Online hyperspectral analysis of tomatoes

Project report

Document id	Online hyperspectral analysis of tomatoes
Issue	18
Date	June 5, 2015
Pages	36
Copyright	

Approved	Chris van Dijk cosine Research B.V.	
Prepared	Alex Veringmeier cosine Research B.V.	

Preface

I wrote this report as part of my graduation internship at the company cosine Research B.V. When I searched for a project to do for my graduation internship I wanted to do a project in which I could implement vision algorithms for a mechatronical system. A teacher told me about the company cosine which offered me the perfect opportunity to do such a project. The company develops and does all kind of research on measurement systems with vision applications.

In this report I will write about the work that is done during the project. There will be described which phases the project did go trough and what results this did deliver. The report give information about what the result is, how it works and what can be done with it.

This report is meant for the graduation committee which will grade the graduation internship. It is also meant for the company in order to know what I have done during my graduation internship and what can be done with the result. It is written such that an engineering student of my own level can understand the report.

At last I want to thank the company cosine for the great opportunity they gave me to perform my graduation internship. It was really nice to work at the cosine, the colleagues are very nice and it has a nice atmosphere. Also the project itself was really nice to do. I learned a lot of new things and I also was capable to really contribute to the project. Special thanks to my supervisor, Chris van Dijk, who helped me really well with the project. I learned a lot of new things from him and I could always ask him for help when needed. I also want to thank my supervisors from the university, Fidelis Theinert and Dick van Teylingen, who guided me trough the proceedings of the graduation internship.

Delft, 5th of June 2015

Summary

This report is about implementing a pipeline in C++ in order to make it possible to perform online hyperspectral analysis of tomatoes. The project is performed by a mechatronic student as a graduation internship. The report will give insight in the design and realization of software which is capable to record images as fast as possible with the use of concurrent programming. Going through this report it will first describe the definition of the project with its requirements. After that it will describe the preliminary design. Where with the use of diagrams the choices and the design of the software and the hardware will be described. Once it is known how the setup will look like the realization of the software and the hardware is described. At the end some test and the results will be described and concluded.

In order to have a good understanding of the project the project definition will be described. First some background information is given which explains what technologies are available and how these could be used for the project. It explains the type of hyperspectral sensor that will be used and how the hyperspectral data can contribute to the analysis of tomatoes. After this the problem is explained. Once this is done the organization of the project is described. With the knowledge of the definition and the problem of the project the requirements are defined.

Now that the problem and the project are defined and it is know what the requirements are the preliminary design can be described. The preliminary design will help to get an understanding of the configuration of the project. It will describe all the components that will be used and how they are connected. First all the components of the hardware that are needed to make the setup will be described. A conveyer belt is used to transport the tomatoes. A hyperspectral camera will be used to record the images and a FPGA based board will be used to control the process. Block definition diagrams will be used to describe the relation between the components. After this the components of the software are described. There will be described how the Non-virtual interface will be used to secure the maintainability and readability of the code. There will be explained how the pipeline works and how it is constructed. It also explains how there will be secured that the pipeline will be executed as fast as possible. Different diagrams like state machine diagrams describe the behavior of the software.

Once there is clear how the setup will be constructed the design can be realized. There will be described which products are used to realize the design of the hardware. After that there is described how the different components of the software are implemented. There is explained what functions are used and how they behave. The content of each specific task of the pipeline will be described. State machine diagrams will be used to show the behavior of some tasks.

With the realized setup some test can be done. The most interesting characteristic of the setup is the timing. Measurements of the time of the software are done with the use of the internal clock of the processor. With the results of the measurements can be seen how long it will take to go through the entire pipeline and how long it will take to perform the different tasks. At the end of the report the project can be concluded with the use of the results and the realized parts. After this recommendations are given about further steps that could be taken and which research would be interesting to do.

Samenvatting

Dit raport gaat over het implementeren van een pipeline in C++ wat het mogelijk maakt om online hyperspectrale analyze van tomaten uit te voeren. Het project is uitgevoerd door een mechatronica student als afstudeer stage. Het raport beschrijft het ontwerpen en realizeren van software dat in staat is om afbeeldingen zo snel mogelijk op te nemen waarbij concurrent programmeren wordt gebruikt. Gaande door dit raport zal het eerst de definitie van het project en de eisen beschrijven. Daarna zal het verslag het preliminary design beschrijven. Dit beschrijft met het gebruik van diagrammen de keuzes en het ontwerp van de software en de hardware. Zodra het bekend is hoe de setup eruit zal zien wordt de realizatie van de software and de hardware beschreven. Aan het einde zullen de testen en de resultaten worden beschreven en beconcludeerd.

