright.

1.3 I am aware that the author also retains the rights specified in clause 1 of this license.

2. I confirm that granting the non-exclusive license does not infringe third persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

---

[1] *Non-exclusive Licence for Publication and Reproduction of Graduation Thesis is not valid during the validity period of restriction on access, except the university`s right to reproduce the thesis only for preservation purposes.*

Signed digitally

21/12/2020

Department of Electrical Power Engineering and Mechatronics

THESIS TASK

Student: Jalal Sadigli,184359MAHM

Study programme,

main speciality: MAHM02/18 - Mechatronics

Supervisor(s): Engineer, Dhanushka Chamara Liyanage

Co-supervisor: Professor, Mart Tamre

Thesis topic:

(in English) *HYPERSPECTRAL IMAGING WITH JETSON TX2*

(in Estonian) HÜPERSPEKTRAAL-PILDITEHNIKA JETSON TX2 ABIL

Thesis main objectives:

1. Improve previously developed image acquisition method
2. Apply push broom scan method by controlling LTS300/M from NVIDIA Jetson TX2 board
3. Apply Kmeans and MiniBatchKmeans algorithm for clustering

Thesis tasks and time schedule:

No	Task description	Deadline
1.	Review previous work	15.10.2020
2.	Study of hyperspectral imaging techniques, principles and features of the hyperspectral camera, NVIDIA Jetson TX2, LTS300/M	30.10.2020
3.	Generating hyperspectral data cube using push broom scanner method with the help of the LTS300/M	20.11.2020
4.	Applying unsupervised machine learning algorithms	01.12.2020
5.	Testing the system	21.12.2020

Language: English **Deadline for submission of thesis:** "21" December 2020

Student: Jalal Sadigli signed digitally "21" December 2020

Supervisor: …… …….. ".....".....20….a

/signature/

Head of study programme: …… ..... ".....".....20…..a

/signature/

Terms of thesis closed defence and/or restricted access conditions to be formulated on the reverse side

CONTENTS

PREFACE

Department of Electrical and Power Engineering and Mechatronics of Tallinn University of Technology provided the thesis. Major thesis work was done in the laboratory (NRG-202) of the same department. After some discussions the topic was offered by co-supervisor professor Mart Tamre.

Hyperspectral imaging is a spectroscopy-based analytical technique. For the same spatial area, it collects hundreds of images at various wavelengths. Recent advances in sensor design resulted in light weight cameras such as XIMEA xIQMQ022HG-IM-LS150-VISNIR that is 32 gram [1] which makes it possible to use camera for remote sensing applications. Hyperspectral cameras produce images which have rich information content, in order to gather and process that data , powerful yet small computer is needed. Jetson TX2 board is a one of the embedded computers that NVIDIA produces. Besides portability, the board provide a large processing capability and having onboard Graphics Processing Unit (GPU) allow the execution of instruction in parallel. With this powerful embedded computer, real-time hyperspectral image processing application was developed.

Keywords: hyperspectral imaging, real-time imaging setup, nvidia jetson tx2, on-board hyperspectral processing, master thesis.

LIST OF ABBREVIATIONS AND SYMBOLS

API – Application Programming Interface

CMOS – Complementary Metal Oxide Semiconductor

CPU – Central Processing Unit

CUDA - Compute Unified Device Architecture

FPGA - Field Programmable Gate Array

FPS – Frame per Second

GPU – Graphical Processing Unit

HSI – Hyperspectral Imaging

IDE – Integrated Development Environment

LPDDR – Low Power Double Data Rate

OS – Operating System

SDK – Software Development Kit

UAV – Unmanned Aerial Vehicle

USB – Universal Serial Bus

VM – Virtual machine

WSL Windows Subsystem for Linux

TABLES

1 INTRODUCTION

1.1 Overview

Hyperspectral imaging is a spectroscopy-based analytical technique. For the same spatial area, it collects hundreds of images at various wavelengths. In contrast to human eye which has three color receptors in the red, green and blue, hyperspectral imaging check the continuous light spectrum with fine wavelength resolution for each pixel of the object, not only in the visible but also in the near infrared. The recorded information forms a so-called hyperspectral cube in which the spatial extent of the object is represented by two dimensions and the spectral data by the third[2].
Every material has a particular spectral signature that can be used for its specific identification as a fingerprint. Therefore, due to its non-destructive, standoff, and label-free capability in recognizing the components of material, hyperspectral imaging finds broad variety of applications in remote sensing[2]. In various fields, such as astronomy, agriculture, molecular biology, biomedical imaging, climate and surveillance, hyperspectral imaging is used.

Recent advances in sensor design resulted in light weight cameras such as XIMEA xIQMQ022HG-IM-LS150-VISNIR that is 32 gram [1] which makes it possible to mount the camera on Unmanned Aerial Vehicle (UAV). Having the camera on the UAV will let us to do remote object detection and classification way easier.

Hyperspectral cameras produce images which have rich information content, in order to gather and process that data , powerful yet small computer is needed. The NVIDIA company has made great progress for the last few years. Jetson TX2 board is a one of the embedded computers that NVIDIA produces. Besides portability, the board provide a large processing capability and having onboard Graphics Processing Unit (GPU) allow the execution of instruction in parallel[3][4].

1.2 Motivation

Hyperspectral imaging is an efficient and flexible instrument that can be used in a wide range of practical application[5] such as solving problem in food analysis, precision agriculture and others. Growing interest to the remote sensing, self-driving car makes companies to develop hardware and/or software platforms to let computer vision reachable to every developer.

Being around self-driving car, delivery robots in the campus of the university gave me great interest in computer vision and machine learning. Working in those fields motivates me to write this thesis.

1.3 Aim

The main goal for this thesis is getting data from the hyperspectral camera with the help of GPU based embedded computer in real time to generate hyperspectral cube in order to do real-time processing on the same single board computer.

1.4 Tasks

- Review precious works
- Setup, flash, configure the system
- Read documentation for the camera and its Application Programming Interface (API)
- Control LTS300/M to apply push broom scanning
- Generate hyperspectral cube
- Check and verify the generated data cube
- Apply processing/Machine Learning techniques
- Test the system

1.5 Structure of the thesis

The thesis consists of four main parts. Introduction, literature review, development and testing. In the introduction part, main idea is presented. It also includes motivation of the author for the topic. Task definition and goal are clearly written as well as structure of the thesis.

The literature review part contains theoretical background for hyperspectral imaging, discuss research areas where hyperspectral imaging is used. Different methods for generating hyperspectral data cube are described in this chapter. Second chapter is about specifications and implementation of hardware and software components in different research works. The same chapter also includes information about unsupervised machine learning algorithms. In software and hardware section of the second chapter, general overview of the system and its components are explained.

Third chapter is mainly for the development. It contains description of the system and development from unboxing the components to assembling fully functional system.
At the beginning of the chapter, whole overview of the implemented components hardware and software are represented. Then, usage of hardware and software components separately is explained in detail. After that, main part of the software implementation is described.
The fourth chapter is showing the result of testing the developed system.
The fifth and sixth chapters give short summary in English and Estonian respectively.
References and appendix are after summary chapters.

2 LITERATURE REVIEW AND COMPONENTS OF THE SYSTEM

2.1 Literature review

Human eyes can perceive specific wavelengths from 380nm to 780nm which is called visible spectrum. But in the electromagnetic spectrum there are a lot of wavelengths out of visible spectrum which are invisible to human beings.

Developments in imaging technology, we can gather data for different wavelengths out of visible spectrum and process it in a useful way. Hyperspectral imaging(HSI) is the simultaneous processing and combination of spatial and corresponding spectral information in an image space.

In the electromagnetic spectrum, hyperspectral sensors capture hundreds of images and each individual pixel in the observed spectrum is a complete spectrum. HSI is thus a three-dimensional data cube with two dimensions of spatial and a spectral dimension. The spectral dimension enables the materials in the scene to be identified[6].

HSI allows us to discover the unexplored by visualizing information which is not visible to human eye. Images captured from HSI camera are in the form of a hypercube which is consist of n different band of same object.

There are three main techniques used for generating three-dimensional hypercube[7]. First one is spectral scanning which is sequential image of full spatial information. In spectral scanning, output of each 2-D sensor represents single-colored, spatial map of the object. Spectral scanning HSI systems are usually based on optical band-pass filters which can be tunable or fixed. By changing one filter after another, the object is spectrally scanned whilst the platform remains stationary[2][3][4].

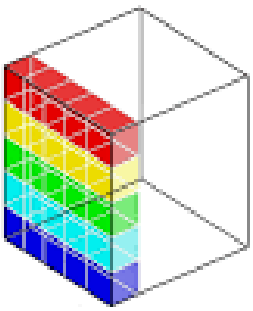

Figure 1 – Spectral scanning[7].

Second one is spatial scanning which is sequential image of common spectral data. In this method output of each 2-D dimensional sensor represents a full narrow line of spectrum. Spatial scanning HSI devices generate image of the object stacking lines together with the help of moving platform(push broom scan). Spatial scanning systems are very common in remote sensing[9], [10].

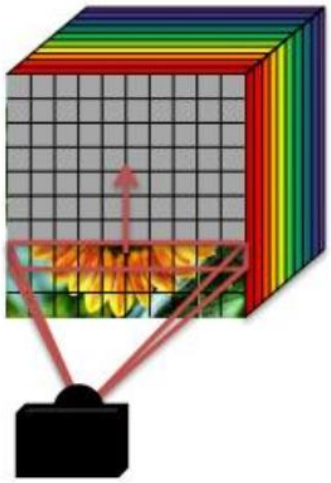

Figure 2 – Spatial scanning[5].

Third one is snapshot or non-scanning where all spectral and spatial information are captured simultaneously. In this method output of a single 2-D sensor contains all spectral and spatial information. Snapshot HSI devices generate full hypercube at once , without doing any scanning[10], [11].

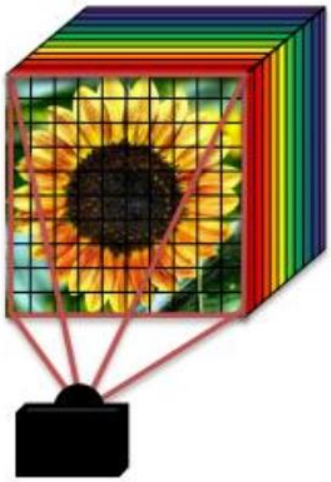

Figure 3 – Snapshot [5].

Qualitative results can be obtained by using first and second methods, but they are not efficient for collecting and processing the entire hypercube[5].

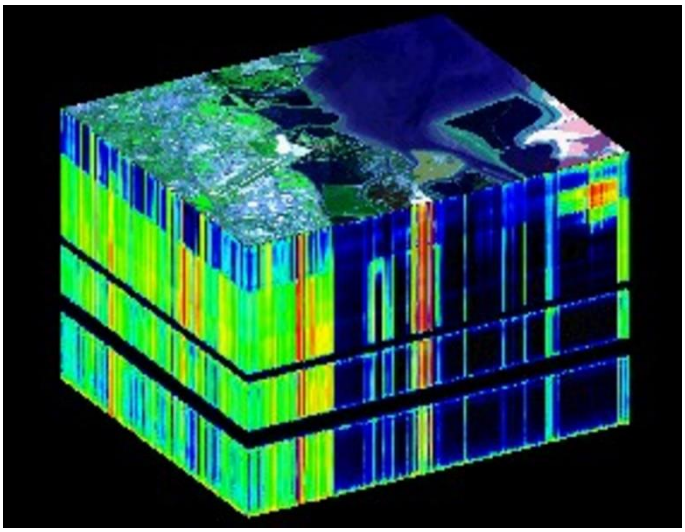

Figure 4 – Hyperspectral cube[12].

In this thesis push broom method was used.

HSI is successfully employed in different industries. The analysis of the generated hypercube allows for a detailed classification by the fingerprints of the object[13][14][15][16][17][18][19]:

- Precision agriculture;
- Environment monitoring;
- Food sorting;
- Robotic vision;
- Blood/ urine analyzers;
- DNA sequencer;
- Print quality inspection;
- Night vision systems;
- Industrial gas leaks monitoring;
- Wounds imaging;
- Water monitoring analyzers;

2.2 Influence of illuminations

To find best source of illumination for hyperspectral image acquisition, previous master's degree students have made tests. As a source, students compared sun, fluorescent, light-emitting diodes (LED), and incandescent. For HSI, Resonon Pika II which is in the range of 400-1000nm and Resonon Near Infrared camera which is in the range of 1000-1700nm were used. Following table shows result of their work[20].

Sources	380-700nm	400-1000nm
Sun	+	+
Incandescent	+	+
Fluorescent	-	-
LED	+	-

Table 1 – Result of the previous research[20].

Student concluded their work as incandescent light source is the best source for HSI[20].

2.3 Processing hyperspectral data in real time

There are many significant applications for remotely sensed hyperspectral images since its high-spectral resolution allows for more precise object detection and classification. Real-time on-board implementation is strongly needed to enable immediate decision-making in critical situations. As mentioned before, the hyperspectral camera collects hundreds of images for the region of interest for different wavelengths. High spectral resolution of hyperspectral cameras allows more efficient diagnostic capabilities than conventional imaging in detection, and classification. In order to support quick decision-making, real-time processing and analysis are needed to provide instant results. It is extremely beneficial to be able to perform onboard processing. In image classification and recognition, machine learning algorithms can realize high precision of classification[21]. The enhanced spatial, spectral, and temporal resolutions given by hyperspectral cameras require fast computing algorithms that can speed up the effective use of hyperspectral data. To accelerate remote sensing techniques, methods rely on dedicated hardware such as clusters, distributed systems, and specialized devices such as GPUa or FPGAs(field programmable gate array) have been popularly used[22].