De definitie van het project zal omschreven worden om een goed begrip van het project te krijgen. Eerst wordt er achtergrond informatie gegeven waarin wordt uitgelegd welke technologieën er mogelijk zijn en hoe deze gebruikt kunnen worden voor het project. Er wordt uitgelegd welke type hyperspectrale sensor er wordt gebruikt en hoe de hyperspectrale data iets toe kan voegen aan het analyzeren van tomaten. Hierna wordt de organizatie van het project beschreven. Aan de hand van de definitie van het probleem en het project worden de eisen samengesteld.

Nu dat het probleem en het project zijn vastgelegd kan het preliminary design worden beschreven. Het preliminary design zal de configuratie van het project verduidelijken. Het beschrijft alle componenten die nodig zijn en hoe ze in verband staan met elkaar. Eerst worden alle hardware componenten van de setup beschreven. Een lopende band wordt gebruikt om de tomaten te transporteren. Een hyperspectrale camera wordt gebruikt om afbeeldingen op te nemen en een FPGA gebaseerd boord wordt gebruikt om het process te besturen. Blok definitie diagrammen beschrijven de relatie tussen de componenten. Hierna worden de componenten van de software beschreven. Er wordt uitgelegd hoe the Non-virtual interface gebruikt wordt om de onderhoudbaarheid en de leesbaarheid van de code te verzekeren. Er wordt uitgelegd hoe de pipeline werkt en hoe deze wordt samengesteld. Er wordt ook uitgelegd hoe er wordt verzekerd dat de pipeline zo snel mogelijk uitgevoerd wordt. Verschillende state machine diagrammen beschrijven het gedrag van de software.

Nu dat het duidelijk is hoe de setup samengesteld is kan het gerealiseerd worden. Er wordt beschreven welke producten er zijn gebruikt om het hardware ontwerp te realizeren. Daarna wordt beschreven hoe de verschillende componenten van de software geïmplementeerd zijn. Er wordt uitgelegd welke functies zijn gebruikt en wat ze doen. The inhoud van elke specifieke taak van de pipeline zal worden beschreven. State machine diagrammen worden gebruikt om het gedrag van sommige taken te beschrijven.

Nadat de setup gerealiseerd was kon de setup getest worden. De meest interessante eigenschap van de setup is de timing. Metingen van de tijd van de software zijn uitgevoerd met behulp van the interne klok van de processor. Met de resultaten kan aangetoond worden hoe lang het duurt om door de pipeline heen te gaan en hoe lang het duurt om iedere taak afzonderlijk uit te voeren. Aan het eind van het raport wordt er een conclusie over het project gemaakt. Hierna worden aanbevelingen gegeven over vervolg stappen om het probleem aan te pakken en het doel te halen en er worden aanbevelingen gegeven waar nog onderzoek naar gedaan kan worden.

Table of Contents

1 Project definition

1.1 Background

Hyperspectral imaging is a technique whereby a larger range of spectral (color) information is obtained than standard color imaging: while RGB color imaging can resolve three colors and has a spectral bandwidth of ~100 nm, see Figure 1, hyperspectral imaging in the visible region can resolve hundreds of colors with a bandwidth as low as 5-10 nm, see Figure 2. As a result, a complete spectrum is obtained for every spatial pixel in the dataset, resulting in a datacube with two spatial axes and one spectral axis, see Figure 3.

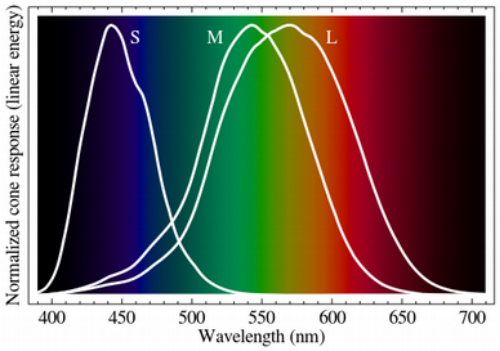


Figure 1: spectral resolution of the human eye [1]

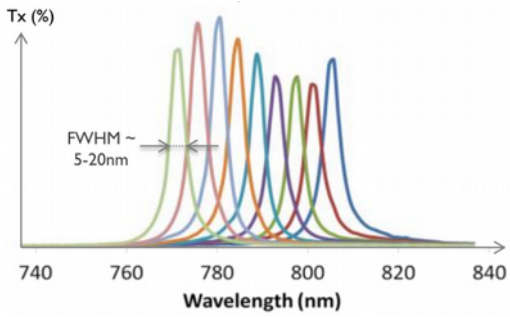


Figure 2: spectral resolution of an IMEC hyperspectral filter [2]

The type of hyperspectral camera used in this study will be an IMEC linearly variable filter (LVF), where a narrow band filter is directly deposited onto a standard CMOS imaging sensor. The center wavelength of the filter varies linearly across one axis of the sensor, such that the sensor has multiple bands which image different wavelengths, see Figure 4. Because of this the object needs to be moved across the field of view of the camera to scan every wavelength. The series of images need to be decomposed (decubed) such that each image contains the spatial data in one specific wavelength. In this way a complete dataset can be obtained.

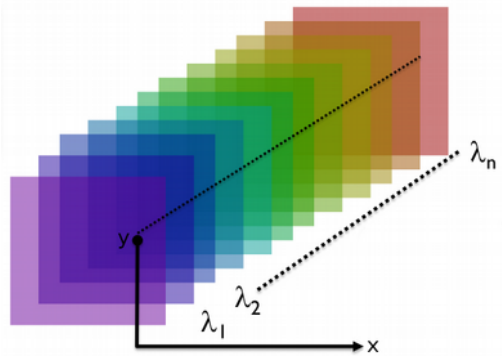


Figure 3: example reconstructed hyperspectral datacube [2]

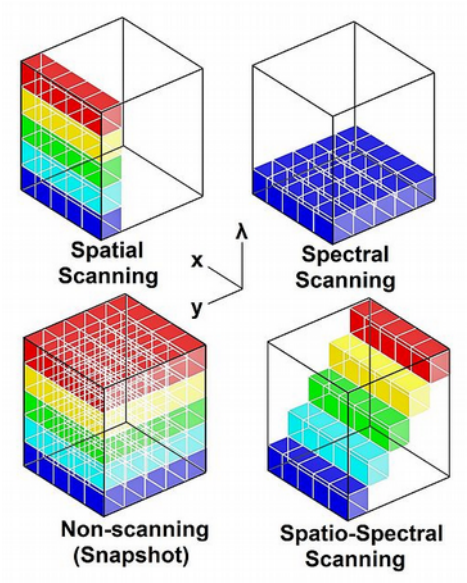


Figure 4: various hyerpsectral acquisition techniques (a LVF is a form of spatio-spectral scanning) [3]

By obtaining the spectrum of an object with a hyperspectral camera, one can perform spectroscopy on specific pixels in the image. Within the application of food analysis, the types of information that can be obtained from the visible spectrum of e.g. a tomato include the specific cultivar and the ripeness (through the sugar content). Each cultivar has a different concentration of lycopene (molecule giving tomatoes their red color), resulting in a different spectral signature (Figure 5) which can be separated via classification, as shown in Figure 6.

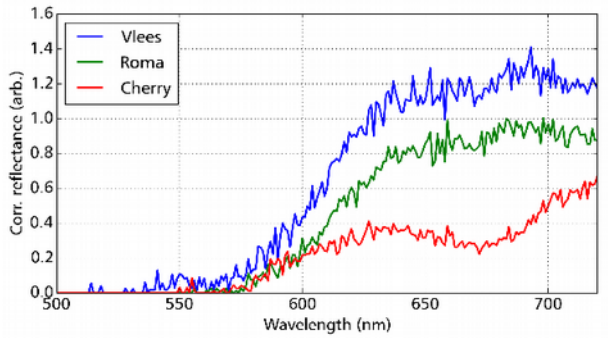


Figure 5: reflectance spectra of various tomato cultivar [4]

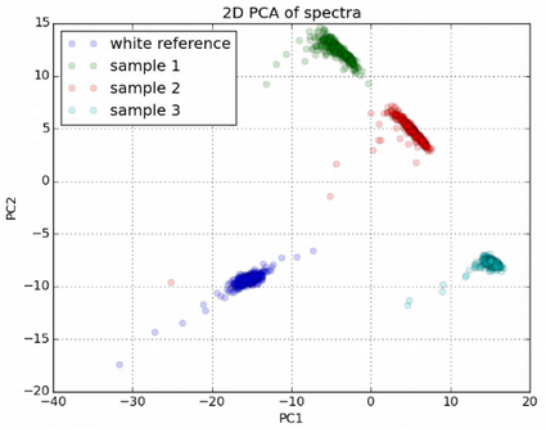


Figure 6: principal component analysis (PCA) of tomato reflectance spectra [4]

The sugar content can be determined by the intensity of peaks near 800-1200 nm (Figure 7), therefore by analysing the relative contribution of these peaks, the sugar content can be determined.