2.3.1 Anomaly detection

Anomaly detection is the process of identifying unexpected items or events that differ from the norm in data sets. And unlabeled data, known as unsupervised anomaly detection, is often used for anomaly detection. Detection of anomalies has two basic assumptions:

- Anomalies only occur quite infrequently in the given data.
- Features of the anomalies differ significantly from ordinary cases[23].

In other words, anomaly detection is a technique of unsupervised target detection in which no prior information available for the target or the background. Anomaly Detection plays significant role in hyperspectral detection[24]. In practical applications, real-time anomaly detection is more useful, especially the detection of moving targets or instantaneous targets.

RX detector is one of the common algorithms for anomaly detection. To detect abnormal data, RX detector implements the Mahalanobis distance between a test pixel and its background metrics[24].

When the samples in the image do not conform to a linear distribution, RX detector often fail to detect anomalous data[25]. To solve this problem new version of RX algorithm was proposed which is called kernel RX. Kernel RX is a nonlinear version of previous algorithm[25]. However , RX detector can easily designed into real-time framework[24]. At the same time, due to computational complexity of the Kernel RX, it can not be implemented in real-time. To make efficient processing in real-time equation of the kernel RX is rewritten to support new proposed algorithm which is called casual kernel RX detector[25]. With very clear boundaries, both algorithms can give bright results, while RX detector alone itself generates very dark results with fuzzified boundaries. Same results were produced by casual kernel RX and kernel RX algorithm, but computing times are different[25].

2.3.2 Spectral unmixing

One of the most popular methods to process hyperspectral images is spectral unmixing[26].

Due to lower spatial resolution of remote sensing spectroscopy, hyperspectral remote sensing technology has a strong capacity for ground object detection. A single pixel that leads to a remote sensing hyperspectral image generally contains more than one type of feature coverage, resulting in a mixed pixel. The existence of a mixed pixel influences the accuracy of the identification and classification of the ground object and hinders the use and development of hyperspectral technology[26]. Two mixture models are available. Linear and nonlinear mixture model[27]. Each pixel in the linear spectral mixing model can be expressed as a linear end-member combination weighted by its corresponding abundance[26]. For multicore implementation following algorithms can be implemented[27]:

- Parallel Virtual Dimensionality.
- Parallel k-means.

- Parallel spatial-spectral preprocessing.
- Parallel N-FINDR .
- Parallel least squares.

Different methods such as nonnegative matrix factorization, Bayesian method, sparse method were also proposed[26].

2.3.3 FPGA based systems

FPGAs are electronic component based around a matrix of programmable interconnected configurable logic blocks. After production, FPGAs can be reprogrammed to the desired application or functionality configurations. This feature distinguishes FPGAs from custom manufactured Application Specific Integrated Circuits for design tasks[28]. The mentioned feature gives makes development time for hardware and software systems shorter compared to application specific integrated circuits. FPGAs can provide better performance closer to GPUs[22]. FPGAs have the potential to increase computational bandwidth due to their parallel nature in systems where software performs processing functions on huge data[29].

Automatic target generation process is a widely used algorithm for target detection mostly because of its aggressive performance. However, it has huge computation issue due to inversion and multiplication of growing matrices. To handle this problem new fast implementation of the algorithm was proposed based on FPGA[30].

As mentioned before, most widely used tool for analyzing hyperspectral images is spectral unmixing. A new algorithm named FUN was proposed for real time processing with spectral unmixing[31]. The implementation of this algorithm was done on FPGA system[32].

With custom hardware architecture based on FPGA can get result of up to 96% quality detection compared to a CPU based system[33].

2.3.4 GPU based systems

A GPU is a computer unit which performs fast mathematical calculations, mainly for image rendering. GPUs were designed primarily to speed up process relateed to image[34]. Parallel implementation of noise adaptive principal components algorithm for feature extraction of hyperspectral image based on CPU-GPU collaboration is flexible and efficient[35].

One of the most important topic with hyperscpetral images is spatial-spectral classification. Parallel implementation of this classifier has advantages of spatial piecewise smoothness and correlation of neighboring pixel[36].

Hundreds of narrow bands of the HSI sensor are used to capture accurate spectral responses from objects. Band selection is popular for reducing dimensions of hyperspectral data. The goal of band selection is to pick a small subset of hyperspectral bands in order to minimize spectral redundancy and reduce computational costs while reserving valuable spectral information for the scanned object[37]. Implemetation of band selection on GPU is more efficient compared to CPU[38].

Parallel application of HSI over the serial application accompilishes considerable improvement. However, there are still many enhancements required for real-time of parallel implementation. The main point for this is because of large hyperspectral cube with rich information which is needed to be processed in real time. One option to achieve real-time implementation of applications isto use GPU.

2.4 Unsupervised classification

The unsupervised classification is the method of clustering without any prior information. In hyperspectral image case the unsupervised method refers to spectral similarities of the hyperspectral data. Assessments of relative pixel positions in the image in an unsupervised classification helps to search for clusters within the data. It is expected that each cluster represents unique properties[39].

2.4.1 KMeans clustering

The K-means clustering method's classification principle is that the sum of squared distances from all the pixels in each cluster to the center point of that cluster is the smallest. The center point of the initial clustering is randomly chosen at the beginning of the clustering. To complete the initial clustering, other pixels to be clustered are classified into one of the categories according to the initially specified principles. Then adjust each class's clustering center point, change the center point of the clustering, and classify again. Iterate this until position of the points of the clustering center no longer varies. Before stopping iteration, make sure that best clustering center is found. Then stop the iteration get the best result for clustering. The number of selected categories cannot be changed during clustering with K-Means. Changing initial clustering center point position is also affect the clustering result[40].

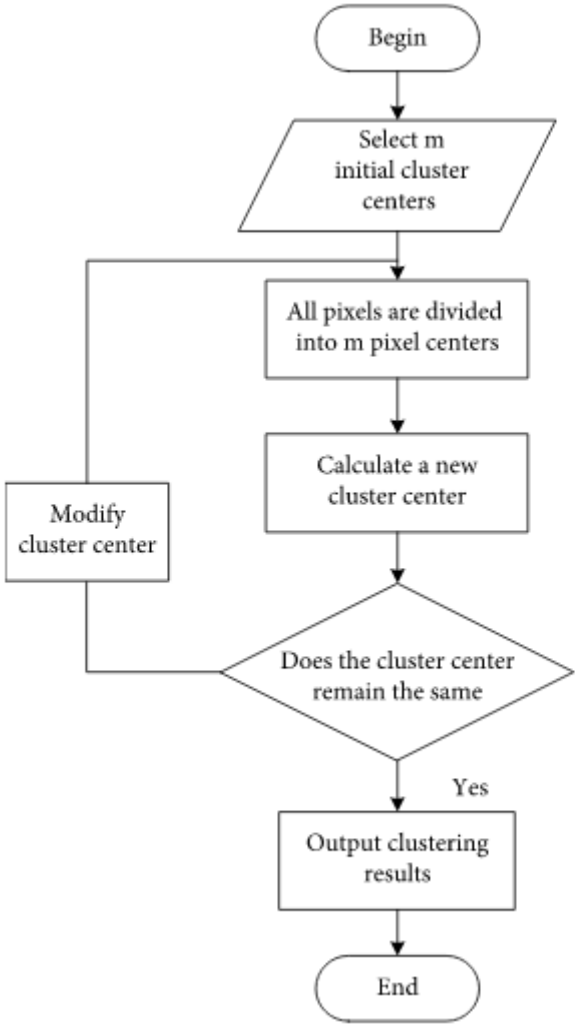

Figure 5 – Flow chart for KMeans algorithm[40].

2.4.2 Mini Batch KMeans clustering

K-means is one of the most common algorithms for clustering, mainly because of its good performance over time. Because of its disadvantage of having the entire dataset in main memory, the processing time of K-means increases with the growing size of the datasets being examined. For this purpose, several methods for reducing the algorithm's temporal and spatial cost have been suggested. The Mini batch K-means algorithm is one of them. The key idea of the Mini Batch K-means algorithm is to use small random data batches of a fixed size so that they can be stored in memory. A new random sample is collected from the dataset for each iteration and used to update the clusters, and this is repeated until convergence.

The pseudo code for Mini Batch KMeans classification is shown below.

```
Given a dataset D = {d1, d2, d3, .....dn},
                        no. of iterations t,
                        batch size b,
                        no. of clusters k.

k clusters C = {c1, c2, c3, ......ck}

initialize k cluster centers O = {o1, o2, .......ok}
# _initialize each cluster
Ci = Φ (1=< i =< k)
# _initialize no. of data in each cluster
Nci = 0 (1=< i =< k)

for j=1 to t do:
    # M is the batch dataset and xm
    # is the sample randomly chosen from D
    M = {xm | 1 =< m =< b}

    # catch cluster center for each
    # sample in the batch data set
    for m=1 to b do:
        oi(xm) = sum(xm)/|c|i (xm ε M and xm ε ci)
    end for
    # update the cluster center with each batch set

     for m=1 to b do:
        # get the cluster center for xm
        oi = oi(xm)
        # update number of data for each cluster center
        Nci = Nci + 1
        #calculate learning rate for each cluster center
        lr=1/Nci
        # take gradient step to update cluster center
        oi = (1-lr)oi + lr*xm
     end for
end for
```

Figure 6 – Pseudo code for Mini Batch KMeans algorithm[41].

MiniBatchKMeans converges more rapidly than KMeans, but there is a decrease in the consistency of the results. In practice, this quality difference, as seen in the figure below, may be very small.

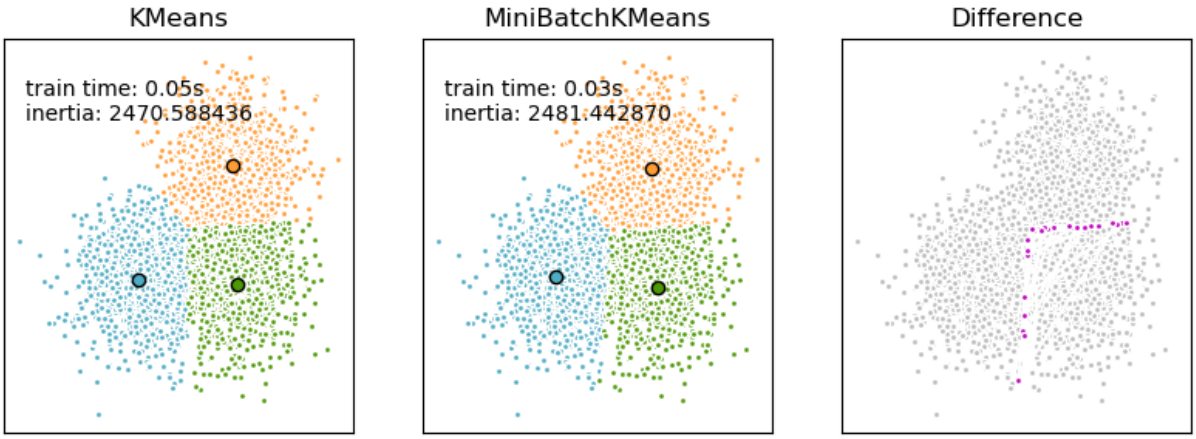

Figure 7 – Kmeans vs MiniBatchKMeans.(train time is the time the algorithm spent on separating image into clusters based on perceived similarities, inertia is a measure how internally coherent clusters are.)[42].

In this thesis Mini Batch KMeans algorithm was used for online clustering and KMeans algorithm for offline clustering. The results were compared at the end.

2.5 Hardware

2.5.1 Overview

Embedded computer, NVIDIA Jetson TX2, is used as main processing unit. NVIDIA TX2 board acquire images through the XIMEA xIQ hyperspectral camera. With the help of Thorlabs LTS300/M motion stage spatial scanning can be implemented. Mini Universal Serial Bus (USB) is used to control motion stage with special commands. The hyperspectral camera is mounted on the motion stage and connected to NVIDIA board through USB3.0.

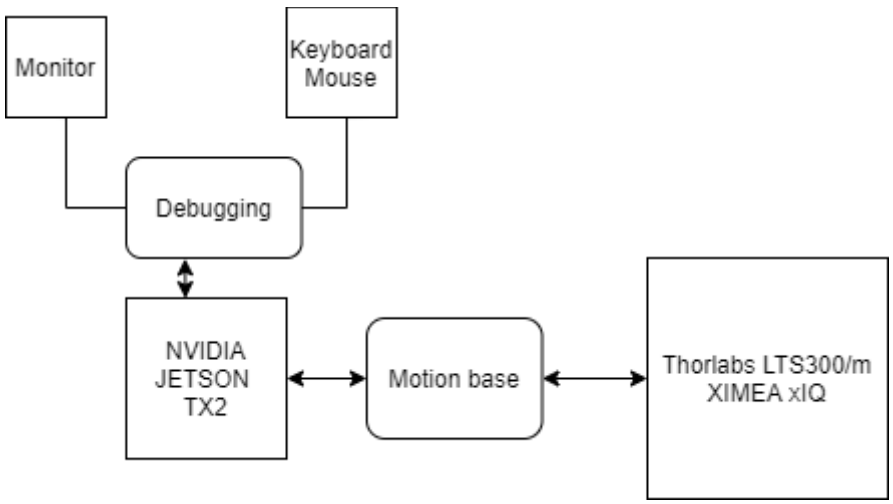

Figure 8 – General overview of the system

2.5.2 NVIDIA TX2

As a result of constant progress in the development of embedded computers, the NVIDIA company produced Jetson series[43]. Jetson TX2 is one of the fastest, most power efficient embedded computer[44].