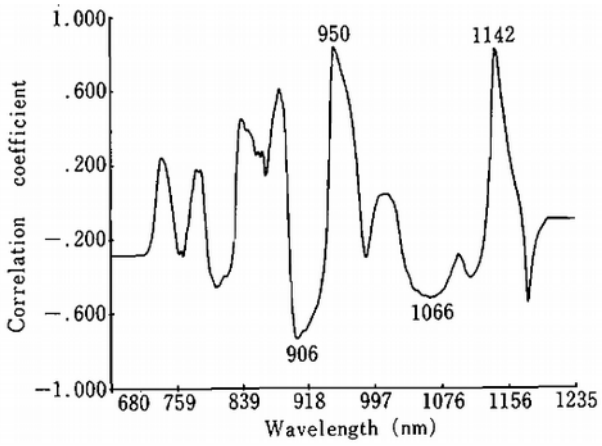


Figure 7: correlation between peaks in the reflectance spectrum and sugar content [5]

1.2 Problem definition

In order to analyze tomatoes and other kind of fruit, data needs to be acquired and analyzed. This will be done using a hyperspectral sensor. A software architecture needs to be developed which is capable of acquiring, preprocessing and analyzing the hyperspectral data within one second. To keep the process fast enough it needs to be tested how much analysis can be done within one second. First some basic vision algorithms need to be tested, like determining the size of the fruit. After that some more complex machine learning algorithms need to be tested, like a classification algorithm to classify the type of fruit and support vector machine (SVM) to determine the ripeness of the fruit. The software will be written in C++ to be as fast as possible and has to run online, in order for it to analyze the data directly after it is acquired and before new data is acquired.

In order to test the software architecture a setup needs to be made. The setup will consist of a hyperspectral camera sensor which will be connected with the EP-12. The EP-12 is a FPGA based board which is developed by 3D-One, which is a company of cosine. The EP-12 has 6 camera inputs and is capable of acquiring the RAW data of the camera, which will be sent to the Qseven. The Qseven is an embedded Linux based PC, which is capable of doing real time decision making. The software running on the Qseven is also developed by 3D-One.

As a test case tomatoes will be analyzed. The tomatoes need to pass by the hyperspectral sensor every second. A conveyor belt will be used which needs to be synchronized with the sensor. For this an optical switch can be used as a trigger.

2 Project organization

The internship will be done for the company cosine. The supervisor will be Chris van Dijk. cosine is a company which develops and builds measurement systems for use in scientific, industrial, medical, space and food applications.

The project is executed by Alex Veringmeier, a mechatronic engineering student from The Hague University of Applied Sciences Delft, as part of his graduation internship. The project will take place from 9 February 2015 until 5 June 2015.

At least the following deliverables need to be made:

- An internship work plan
- A plan of approach
- A planning
- A final report
- Basic design of the architecture of the software
- Description of the functioning of the system:
  - (SysML) diagrams of electronics, mechanics and software
  - Explanation of the diagrams
- Description of the test cases
- Documentation of the source code:
  - Comments describing the code
  - Description and diagrams of the code, doxygen can be used for this
- A branch in git for code revision

3 Requirements

There are multiple deliverables which should be realized. Every deliverable has its own requirements which will be validated at the end of the project. To describe how important every deliverable and its requirements are, the MoSCoW method will be used. The MoSCoW method classifies the requirements into must haves, should haves, could haves and wouldn't haves.

Must haves are requirements which need to be satisfied in order to make the system function properly. Should haves are requirements which have high priority but the system will function without them. Could haves are requirements that are nice to have but do not affect the performance of the system. Wouldn't haves are requirements which will not be satisfied.