Single chip system used in the Jetson TX2 which includes 2 64-bit Central Processing Unit (CPU) cores Denver 2, 4 CPU ARM Cortex-A57 cores and 256 Compute Unified Device Architecture (CUDA) cores on the pascal architecture. Encoding and decoding of 4K x 2K video with frame rate of 60 frame per second (fps) is supported. 8 GB Low-Power Double Data Rate (LPDDR) memory with a bandwith of 58.4GBps and 32GB eMMC are included. For the connectivity, the module has Gigabit Ethernet port, Wi-Fi and Bluetooth. Assembled dimension of the board is 50mm x 80mm[3][9][26].

Figure 9 – NVIDIA TX2 developer kit without accessories[45].

In addition to their portability, these boards provide a large processing capacity and allow the execution of instructions in parallel as they have Graphics Processing Units(GPU) onboard[4]. Jetson platforms are compatible with the JetPack Software Development Kit(SDK), SDK has libraries for deep learning, computer vision and

accelerated computing[46]. Because of that, Jetsons are being used in various applications, including embedded vision systems[47], [48], autonomous vehicles[49], medical systems[50].

The emerging low-power multi-threaded architectures from ARM and NVIDIA can be a practical approach for hyperspectral image processing as compared to a standard high-performance multi core processor. The experimental results from earlier studies showed that low power GPUs deliver reasonable performance and high power/energy grains[3].

2.5.3 Hyperspectral camera – XIMEA

Ximea xIQ MQ022HG-IM-LS150 VISNIR is an USB3.0 hyperspectral camera with dimensions of 26 x 26 x 31mm and weight of 32g.It has 2.2 megapixel sensor which can shoot 90fps. It can scan up to 850 lines per second. The sensor technology that is used in the camera is based on standard Complementary Metal Oxide Semiconductor (CMOS) with a native resolution of 2048 x1088 pixels. It can capture spectral range between 470-900 nm with spatial resolution of 2048x5 lines in 3nm steps[51]. The camera can be powered directly from USB due to its low power consumption which makes it ideal for embedded vision systems. It can be used in applications from remote sensing to optical sorting[52].

Figure 10 - Ximea xIQ MQ022HG-IM-LS150 VISNIR[1].

EDMUND OPTICS C series lens with 35mm focal length and aperture of F1.65 is installed on the camera[53].

2.5.4 Thorlabs

For the push broom scan or in other name spatial scanning needs moving platform with respect to object. With integrated stepper motor controller , Thorlabs LTS300/M which is optimized for applications requiring high accuracy is great for this thesis[54].

Figure 11 – Thorlabs LTS300/M [54].

The stage has accuracy of 5.0 um It can be controlled via manual keypad or remote pc over serial bus[54].

Communications parameters are fixed at:

- 115200 bits/sec;
- 8 data bits, 1 stop bit;
- No parity ;
- No handshake.

The communications protocol used in the Thorlabs controllers is based on the message structure that always starts with a fixed length, 6-byte message header which, in some cases, is followed by a variable length data packet. For simple commands, the 6-byte message header is enough to convey the entire command. For more complex commands, for example, when a set of parameters needs to be passed on, the 6 byte header is not enough and in this case the header is followed by the data packet.

The header part of the message always contains information that indicates whether a data packet follows the header and if so, the number of bytes that the data packet contains. In this way the receiving process is able to keep tracks of the beginning and the end of messages[55].

2.6 Software

2.6.1 Jetsonpack

The jetson development pack for linux for tegra is an installer that automates installing and setting up a development environment required to develop for the NVIDIA jetson embedded platform, including flashing the board with the latest available Operating system (OS) image. Jetpack includes host and target tools, APIs, packages[56].

Inside Jetpack Software development kit (SDK) there are components for Deep learning (TensorRT[57], cuDNN[58]), computer vision(VisionWorks[59], OpenCV[60]), accelerated computing(cuBLAS[61], cuFFT[62]), graphics(Vulkan[63], OpenGL[64]), Multimedia(libargus[65]) and many more.

Those components are widely used by researchers. Deep learning application based on embedded GPU[66], accelerating deep learning frameworks with micro-batches[67], GPU-SFFT[68] are some example for recent works.

2.6.2 Python

Python is the best choice as compared to other programming languages in many area and applications[69]. Image processing[70][71][72] is one of them. Besides being an extremely powerful programming language , another reason for choosing python as programming tool is the camera. Ximea company has API for python and C++[73]. Also, python has numpy[74] package that can handle n dimensional arrays which helps while generating hypercube.

3 DEVELOPMENT

This chapter is about overall system in detail.

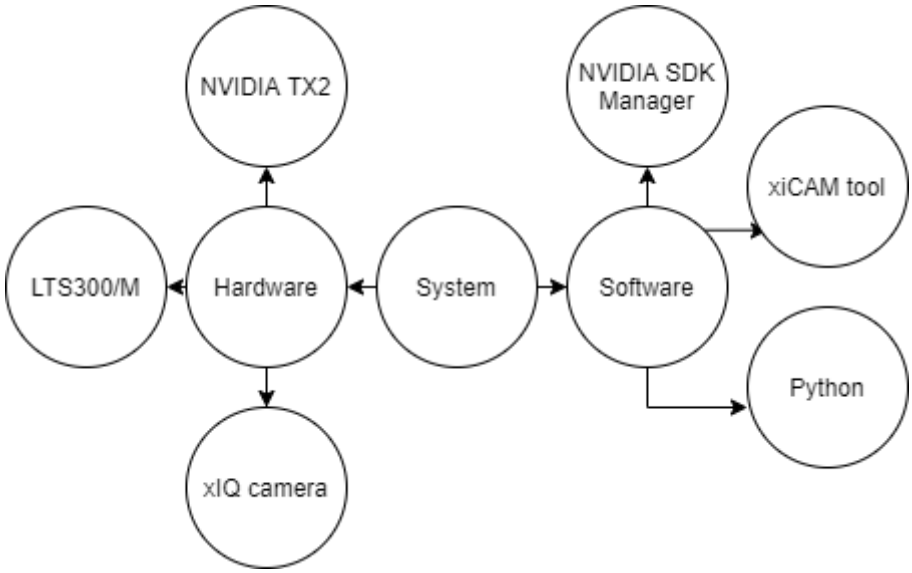

Figure 12 – overview of the system.

3.1 Hardware Subsystem

This section is about how hardware subsystem was developed. The subsystem consists following components.

- Embedded board – NVIDIA Jetson TX2
- Motion stage – Thorlabs LS300/M
- Hyperspectral camera - Ximea xIQ MQ022HG-IM-LS150 VISNIR
- Debugging interfaces – keyboard/mouse/monitor
- Other accessories – USB hub, power supply, ethernet, etc.

3.1.1 NVIDIA Jetson TX2

In order to start working, Jetson TX2 board must be reflashed with its own Linux OS, because password for the board was not found. To flash the board host machine with Linux OS is needed. There were 3 methods available. First method is using host machine with windows OS which has Ubuntu on VirtualBox[75]. The problem with this method is while flashing the board, it keeps restarting itself which is a problem for VirtualBox

Machine(VM) to enumerate USB after restart. Second way to flash the board is using Windows Subsystem for Linux(WSL)[76]. It is a subsystem developed for windows to run Linux on it but at the time of starting this thesis , the WSL system was not developed very well, and it was not a reliable system. Third and used method is installing Ubuntu on a host machine. At the time of starting this work, the board was recommended to be flashed by Ubuntu 16.04 or Ubuntu 18.04.

After installing OS on the host machine, NVIDIA SDK Manager was installed to the host machine. SDK manager helps to install everything required for the board. Defining used board in the SDK manager is enough to get all packages ready for flashing the board.

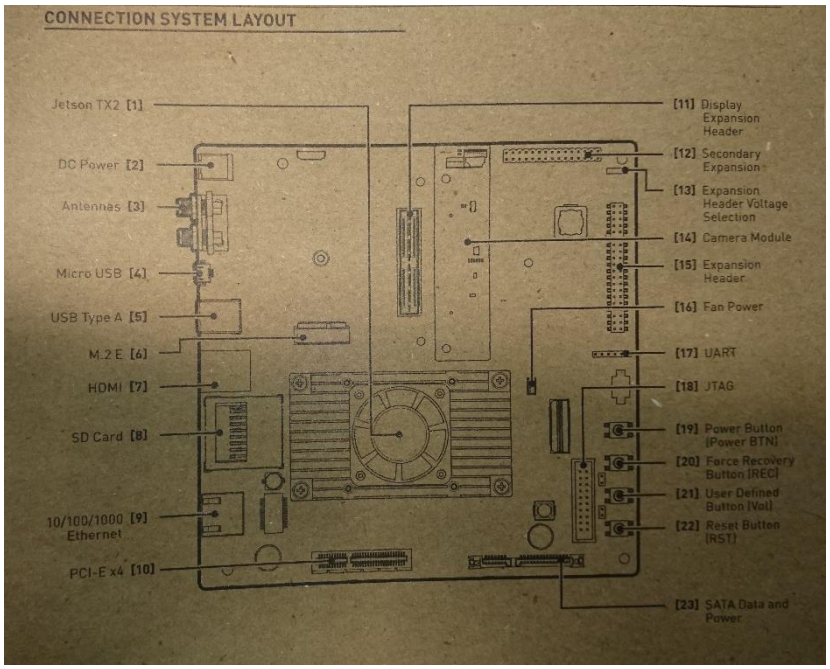

Figure 13 – Nvidia TX2 developer kit connection layout.

To make the board ready for flashing, board has to be in Force USB Recovery mode. To make it so:

1. The device must be shut down, not in a suspended or sleep mode
2. Connect the Micro-B plug on the USB cable to the USB Micro-B port on the device(4) and the other end to an available USB port on the host machine
3. Connect power adapter to the device(2)
4. While the system powered off, press and hold the Recovery Force button(20), press and release the Power button(19), press and release the Reset button(22), wait for 2 seconds and release the recovery force button.

When the board is in the Force USB Recovery Mode , then software components can be flashed with the help of the NVIDIA SDK Manager.

3.1.2 Thorlabs LTS300/M

The travel stage for the moving the hyperspectral camera to scan objects requires 24V DC voltage source which was provided with AC to DC adapter. The USB port on the stage was used for communication between the stage and the board.

Figure 14 – LTS300/M wiring. Power supply(left), USB(right).

3.1.3 The hyperspectral camera

The hyperspectral camera was mounted on the stage with the help of 3D printed right angle mounting bracket. The camera gets it power from USB port which is also used for communication to the board.

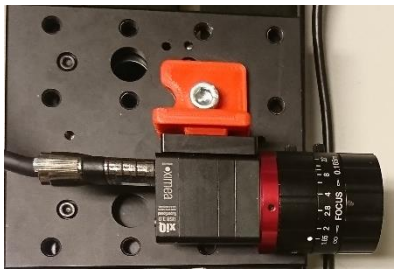

Figure 15 – The camera installed on the motion stage.

3.1.4 USB hub

As there is only one USB Type-A port on the board, and there are at least 2 devices that need communication over USB in the system, a USB HUB was used. The USB Hub must be USB3.0 compatible, because the USB port on the camera is USB3.0

3.1.5 Debugging

As for all hardware system, a debugging interface is a must to have in the system. For that purpose, a separate monitor/keyboard/mouse was connected to the board and also internet over ethernet cable was provided to have SSL(Secure Socket Layer) connection between the host and the board

3.2 Software subsystem

This section is about how software subsystem was developed. The subsystem consist following components.

- Board flashing tool - NVIDIA SDK Manager
- Programming language - Python
- Debugging - xiCamTool

3.2.1 NVIDIA SDK Manager

NVIDIA SDK Manager was used on the host machine to download and prepare all necessary packages for flashing the board

3.2.2 Python

Python 2.7.17 was installed with the SDK manager and used with this version without updating it. No IDE(Integrated Development Environment) was used. All code was written in the default text editor of Ubuntu.

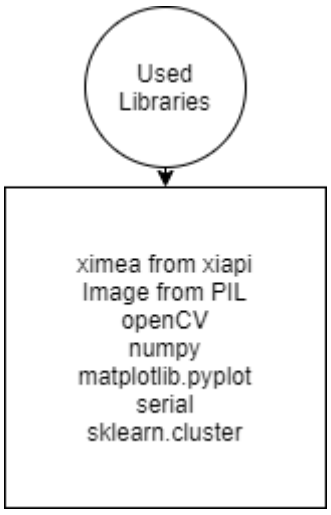

Figure 16 - Imported libraries.

3.2.3 xiCamTool

To configure and test the camera, and check captured images, xiCamTool[77] was used. It was installed with Ximea SDK for Ubuntu. The Ximea SDK also has all necessary files for python API.

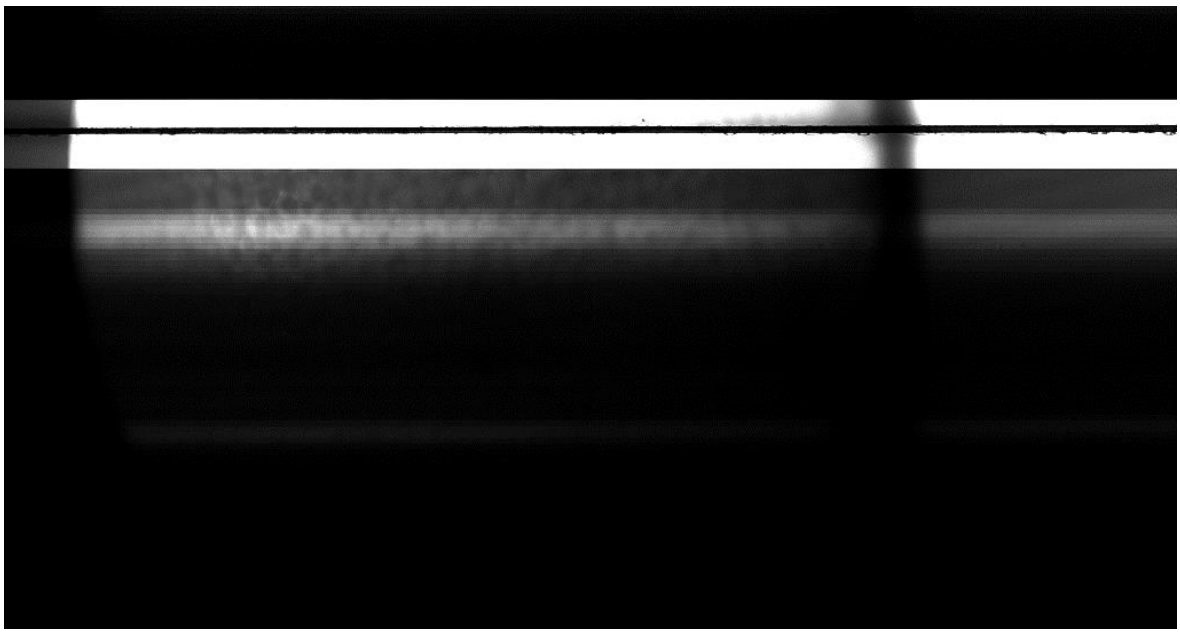

Figure 17 – Raw image from camera using xiCam tool.

3.3 Image Acquisition

This section is about how images were captured using the hyperspectral camera with motion stage, and generating hyperspectral cube as a result of applying push broom scan method

3.3.1 The motion stage

The LTS300/M is a linear translation stage that has own integrated controller which means to control the stepper motor inside the stage requires only specific commands. The commands were sent over USB cable. The motion stage is calibrated before starting to work in order to force the controller to correct any mechanical errors present in the system. Calibration files can be found on the website of the manufacturer[78].

The motion stage supports RS-232 and USB communication protocols. The communication protocols are identical, so USB port on the motion stage can be used for both. To get rid of enumeration part of the USB protocol, RS-232 was implemented.

Python has library called serial that can handle RS-232 communication protocol and imported to the main code.

To give python script to access USB ports without having administrator rights, following command line code was used.

-sudo chmod 666 /dev/ttyUSB0

To establish communication to the motion stage, a serial object must be created with following settings.

- 115200 bits/sec;
- 8 data bits, 1 stop bit;
- No parity ;
- No handshake.

In general, two packages were sent to the motion stage.

1. Relative move command
2. Homing command

Relative move command was used to start a relative move with given relative distance. There are two versions of this command: a shorter version and a longer version. Shorter version has only 6-byte header while longer version has 6-byte header plus 6-byte data. Shorter version uses predefined values for relative movement. Longer version uses values that defined in data bytes. Values that are longer than a byte follow the intel little-endian format

0	1	2	3	4	5	6	7	8	9	10	11
Header						Data					
48	04	06	00	0D	01	Chan ident		Relative Distance			

Table 2 – Relative Move command structure [55]

0^{th} and 1^{st} bytes represent code for relative move command(0x0448), 2^{nd} and 3^{rd} bytes shows is there is data bytes or not(0x0006), 4^{th} one is for bitwise or operation between destination address with 0x80, 0x50 is the destination address for Generic USB

hardware unit.5th byte gives information about source. Source is the host machine and 0x01 corresponds to host machine.

6th and 7th bytes are there for channel identification, as our motion controller has one channel then it should be 0100(0x0001), the last for bytes show relative distance that motor will go.

Upon completion of the relative move the controller sends a message that indicate completed movement with following structure

0	1	2	3	4	5
Header					
64	04	Chan ident	00	81	50

Table 3 – Move Completed message structure[55]

Homing command was used to send the stage to home position. It has only one short version. Information in header bytes are enough for this command.

0	1	2	3	4	5
Header					
43	04	Chan ident	00	0D	01

Table 4 – Homing command structure[55]

Channel identification, source and destination address are same with the relative move command, 0x01,0x01,0x0D respectively. 0x0443 is the code for homing command

3.3.2 The hyperspectral camera

The hyperspectral camera uses USB3.0 protocol for communication. USB3.0 implementation on Linux based systems has some problem that cannot allocate enough buffer for USB3.0 compatible devices which in our cases was resulted as not getting all frames that the camera captured. To increase buffer size for the camera, following command line code was used.

sudo tee /sys/module/usbcore/parameters/usbfs_memory_mb >/dev/null <<<0 [79]

API from the XIMEA for python was used to access the hyperspectral camera. The API creates an interface with which the features and all capabilities can be used.

To establish and get data from the camera following steps implemented.

- Imported API to the python code
- Created instance for the connected camera
- Changed basic settings of the camera
- Created instance for storing captured images
- Sent start acquisition command
- Captured images
- Sent stop acquisition command
- Disconnected from the connected camera

3.3.3 Preprocessing captured image

To obtain hyperspectral data, the sensor inside camera is covered with special filters. Each filter is responsible for a range. But entire surface of the sensor is not covered with the filters. Following figure shows the view of the sensor. Active area shows the covered region of sensor[80]

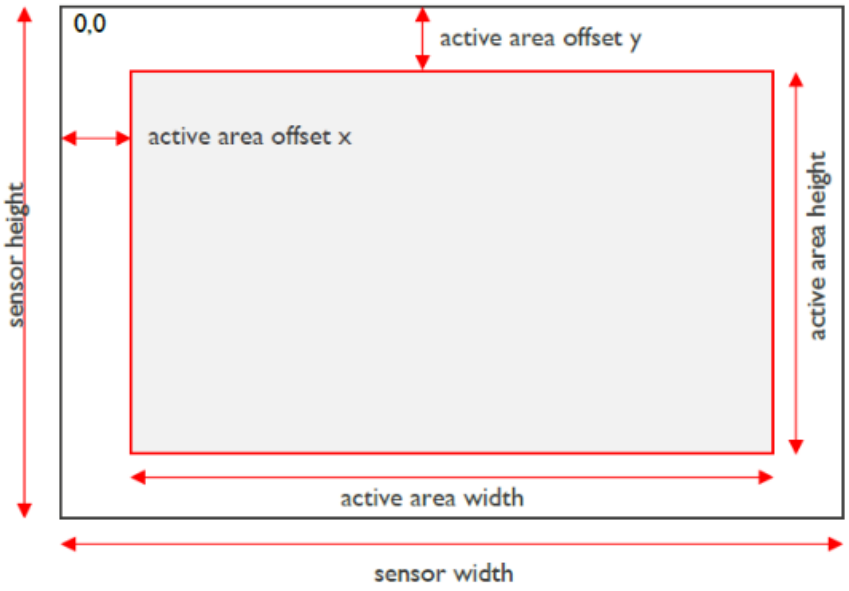

Figure 18 – View of the sensor[80]

Wedge design is implemented on the sensor filter layout. 192 filters in the linescan layout are organized in 192 bands of a fixed height over the fullwidth of the active area. The width of the active area equals to 2040. The height of the active area equals to 192 times 5 pixels(the height of the bands in pixels). Position index 0 indicates the band at the top of the active area. The position index is incremented to the bottom of the active area. Following figure shows wedge layout of the sensor.

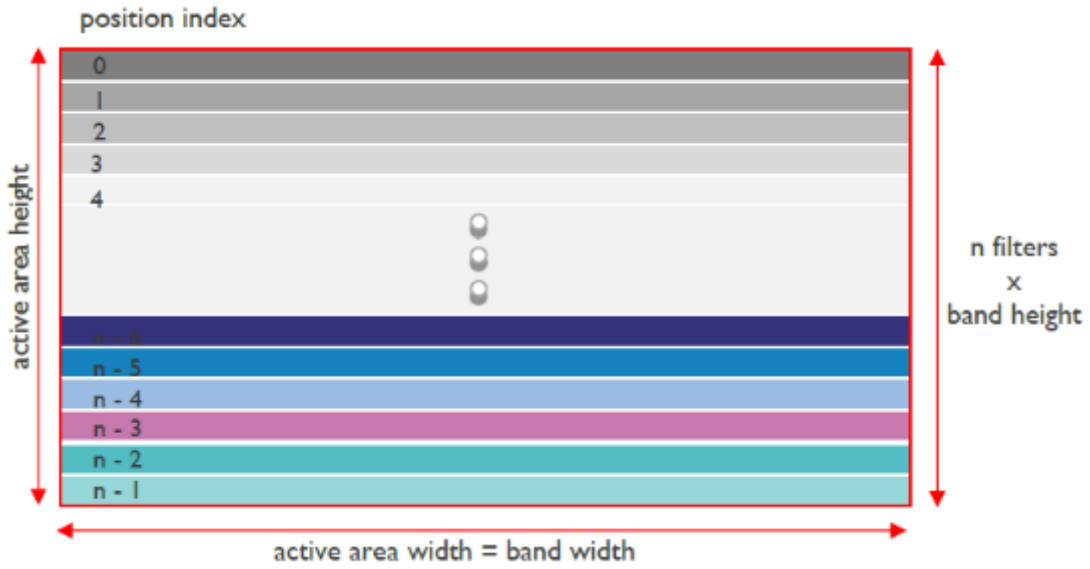

Figure 19 – wedge layout[80]

Two light-sensitive sensors designed for different ranges are composed together in the camera. Between those filters there are empty interface zone of 120 row. Following figure shows positions of the different sensors. After preprocessing active area becomes 960x2040.

0	VIS
1	VIS
	...
62	VIS
63	VIS
empty interface zone	
64	NIR
65	NIR
...	
190	NIR
191	NIR

Figure 20 – Filters and empty interface zone[80]

Figure 21 – empty interface zone and offsets from edges on a raw image[5].

3.3.4 Generating hyperspectral cube

Hyperspectral representation of an object is acquired by the line scan sensor which has conventional sensor with specialized filters on top. Each filter emits only a small portion of the entire spectrum that the object reflects[51]. All this small portion are then combined to create a hyperspectral representation of the object: hyperspectral cube.

In case of the used camera which consists of a sensor with resolution of 2040*960 with 192 filter bands processed. Each band is 5(960/192) pixels in height, and 2040 pixels in width. On the sensor wavelength specific regions are organized in adjacent bands.

Following figure shows generalized view for the sensor

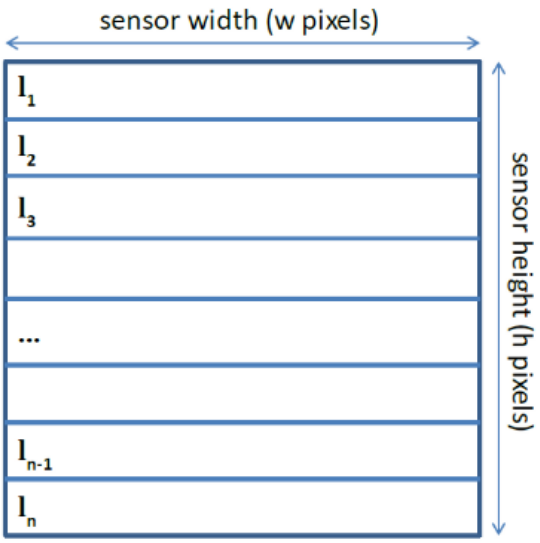

Figure 21 – Generalized view for bands[80].

In order to every point of an object to get captured by the camera, it is needed to make sure that each point passes through each individual band. To obtain whole data about the reflection of light on the object for each band, series of images must be taken.

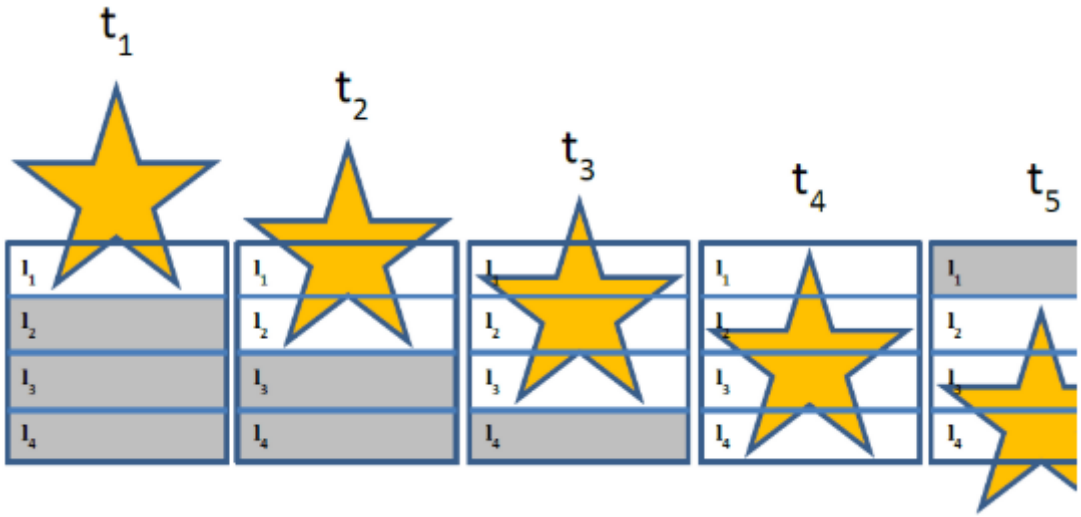

Figure 22 – Full scan of the object[80].

As seen from the figure above, a full scan of the object has three phases which are start-up phase, steady-state phase, and shutdown-phase. t1, t2, t3 phases are corresponding the start-up phase. During start-up phase not all captured data are used. Only the part of the image which has white background is used to generate hyperspectral cube. t4 is the steady state phase in which all captured data are used. Length of the object defines the number of frames in steady state phase. t5 and others corresponds to shut-down phase. As in the start-up phase not all captured data are used.

After shut-down phase, all usable data from capture images are stitched together to construct hyperspectral cube. To get a perfect hyperspectral cube the camera must capture pictures in way that the object positioned perpendicular to the orientation of the sensor. Any misalignment led to incorrectly stitched images.

Following figure shows hyperspectral cube that is generated by capturing and stitching images ideally.

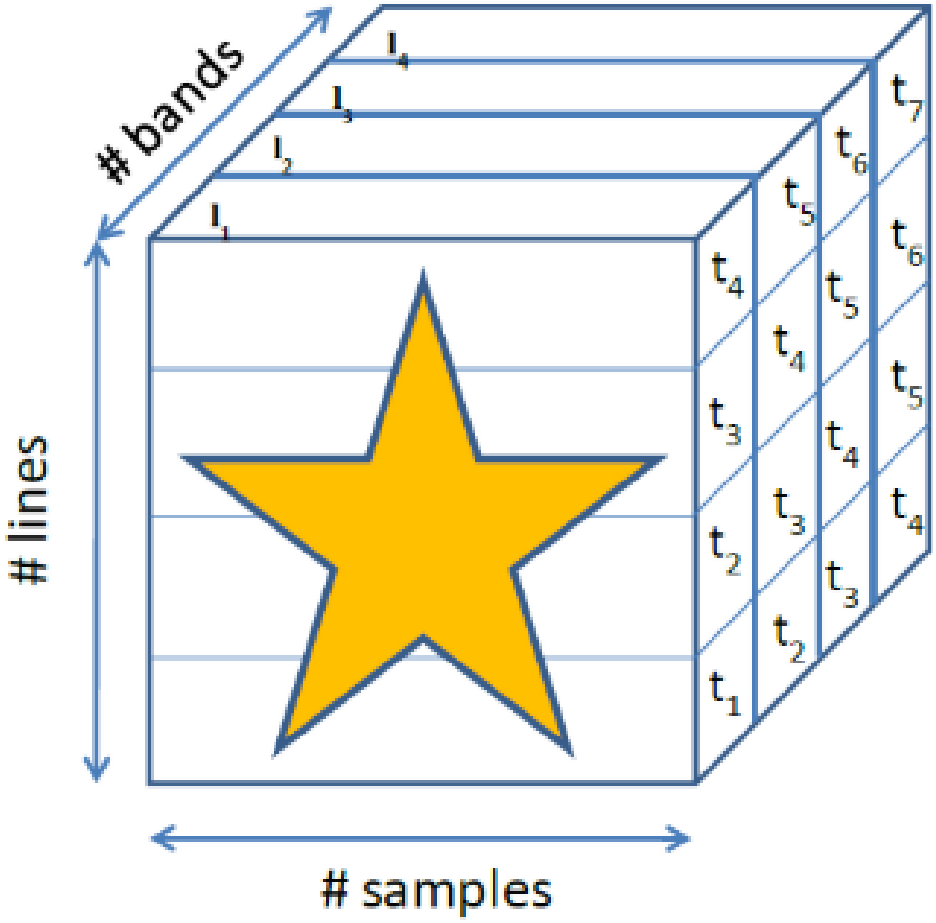

Figure 23 – Ideal hyperspectral cube

3.3.5 Kmeans clusterring

By attempting to divide samples into n groups of equal variances, the KMeans algorithm clusters data, minimizing a criterion known as inertia or within-cluster sum-of-squares. The number of clusters to be listed is required by this algorithm. It scales well across a large number of samples and has been used in many different fields in a wide variety of application areas. The K-means algorithm seeks to choose centroids that minimize inertia, or the criterion of sum-of-squares within the cluster. As a measure of how internally coherent clusters are, inertia can be recognized. There are three steps to the algorithm in simple terms. The first step is to pick the original centroids, with samples from the dataset being the most basic process. K-means consists of looping between the two other stages after initialization. The first stage allocates each specimen to its nearest centroid. By taking the mean value of all the samples assigned to each previous centroid, the second step generates new centroids. The difference between the old centroids and the new centroids is determined and these last two steps are repeated by

the algorithm until this value is less than a threshold. It repeats, in other words, until the centroids do not shift dramatically[81].

3.3.6 Mini Batch Kmeans clustering

The MiniBatchKMeans is a version of the KMeans algorithm that uses mini batches to decrease processing time while still trying to optimize the same objective function. Mini batches are subsets of the input data, sampled randomly in each iteration of the training. The amount of computation required to converge to a local solution is significantly reduced by these mini batches. Mini-batch k-means yields results that are usually just marginally worse than the regular algorithm, in contrast to other algorithms that decrease the convergence time of k-means. Similar to vanilla k-means, the algorithm iterates between two major stages. Samples are drawn randomly from the dataset in the first stage, to form a mini batch. These are then allocated to the centroid that is closest. The centroids are modified in the second stage. This is achieved on a per-sample basis, in comparison to k-means. The assigned centroid is modified for every sample in the mini batch by taking the streaming average of the sample and all previous samples assigned to that centroid. These procedures are carried out before there is convergence or a predetermined number of iterations[81].

3.4 Implementation

This section is about how the previous sections of this chapter implemented using python programming language.

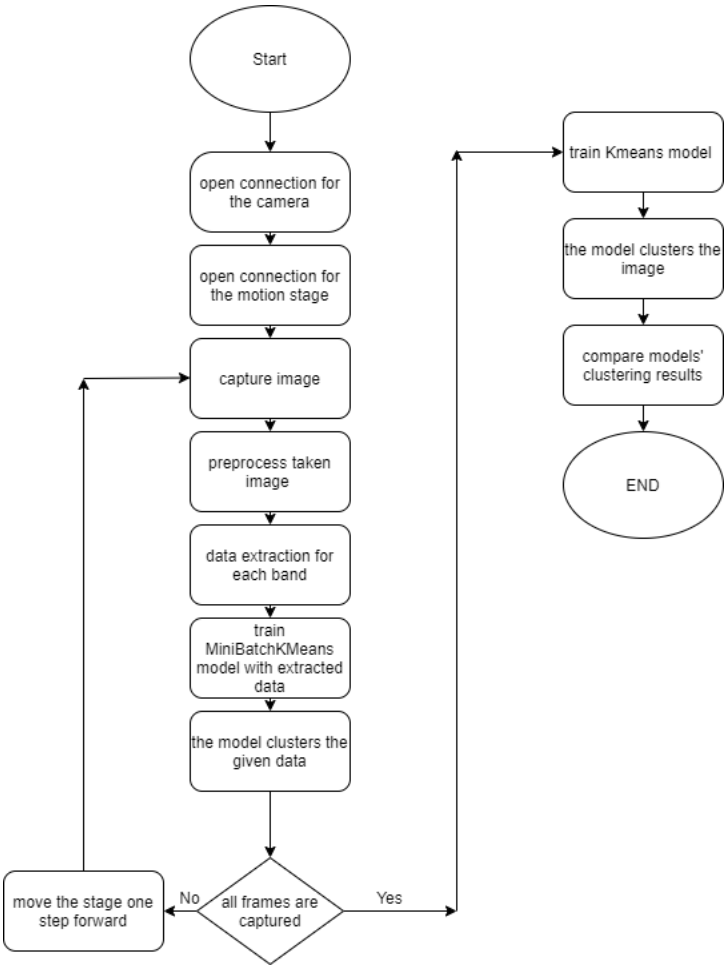

Figure 24 – Flow of the program

3.4.1 The Motion Stage

The motion stage can be controlled over USB port. Python has package which is called pyserial for accessing serial port. Pyserial was installed running following command on terminal.

pip install pyserial

After installing, module was imported to the program.

import serial

To have access to the serial port instance was created for handling port related settings

```
ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=0,
                    bytesize=8, stopbits=serial.STOPBITS_ONE,parity=serial.PARITY_NONE)
```

Figure 25 – Instance for serial port.

If there is not port related problems, then code shown in figure 15 opens the requested port for communications.

After successfully creating port control instance, packets for relative movement and homing are formed as in figure below.

```
pckt = "\x48\x04\x06\x00\xD0\x01\x01\x00\x9a\xf5\x01\x00" #go relative 1 line of cam - 0.313500 mm
pckt1 = "\x11\x04\x01\x00\x50\x01" #request position
gohome = "\x43\x04\x01\x00\x50\x01"  #home
```

Figure 26 – Packets for motion stage.

The value for relative distance will be explained in this section.

The packet for requesting position was created for debugging purposes to verify that requested relative distance was reached. In program move completed message was used to detect end of the movement

As speed of the stepper motor in motion stage is not same as the speed of the execution of the python program, program has to wait until stepper finished the requested relative movement.

3.4.2 The hyperspectral camera

Ximea Linux software package was installed to the board in order to make reliable communication between the camera and the board. xiAPI for python was installed with the package. To get xiAPI to work without any problem some packages has to be downloaded and installed to the board.

Most of the camera controlling code developed on top of the xiAPI example for camera control

NumPy package must be installed for numerical operations with following command.

```
pip install numpy
```

Figure 27 – NumPy installing command.

Matplotlib package must be installed for visualization with following command.

```
pip install matplotlib
```

Figure 28 – Matplotlib installing command.

Pillow and OpenCV must be installed as required from the API

```
pip install Pillow
```

Figure 29 – Pillow installing command.

```
sudo apt-get install libopencv-dev python-opencv
```

Figure 30 – OpenCV installing command.

After installing all required modules, all the used ones were imported to the program.

```
from ximea import xiapi
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Figure 31 – Imported modules.

After that instance for handling camera related tasks was created. Communication between the board and the camera was opened, and also basic settings which were taken from the previous[5] work passed to camera using camera instance.

```
#create instance for first connected camera
cam = xiapi.Camera()

#start communication
print('Opening first camera...')
cam.open_device()

#camera settings
cam.set_imgdataformat('XI_RAW8')
cam.set_buffer_policy('XI_BP_SAFE')
cam.set_gain(4)
cam.set_exposure(100000)
```

Figure 32 – Camera settings.

After setting basic configurations, camera can start acquisition.

```
#start data acquisition
print('Starting data acquisition...')
cam.start_acquisition()
```

Figure 33 – acquisition start command.

3.4.3 Preprocessing captured images

As explained before , not all surface of sensor is covered with filters. In order to use active area with filters , some constant was defined at the beginning of the program. They are pixel height, offset from edges, starting and ending position of the empty interface zone. After removing 2 offsets and the height of the empty interface zone, height of the active are was formed. These variables were defined as constants to easily change values for different cameras on the series.

```
pixel = 5
offset  = 4
empty_interface_zone_start  = 323
empty_interface_zone_end    = 443
height  = cam.get_height()
width   = cam.get_width()

lines   = height - 2 * offset - (empty_interface_zone_end - empty_interface_zone_start)

bands   = int(lines / pixel)
```

Figure 34 – Constants for preprocessing.

The defined constants were used with NumPy array indexing to cut the edges as well as empty interface region.

```
data = data [offset : height - offset, offset : width - offset]
data = np.delete(data,
                 np.s_[empty_interface_zone_start - offset : empty_interface_zone_end - offset], 0 )
```

Figure 35 – Removing edges and empty interface region.

3.4.4 Generating hyperspectral cube

To get hyperspectral cube without any misalignments, some parameters for object were defined such as object width. working distance, focal length.

To keep scanning time reasonable, object width can be modified according to the object. Focal length for the lens is 35m , line height is 5px which means 27.5 micrometer .

From the equations of lenses[82], the 5px line on sensor corresponds to 313.5 micrometer.

```
#object parameters
object_width        = 60
working_distance    = 400
line_height         = 0.0275
focal_length        = 35

object_resolution   = (working_distance * line_height) / focal_length
```

Figure 36 – Defining object parameters.

Camera started to capture from edge of the object. In order to scan whole object with 0th band , at least (object width)/(object resolution) frames are needed, and to complete scanning for all bands extra 191 frames are needed. If this is compared with the previous work[5], it can be seen that this line does not waste extra 191 frames

```
frames              = int((object_width / object_resolution  + ((bands - 1))))
```

Figure 37 – Defining frames variable.

To generate hyperspectral cube 3-dimensional array(bands, height, width) was needed. With the help of NumPy module of python , it is easy to work with n dimensional arrays.at the beginning array was initialized with zeros.

```
cube        = np.zeros((bands,frames*pixel, width - offset * 2), dtype = np.uint8)
```

Figure 38 – Initializing 3D array for hyperspectral cube

The program must capture as much as “frames” variable. Making a loop that counts captured frames was implemented. After every frame has taken, program waits until move complement message which is an improvement to the previous work where 2nd computer was used with windows OS to control the motion stage[5]. Commented out lines were there for debugging purposes.

```
while(i < frames):
        print(i)
        #break
        ser.write(pckt)
        #ser.write(pckt1)
        while(a[:2] != "64"): #check move complete
                while not ser.inWaiting():
                        continue
                a = str(ser.readline().encode("hex"))
                #print(a)
        a=""
```

Figure 39 – Loop for frame capturing with motion stage control.

Another improvement to the previous work[5] is parsing image data for constructing hyperspectral cube. In the previous work every pixel was scanned to generate data, but in this work image data was parsed line by line.

```
for band in range(192):
        if i >= band and i<= int(object_width/object_resolution) + band:
                k = 191-band
                #print(data[k*5:k*5+5].shape)
                cube[k][counter[k]*5:(counter[k]+1)*5]= data[k*5:k*5+5]
                counter[k] = counter[k] + 1
```

Figure 40 – Parsing image data for bands.

After capturing all frames , hyperspectral cube can be viewed using view_cube() function from spectral module in python.

3.4.5 Kmeans clustering

There are lots of python modules that make it easy to apply Kmeans clustering algorithm. In this work scikit-learn was used because ,it is simple and efficient one[42].

Every band is 2-dimensional array. Kmean from scikit requires 1-Dimensional array so a buffer array was created to hold flattened array. Then number of clusters has to be defined. After that using flattened array, kmeans instance was trained then the instance was used to predict clusters in the image. After prediction completed, prediction result converted back to 2-dimensional array. Matplotlib was used to visualize result.

```
#start KMEans clustering
flat_data_km = np.empty((cube.shape[1]*cube.shape[2],5))
for fi in [50,51]:
        band = cube[fi,:,:]
        flat_data_km[:,fi-50-2] = band.flatten()
km = KMeans(n_clusters=2)
km.fit(flat_data_km)
flat_predictions = km.predict(flat_data_km)
prediction_mask = flat_predictions.reshape((cube.shape[1], cube.shape[2]))
pre_rotated = cv2.rotate(prediction_mask,cv2.ROTATE_90_CLOCKWISE)
plt.subplot(1,2,1)
plt.title("KMeans Clustering")
plt.imshow(pre_rotated)
```

Figure 41 – Kmeans implementation.

3.4.6 MiniBatchKmeans clustering

When using Kmeans algorithm, data passed for training the model cannot be changed. But in MiniBatchKmeans algorithm training data can be updated. This algorithm also implemented from scikit-learn library. General sequence of both algorithms is the same, but in MiniBatchKmean algorithm, small size of data can be used to train model. That's why, this algorithm was used directly on the data that had been extracted from captured image to construct hyperspectral data. In every iteration,5px of image data was used to train model.

```
flat_data[(i-band)*10200:(i-band+1)*10200,0] = data[k*5:(k+1)*5].flatten()
mbkm = mbkm.partial_fit(flat_data[(i-band)*10200:(i-band+1)*10200])
pre_data[(i-band)*5:(i-band+1)*5,:] = mbkm.predict(flat_data[(i-band)*10200:(i-band+1)*10200]).reshape(5,2040)
cv2.imshow("raw 51th band",cv2.rotate(cv2.resize(cube[50],None,fx=0.5,fy=0.5,interpolation = cv2.INTER_LINEAR),
                                                                    cv2.ROTATE_90_CLOCKWISE))
cv2.imshow("MBKM clustering using 51th band",cv2.rotate(cv2.resize(pre_data,None,fx=0.5,fy=0.5,
                                            interpolation = cv2.INTER_LINEAR),cv2.ROTATE_90_CLOCKWISE))
```

Figure 42 – MiniBatchKmeans implementation.

For every iteration, extra ~60ms was spent for training the model.

OpenCV was used to visualize the result.

4 TESTING THE SYSTEM

This chapter is about procedures for testing and results.

4.1 Overview

All system components assembled in NRG-202 room. Available lighting in the room was used, no special lamp was installed. Curtains were closed to block outer lightings. Hardware components were powered with their own adapters. Objects were placed 400mm away from the camera. Objects' width was configured on the software for 70mm. Black fabric was used to cancel light reflections between camera and the object. Before starting system, some necessary commands were run on Ubuntu terminal. "*sudo tee /sys/module/usbcore/parameters/usbfs_memory_mb >/dev/null <<<0"* for USB3 buffer problem for the camera and "*sudo chmod 666 /dev/ttyUSB0"* for using USB port to communicate with the motion stage without administration privileges. "datetime.datetime.now()" function from datetime module was used to measure timing where appropriate. 2 fruits were used to test the system.

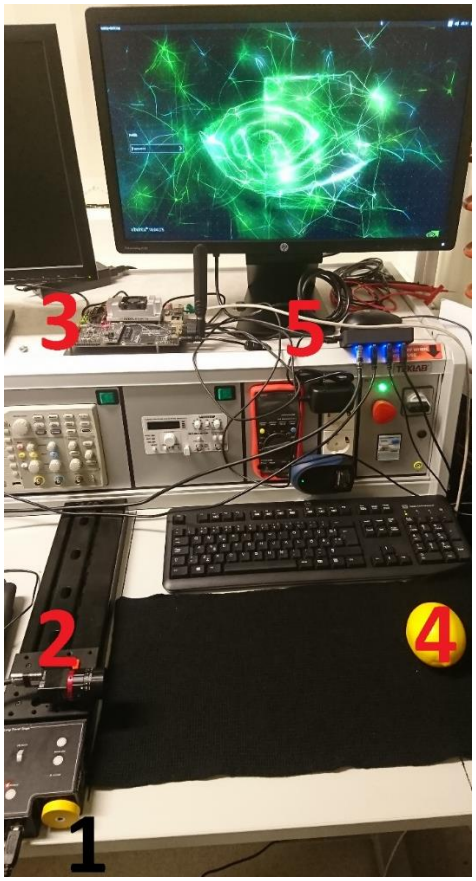

Figure 43 – the developed system(1-The motion stage, 2- The camera, 3- Jetson TX2, 4 – object to be scanned).

4.2 Results

While scanning the object for generating hyperspectral cube ~20us was spent on extracting data(5x2040) for each band. Which is approximately 2 times faster than the previous work[5] in which ~43us was spent. Controlling motion stage from the jetson board helped to use the jetson as it meant for. Previously the motion stage was controlled by another computer.

Scanning object with push broom method, stitching appropriate data together to generate image for each band in real time and generating hypercube were successful.

System was tested with orange and lemon. While gathering data for generating image for bands, same data was used for MiniBatchKmeans without any problem. MiniBatchKmeans instance was configured for finding 2 clusters on software. For each 5x2040 portion of the image, approximately 60ms was spent on clustering and fitting the data in clusters. At the end of the scanning the object, Kmeans algorithm was used for offline processing. Kmeans algorithm was also configured for 2 clusters. Both algorithms produced almost the same result, but if the result was compared with ground truth image, it was obvious that bottom side of the objects were not clustered because of the shadow of themselves.

Figure 44 – Raw data - Orange.

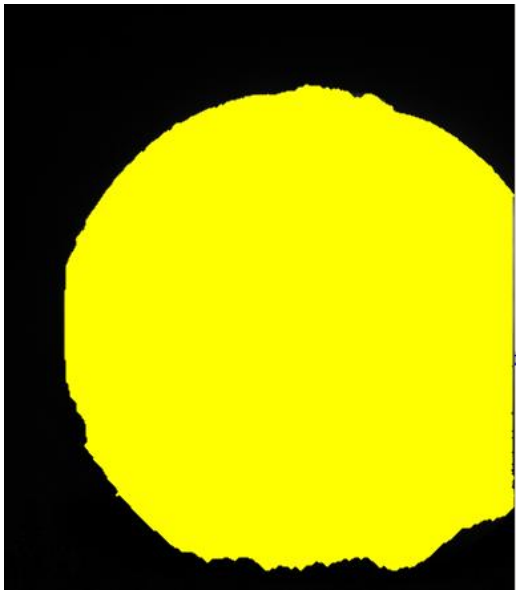

Figure 45 – Ground truth - Orange

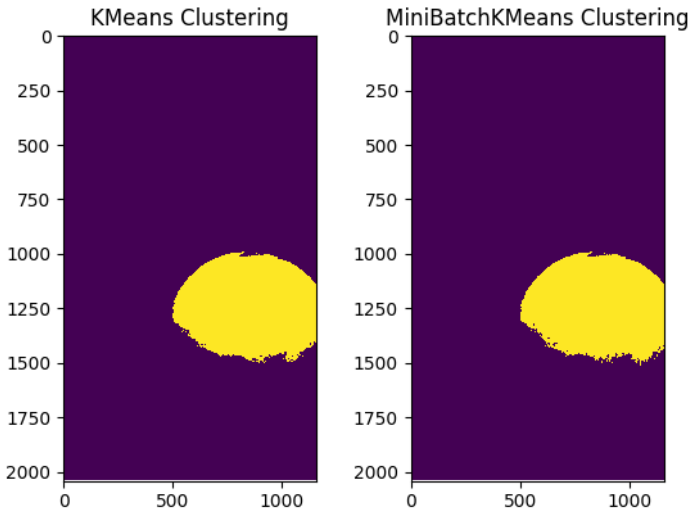

Figure 46 – Clustering results - Orange.

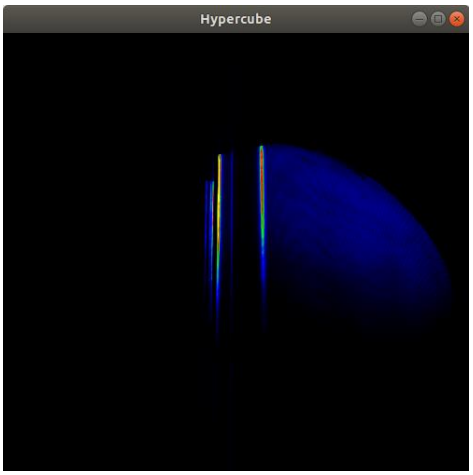

Figure 47 – Hyperspectral cube – Orange.

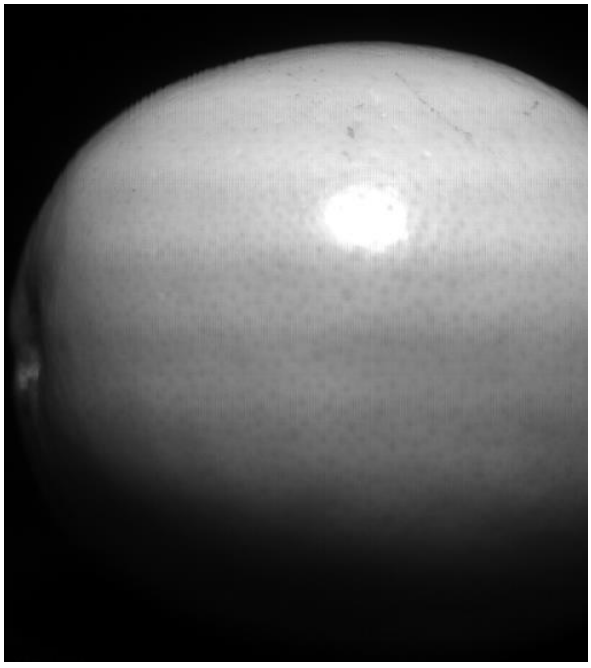

Figure 48 – Raw image - Lemon.

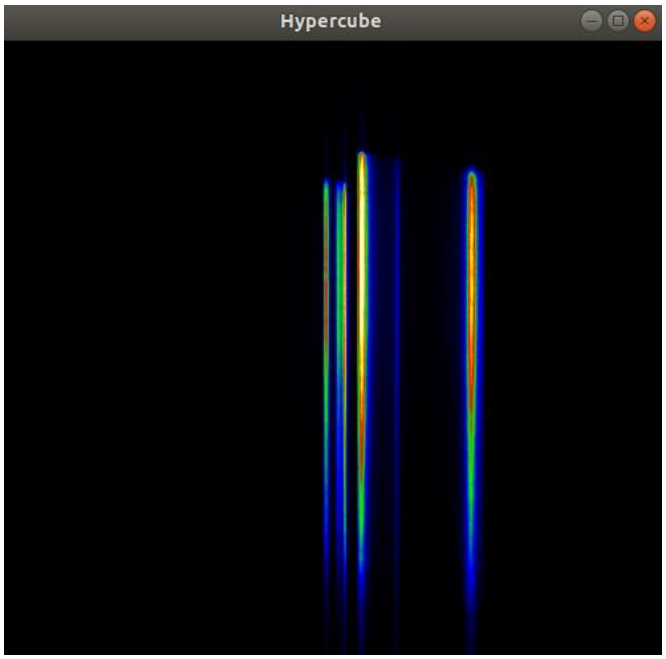

Figure 49 – Hyperspectral cube - Lemon.

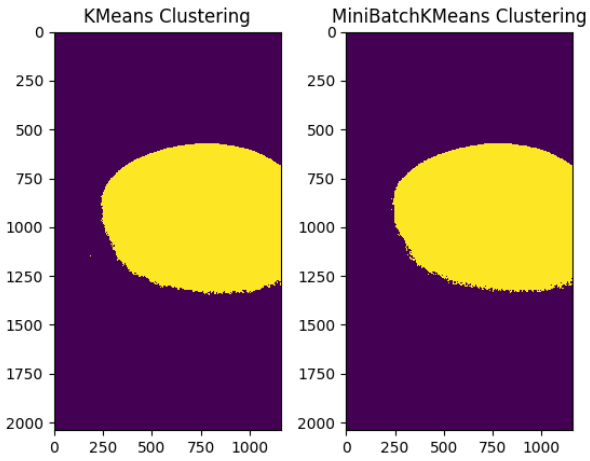

Figure 50 - Clustering results - Lemon.

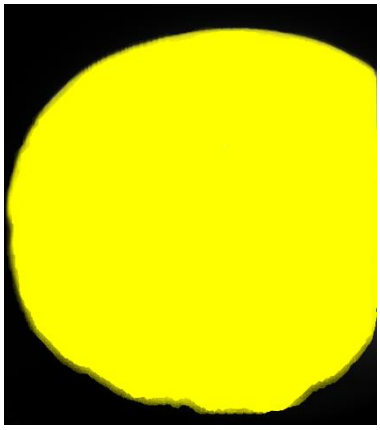

Figure 51 – Ground truth - lemon

5 SUMMARY

LTS300/M was successfully controlled by the Nvidia Jetson TX2 to apply push broom scan method. Controlling LTS300/M directly from the board means that the system is totally mobile now compared to the previous work. While scanning object, collected information was used for both unsupervised clustering and hyperspectral cube generating.

Result of the applied unsupervised algorithm was as expected. MiniBatchKMeans which was used for online clustering generated almost same result as offline applied Kmeans algorithm.

5.1 Future work and suggestions

Results were satisfactory, but the system still needs improvements:

- Scanning better to be performed from top side not from front side of the object
- Speed of the moving stage can be increased
- Better to implement edge detection algorithms to define start of the scanning

There is not much dataset for laboratory research, so it will be better to use the developed system to generate datasets for different objects.

6 KOKKUVÕTE

LTS300 / M-i kontrollis edukalt Nvidia Jetson TX2, et rakendada luudade tõukamismeetodit. LTS300 / M juhtimine otse laualt tähendab, et süsteem on võrreldes eelmise tööga nüüd täiesti mobiilne. Objekti skannimise ajal kasutati kogutud teavet nii järelevalveta klastrite loomiseks kui ka hüperspektrilise kuubi genereerimiseks.

Rakendatud järelevalveta algoritmi tulemus oli ootuspärane. Veebiklastrite jaoks kasutatud MiniBatchKMeans andis peaaegu sama tulemuse kui võrguühenduseta rakendatud Kmeansi algoritm.

6.1 Edasine töö ja ettepanekud

Tulemused olid rahuldavad, kuid süsteem vajab siiski täiendamist:

- Parem skaneerimine tuleb teha ülevalt, mitte esiosalt
- Liikuva etapi kiirust saab suurendada
- Parem rakendada servade tuvastamise algoritme, et määratleda skannimise algus

Laboratoorsete uuringute jaoks pole palju andmekogumeid, seega on parem kasutada väljatöötatud süsteemi erinevate objektide andmekogumite loomiseks

LIST OF REFERENCES

[1] "XIMEA - MQ022HG-IM-LS150-VISNIR." [Online]. Available: https://www.ximea.com/en/products/hyperspectral-cameras-based-on-usb3-xispec/mq022hg-im-ls150-visnir. [Accessed: 04-Nov-2020].

[2] "What Is Hyperspectral Imaging? | NIREOS." [Online]. Available: https://www.nireos.com/hyperspectral-imaging/. [Accessed: 21-Dec-2020].

[3] P. H. Randhe, S. S. Durbha, and N. H. Younan, "Embedded high performance computing for on-board hyperspectral image classification," in *Workshop on Hyperspectral Image and Signal Processing, Evolution in Remote Sensing*, 2016, vol. 0.

[4] "Embedded Systems Developer Kits & Modules from NVIDIA Jetson." [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/. [Accessed: 03-Nov-2020].

[5] A. Palshin, "MOBILE HYPERSPECTRAL ACQUISITION SYSTEM," 2019.

[6] H. Irmak, G. B. Akar, and S. E. Y, "IMAGE FUSION FOR HYPERSPECTRAL IMAGE SUPER-RESOLUTION (1) Radar and Electronic Warfare Systems Business Sector , Aselsan Inc ., Ankara , TURKEY (2) Middle East Techical University , Dept . of Electrical and Electronics Eng ., Ankara , TURKEY (3) Hac," *2018 9th Work. Hyperspectral Image Signal Process. Evol. Remote Sens.*, vol. 3, no. 1, pp. 1–5.

[7] Y. Oiknine, I. August, and A. Stern, "Along-track scanning using a liquid crystal compressive hyperspectral imager," *Opt. Express*, vol. 24, no. 8, p. 8446, Apr. 2016.

[8] "Spectral Imaging and Linear Unmixing | Nikon's MicroscopyU." [Online]. Available: https://www.microscopyu.com/techniques/confocal/spectral-imaging-and-linear-unmixing. [Accessed: 01-Dec-2020].

[9] G. Lu and B. Fei, "Medical hyperspectral imaging: a review," *J. Biomed. Opt.*, vol. 19, no. 1, p. 010901, Jan. 2014.

[10] G. Coltof, "Hyperspectral Techniques Explained," 2012.

[11] "HYPERSPECTRAL IMAGING: One-shot camera obtains simultaneous hyperspectral data | Laser Focus World." [Online]. Available: https://www.laserfocusworld.com/detectors-imaging/article/16562077/hyperspectral-imaging-oneshot-camera-obtains-simultaneous-hyperspectral-data. [Accessed: 01-Dec-2020].

[12] “ESA - Hyperspectral image ‘data cube.’” [Online]. Available: https://www.esa.int/ESA_Multimedia/Images/2014/04/Hyperspectral_image_data_cube. [Accessed: 07-Dec-2020].

[13] A. Public, “Improves vision and discrimination power by using spectral signature information of surface material / object being captured,” 2014.

[14] K. Safari, S. Prasad, and D. Labate, “A Multiscale Deep Learning Approach for High-Resolution Hyperspectral Image Classification,” *IEEE Geosci. Remote Sens. Lett.*, pp. 1–5, Feb. 2020.

[15] S. T. Wahyuni Siregar, W. Handayani, and A. H. Saputro, “Bananas moisture content prediction system using Visual-NIR imaging,” in *Proceedings of the 2017 5th International Conference on Instrumentation, Control, and Automation, ICA 2017*, 2017, pp. 89–92.

[16] A. Soszyńska, M. Müller-Rowold, and R. Reulke, “Feasibility Study of Hyperspectral Line-Scanning Camera Imagery for Remote Sensing Purposes,” in *International Conference Image and Vision Computing New Zealand*, 2019, vol. 2018-November.

[17] R. T. Nallapu *et al.*, “Smart camera system on-board a CubeSat for space-based object reentry and tracking,” in *2018 IEEE/ION Position, Location and Navigation Symposium, PLANS 2018 - Proceedings*, 2018, pp. 1294–1301.

[18] D. Nakaya *et al.*, “Development of high-performance pathological diagnosis software using a hyperspectral camera,” in *2018 IEEE EMBS Conference on Biomedical Engineering and Sciences, IECBES 2018 - Proceedings*, 2019, pp. 217–220.

[19] M. De Landro *et al.*, “Hyperspectral imaging system for monitoring laser-induced thermal damage in gastric mucosa,” in *IEEE Medical Measurements and Applications, MeMeA 2020 - Conference Proceedings*, 2020.

[20] A. Zahavi, A. Palshin, D. C. Liyanage, and M. Tamre, “Influence of illumination sources on hyperspectral imaging,” in *Proceedings of the 2019 20th International Conference on Research and Education in Mechatronics, REM 2019*, 2019.

[21] Y. Li, J. Wang, T. Gao, Q. Sun, L. Zhang, and M. Tang, “Adoption of Machine Learning in Intelligent Terrain Classification of Hyperspectral Remote Sensing Images,” *Comput. Intell. Neurosci.*, vol. 2020, p. 8886932, 2020.

[22] B. Yang, M. Yang, A. Plaza, L. Gao, and B. Zhang, “Dual-Mode FPGA Implementation of Target and Anomaly Detection Algorithms for Real-Time Hyperspectral Imaging,” *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 8, no. 6, pp. 2950–2961, Jun. 2015.

[23] “Anomaly Detection for Dummies. Unsupervised Anomaly Detection for… | by Susan Li | Towards Data Science.” [Online]. Available: https://towardsdatascience.com/anomaly-detection-for-dummies-15f148e559c1. [Accessed: 26-Dec-2020].

[24] X. Yao and C. Zhao, “Kernel-band-projection algorithm for anomaly detection in hyperspectral imagery,” in *International Conference on Signal Processing Proceedings, ICSP*, 2019, vol. 2018-August, pp. 300–303.

[25] C. Zhao, W. You, J. Wang, and Y. Wang, “Kernel subspace-based real-time anomaly detection for hyperspectral imagery,” in *International Geoscience and Remote Sensing Symposium (IGARSS)*, 2015, vol. 2015-November, pp. 1865–1868.

[26] J. Wei and X. Wang, “An Overview on Linear Unmixing of Hyperspectral Data,” *Mathematical Problems in Engineering*, vol. 2020. Hindawi Limited, 2020.

[27] S. Bernabe, L. I. Jimenez, C. Garcia, J. Plaza, and A. Plaza, “Multicore Real-Time Implementation of a Full Hyperspectral Unmixing Chain,” *IEEE Geosci. Remote Sens. Lett.*, vol. 15, no. 5, pp. 744–748, May 2018.

[28] “What is an FPGA? Field Programmable Gate Array.” [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. [Accessed: 27-Dec-2020].

[29] “FPGA-based Data Processing Increases Hyperspectral Camera Resolution | Electronic Design.” [Online]. Available: https://www.electronicdesign.com/technologies/fpgas/article/21800944/fpgabased-data-processing-increases-hyperspectral-camera-resolution. [Accessed: 27-Dec-2020].

[30] J. Lei *et al.*, “A Novel FPGA-Based Architecture for Fast Automatic Target Detection in Hyperspectral Images,” *Remote Sens.*, vol. 11, no. 2, p. 146, Jan. 2019.

[31] R. Guerra, L. Santos, S. Lopez, and R. Sarmiento, “A New Fast Algorithm for Linearly Unmixing Hyperspectral Images,” *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 12, pp. 6752–6765, Dec. 2015.

[32] R. Guerra, S. López, and R. Sarmiento, “A FPGA implementation for linearly unmixing a hyperspectral image using OpenCL,” in *SPIE*, 2017, vol. 10430, p. 16.

[33] S. Vellas, G. Lentaris, K. Maragos, D. Soudris, Z. Kandylakis, and K. Karantzalos, “FPGA acceleration of hyperspectral image processing for high-speed detection applications,” in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2017.

[34] A. Yusuf and S. Alawneh, “A Survey of GPU Implementations for Hyperspectral Image

Classification in Remote Sensing," *Can. J. Remote Sens.*, vol. 44, no. 5, pp. 532–550, 2018.

[35] C. Li, Y. Peng, M. Su, and T. Jiang, "GPU Parallel Implementation for Real-Time Feature Extraction of Hyperspectral Images," *Appl. Sci.*, vol. 10, no. 19, p. 6680, Sep. 2020.

[36] Z. Wu, Q. Wang, A. Plaza, J. Li, L. Sun, and Z. Wei, "Parallel Spatial-Spectral Hyperspectral Image Classification with Sparse Representation and Markov Random Fields on GPUs," *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 8, no. 6, pp. 2926–2938, Jun. 2015.

[37] W. Sun and Q. Du, "Hyperspectral Band Selection: A Review," *IEEE Geosci. Remote Sens. Mag.*, vol. 7, no. 2, pp. 118–139, 2019.

[38] A. Fontanella, E. Marenzi, E. Torti, G. Danese, A. Plaza, and F. Leporati, "A suite of parallel algorithms for efficient band selection from hyperspectral images," in *Journal of Real-Time Image Processing*, 2018, vol. 15, no. 3, pp. 537–553.

[39] R. Nijhawan, I. Srivastava, and P. Shukla, "Land cover classification using super-vised and unsupervised learning techniques," in *ICCIDS 2017 - International Conference on Computational Intelligence in Data Science, Proceedings*, 2018, vol. 2018-January, pp. 1–6.

[40] W. Lv and X. Wang, "Overview of Hyperspectral Image Classification," *J. Sensors*, vol. 2020, p. 4817234, 2020.

[41] "ML | Mini Batch K-means clustering algorithm - GeeksforGeeks." [Online]. Available: https://www.geeksforgeeks.org/ml-mini-batch-k-means-clustering-algorithm/. [Accessed: 01-Jan-2021].

[42] "scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation." [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 21-Dec-2020].

[43] "Jetson Modules | NVIDIA Developer." [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules. [Accessed: 03-Nov-2020].

[44] "Jetson TX2 Module | NVIDIA Developer." [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2mendeley. [Accessed: 03-Dec-2020].

[45] "Harness AI at the Edge with the Jetson TX2 Developer Kit | NVIDIA Developer." [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2-developer-kit. [Accessed: 03-Dec-2020].

[46] "JetPack SDK | NVIDIA Developer." [Online]. Available: https://developer.nvidia.com/embedded/jetpack. [Accessed: 03-Nov-2020].

[47] S. Aldegheri and N. Bombieri, “Rapid Prototyping of Embedded Vision Systems: Embedding Computer Vision Applications into Low-Power Heterogeneous Architectures,” in *Proceedings - IEEE International Symposium on Rapid System Prototyping, RSP*, 2019, vol. 2018-October, pp. 63–69.

[48] N. Deepika and V. V. Sajith Variyar, “Obstacle classification and detection for vision based navigation for autonomous driving,” in *2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2017*, 2017, vol. 2017-January, pp. 2092–2097.

[49] Y. K. Lai, C. Y. Ho, Y. H. Huang, C. W. Huang, Y. X. Kuo, and Y. C. Chung, “Intelligent Vehicle Collision-Avoidance System with Deep Learning,” in *2018 IEEE Asia Pacific Conference on Circuits and Systems, APCCAS 2018*, 2019, pp. 123–126.

[50] J. Shihadeh, A. Ansari, and T. Ozunfunmi, “Deep Learning Based Image Classification for Remote Medical Diagnosis,” in *GHTC 2018 - IEEE Global Humanitarian Technology Conference, Proceedings*, 2019.

[51] Ximea, “Very compact hyperspectral cameras xiSpec,” pp. 4–6.

[52] “XIMEA - Hyperspectral Linescan USB3 camera 150 bands 470-900nm.” [Online]. Available: https://www.ximea.com/en/products/hyperspectral-cameras-based-on-usb3-xispec/mq022hg-im-ls150-visnir. [Accessed: 03-Dec-2020].

[53] “35mm C Series VIS-NIR Fixed Focal Length Lens | Edmund Optics.” [Online]. Available: https://www.edmundoptics.com/p/35mm-c-series-vis-nir-fixed-focal-length-lens/22384/. [Accessed: 03-Dec-2020].

[54] “Thorlabs - LTS300/M 300 mm Translation Stage with Stepper Motor, Integrated Controller, M6 Taps.” [Online]. Available: https://www.thorlabs.com/thorproduct.cfm?partnumber=LTS300/M. [Accessed: 08-Nov-2020].

[55] H. C. Protocol, “Thorlabs APT Motor Controllers Host-Controller Communications Protocol,” no. 15, pp. 1–23, 2006.

[56] “JetPack L4T.” [Online]. Available: https://docs.nvidia.com/jetpack-l4t/2_1/content/developertools/mobile/jetpack/jetpack_l4t/2.0/jetpack_l4t_main.htm#System_Requirements. [Accessed: 05-Dec-2020].

[57] “NVIDIA TensorRT | NVIDIA Developer.” [Online]. Available: https://developer.nvidia.com/tensorrt. [Accessed: 05-Dec-2020].

[58] “NVIDIA cuDNN | NVIDIA Developer.” [Online]. Available: https://developer.nvidia.com/cudnn. [Accessed: 05-Dec-2020].

[59] “VisionWorks | NVIDIA Developer.” [Online]. Available: https://developer.nvidia.com/embedded/visionworks. [Accessed: 05-Dec-2020].

[60] “Home - OpenCV.” [Online]. Available: https://opencv.org/. [Accessed: 05-Dec-2020].

[61] “cuBLAS :: CUDA Toolkit Documentation.” [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html. [Accessed: 05-Dec-2020].

[62] “cuFFT :: CUDA Toolkit Documentation.” [Online]. Available: https://docs.nvidia.com/cuda/cufft/index.html. [Accessed: 05-Dec-2020].

[63] “Vulkan | NVIDIA Developer.” [Online]. Available: https://developer.nvidia.com/Vulkan. [Accessed: 05-Dec-2020].

[64] “OpenGL - The Industry Standard for High Performance Graphics.” [Online]. Available: https://www.opengl.org/. [Accessed: 05-Dec-2020].

[65] “Jetson Linux Multimedia API Reference: Libargus Camera API.” [Online]. Available: https://docs.nvidia.com/jetson/l4t-multimedia/group__LibargusAPI.html. [Accessed: 05-Dec-2020].

[66] J. Xu, B. Wang, J. Li, C. Hu, and J. Pan, “Deep learning application based on embedded GPU,” in *1st International Conference on Electronics Instrumentation and Information Systems, EIIS 2017*, 2018, vol. 2018-January, pp. 1–4.

[67] Y. Oyama, T. Ben-Nun, T. Hoefler, and S. Matsuoka, “Accelerating Deep Learning Frameworks with Micro-Batches,” in *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, 2018, vol. 2018-September, pp. 402–412.

[68] O. Artiles and F. Saeed, “GPU-SFFT: A GPU based parallel algorithm for computing the Sparse Fast Fourier Transform (SFFT) of k-sparse signals,” in *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019*, 2019, pp. 3303–3311.

[69] A. Kumar and S. P. Panda, “A Survey: How Python Pitches in IT-World,” in *Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing: Trends, Prespectives and Prospects, COMITCon 2019*, 2019, pp. 248–251.

[70] D. O. Dantas, H. Danilo Passos Leal, and D. O. Barros Sousa, “Fast multidimensional image processing with OpenCL,” in *Proceedings - International Conference on Image Processing, ICIP*, 2016, vol. 2016-August, pp. 1779–1783.

[71] T. Treebupachatsakul and S. Poomrittigul, “Bacteria Classification using Image Processing and Deep learning,” in *34th International Technical Conference on Circuits/Systems, Computers and Communications, ITC-CSCC 2019*, 2019.

[72] H. Ucuzal, S. Yasar, and C. Colak, “Classification of brain tumor types by deep learning with convolutional neural network on magnetic resonance images using a developed web-based interface,” in *3rd International Symposium on Multidisciplinary Studies and Innovative Technologies, ISMSIT 2019 - Proceedings*, 2019.

[73] “APIs - APIs - ximea support.” [Online]. Available: https://www.ximea.com/support/wiki/apis/APIs. [Accessed: 05-Dec-2020].

[74] “NumPy.” [Online]. Available: https://numpy.org/. [Accessed: 05-Dec-2020].

[75] “Oracle VM VirtualBox.” [Online]. Available: https://www.virtualbox.org/. [Accessed: 19-Dec-2020].

[76] “Install Windows Subsystem for Linux (WSL) on Windows 10 | Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/en-us/windows/wsl/install-win10. [Accessed: 19-Dec-2020].

[77] “XIMEA CamTool - All products - ximea support.” [Online]. Available: https://www.ximea.com/support/wiki/allprod/ximea_camtool. [Accessed: 19-Dec-2020].

[78] “300 mm Linear Translation Stage with Integrated Controller, Stepper Motor.” [Online]. Available: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=7652&pn=LTS300. [Accessed: 03-Dec-2020].

[79] “Linux USB30 Support - APIs - ximea support.” [Online]. Available: https://www.ximea.com/support/wiki/apis/Linux_USB30_Support. [Accessed: 21-Dec-2020].

[80] “xiSpec linescan cameras sensor info and data processing,” pp. 1–8.

[81] “2.3. Clustering — scikit-learn 0.23.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html#k-means. [Accessed: 19-Dec-2020].

[82] “Lens Formula and Magnification | CK-12 Foundation.” [Online]. Available: https://www.ck12.org/book/cbse-physics-book-class-x/section/1.6/. [Accessed: 19-Dec-2020].

APPENDICES

```
from ximea import xiapi
import PIL.Image
from PIL import Image
import cv2
import numpy as np
import serial
from os import mkdir
from os.path import expanduser
import time
import PIL.Image
#from spectral import *
from sklearn.cluster import KMeans
from sklearn.cluster import MiniBatchKMeans

import matplotlib.pyplot as plt

np.set_printoptions(threshold=np.inf)
#np.set_printoptions(threshold=1000)

path        = expanduser("~/Documents/test/")

#create instance for first connected camera
cam = xiapi.Camera()

#start communication
print('Opening first camera...')
cam.open_device()

#camera settings
cam.set_imgdataformat('XI_RAW8')
cam.set buffer policy('XI BP SAFE')
cam.set_gain(4)
cam.set_exposure(100000)

pixel = 5
offset  = 4
empty_interface_zone_start  = 323
empty interface zone end    = 443
height  = cam.get_height()
width   = cam.get_width()

lines   = height - 2 * offset - (empty interface zone end -
empty_interface_zone_start)
bands   = int(lines / pixel)

#object parameters
object_width        = 70
working_distance    = 400 #Distance to object
line height         = 0.0275 # (Seonsor height / ROW pixel) * pixel per line
focal_length        = 35 # Focal length in mm

#calculation
object_resolution   = (working_distance * line_height) / focal_length
frames              = int((object_width / object_resolution  + ((bands - 1))))

cube        = np.zeros((bands,int(object width/object resolution)*pixel+50, width -
offset * 2), dtype = np.uint8)
```

```
#print("**********************")
#print(type(cube))
#settings
#cam.set_imgdataformat('XI_RGB24')
#cam.set_exposure(10000)

#create instance of Image to store image data and metadata
img = xiapi.Image()

#start data acquisition
print('Starting data acquisition...')
cam.start_acquisition()

ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=0, bytesize=8,
stopbits=serial.STOPBITS_ONE,parity=serial.PARITY_NONE)


pckt = "\x48\x04\x06\x00\xD0\x01\x01\x00\x9a\xf5\x01\x00" #go relative 1 line of cam
- 0.313500 mm
pckt1 = "\x11\x04\x01\x00\x50\x01" #request position
gohome = "\x43\x04\x01\x00\x50\x01"  #home

#definitions for cube
counter = [0]*192
i = 0
a=""

flat_data = np.empty((cube.shape[1]*cube.shape[2],1))
pre_data  = np.empty((cube.shape[1],cube.shape[2]))

km = KMeans(n_clusters=2)
mbkm = MiniBatchKMeans(n_clusters=2)
show = True

while(i < frames):
    print("frame: {}".format(i))
    #ser.write(gohome)
    #break
    ser.write(pckt)
    ser.write(pckt1)
    while(a[:2] != "64"): #check move complete
        while not ser.inWaiting():
            continue
        a = str(ser.readline().encode("hex"))


        #print(a)
    a=""


    cam.get_image(img)
        #print(type(img))
    #create numpy array with data from camera. Dimensions of array are determined
    #by imgdataformat
    #NOTE: PIL takes RGB bytes in opposite order, so invert_rgb_order is True
    data = img.get_image_data_numpy(invert_rgb_order=True)
    data = data [offset : height - offset, offset : width - offset] #Cut the edges
    data = np.delete(data, np.s_[empty_interface_zone_start - offset :
    empty_interface_zone_end - offset], 0 ) #Cut the sensor connection (empty
    interface zone)

    for band in range(192):
        if i >= band and i<= int(object_width/object_resolution) + band:
            k = 191-band
            #print(data[k*5:k*5+5].shape)
            cube[k][counter[k]*5:(counter[k]+1)*5]= data[k*5:k*5+5]
            if band == 141:
                flat data[(i-band)*10200:(i-band+1)*10200,0] =
                data[k*5:(k+1)*5].flatten()
                mbkm = mbkm.partial_fit(flat_data[(i-band)*10200:(i-band+1)*10200])
                pre data[(i-band)*5:(i-band+1)*5,:] =
                mbkm.predict(flat_data[(i-band)*10200:(i-band+1)*10200]).reshape(5,204
                0)
                cv2.imshow("raw 51th
                band",cv2.rotate(cv2.resize(cube[50],None,fx=0.5,fy=0.5,interpolation
                = cv2.INTER_LINEAR),

                                                        cv2.ROTATE 90 CLOCKWISE)
```

```
                cv2.imshow("MBKM clustering using 51th
                band",cv2.rotate(cv2.resize(pre_data,None,fx=0.5,fy=0.5,
                                            interpolation =
                                            cv2.INTER_LINEAR),cv2.ROTATE_90_CLOCKWISE)
                                            )

                #cv2.waitKey(1)

            counter[191-band] = counter[191-band] + 1
    i=i+1
    if cv2.waitKey(1) == ord('q'):
            break


#stop data acquisition
print('Stopping acquisition...')
cam.stop_acquisition()
#start KMEans clustering
flat_data_km = np.empty((cube.shape[1]*cube.shape[2],5))
for fi in [50,51]:
    band = cube[fi,:,:]
    flat_data_km[:,fi-50-2] = band.flatten()
km = KMeans(n_clusters=2)
km.fit(flat_data_km)
flat_predictions = km.predict(flat_data_km)
prediction_mask = flat_predictions.reshape((cube.shape[1], cube.shape[2]))
pre_rotated = cv2.rotate(prediction_mask,cv2.ROTATE_90_CLOCKWISE)
plt.subplot(1,2,1)
plt.title("KMeans Clustering")
plt.imshow(pre_rotated)

plt.subplot(1,2,2)
plt.title("MiniBatchKMeans Clustering")
plt.imshow(cv2.rotate(pre_data,cv2.ROTATE_90_CLOCKWISE))

plt.show()


cv2.waitKey(0)
cv2.destroyAllWindows()

#stop communication
cam.close_device()
#show hyper cube
#view_cube(cube)
print("saving array...")
np.save('./test/dataF.npy',cube)
print('Done.')
ser.write(gohome)
```