3.1 Must haves

- Acquiring online hyperspectral data every second
- Preprocessing of the data, like:
  - Compensation for the sensor characteristics
  - Reflectance correction
- Analysis of the data within one second
- Synchronization of the conveyor belt with the sensor

3.2 Should haves

- Basic vision algorithms to analyze the data, like:
  - Determining the size
  - Determining the position
  - Determining the color
  - Determining artifacts
- Test how much analysis can be done within one second
- More complex algorithms to analyze the data, like:
  - Classification
  - Support vector machine

3.3 Could haves

- Output of the analysis as text on the image, like size, cultivar, ripeness
  Qt embedded frame buffer driver can be used for this
- Flexibility for more types of fruit

3.4 Wouldn't haves

- Development of the Qt embedded frame buffer driver

3.5 Boundaries

- The software has to be written in C++
- The EP-12 will be used to acquire the RAW data
- The software has to run on the Qseven
- The CMOSIS CMV4000 with an IMEC coated filter with a linearly variable filter (LVF) configuration will be used

4 Preliminary design

Before the realization of the project can start, some design choices need to be made. The primary functioning and components of the hardware and software need to be described. The preliminary design will define the overall system configuration, described with the use of schematics and diagrams. The preliminary design will describe the components of the hardware and the connection between them. Besides that, the essential functions and the architecture of the software will be described.

4.1 Hardware

For most of the components of the setup it was already decided which products would be used. This was mostly based on products that were already available. It was not necessary to make a design of the setup or to make choices about which products should be used. There will be described which components were needed for the setup, what function they have and how they are related to each other.

The setup can be separated into three parts, vision/optics, control and movement/transportation. Were the optics need to record the images, the transportation is used to move the object through the field of view of the camera and for synchronization and the control is used to control the process. A block definition diagram of these three parts and their components can be seen in Figure 8. The optics will consist of a light source and the camera. Where the camera can be further split down into the lens and the head, with in the head the sensor and a FPGA. The transportation consist of the conveyer belt and for the synchronization an optical switch will be used. The control can be split down into the hardware and the software. The hardware consist of the EP-12, Qseven and the SSD. The software consist of the source code, the kernel and some libraries.

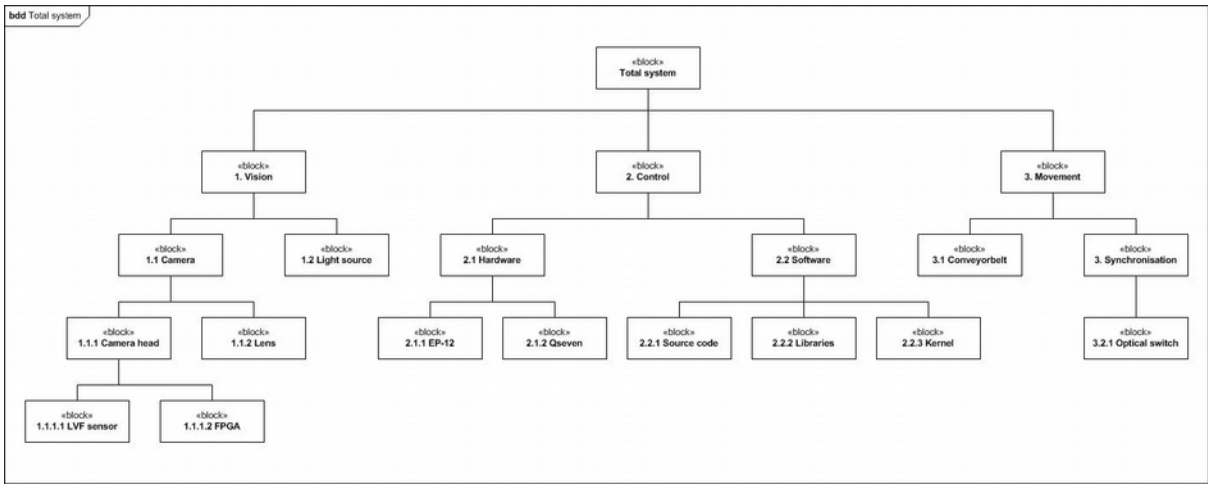


Figure 8: Block definition diagram of the total system

The way the components are connected can be seen in Figure 9. There can be seen what kind of connections are used between the components and in which way they transmit. The FPGA of the EP-12 is the central component which connect the camera and the trigger with the kernel. The kernel is connected with some software developed by 3D-One which is used to control the camera. The software of 3D-One is connected with the SSD in order to safe the images. In the executable is the pipeline and some libraries, where here only the Ebus SDK libraries is shown. The pipeline has a connection with the SSD in order to read the saved images. When the Keyence sensor get triggered it will send a signal to the FPGA of the EP-12 trough a CMOS connection. Where the FPGA will send it to the kernel with an interrupt and the kernel will send the signal to the software also with an interrupt.

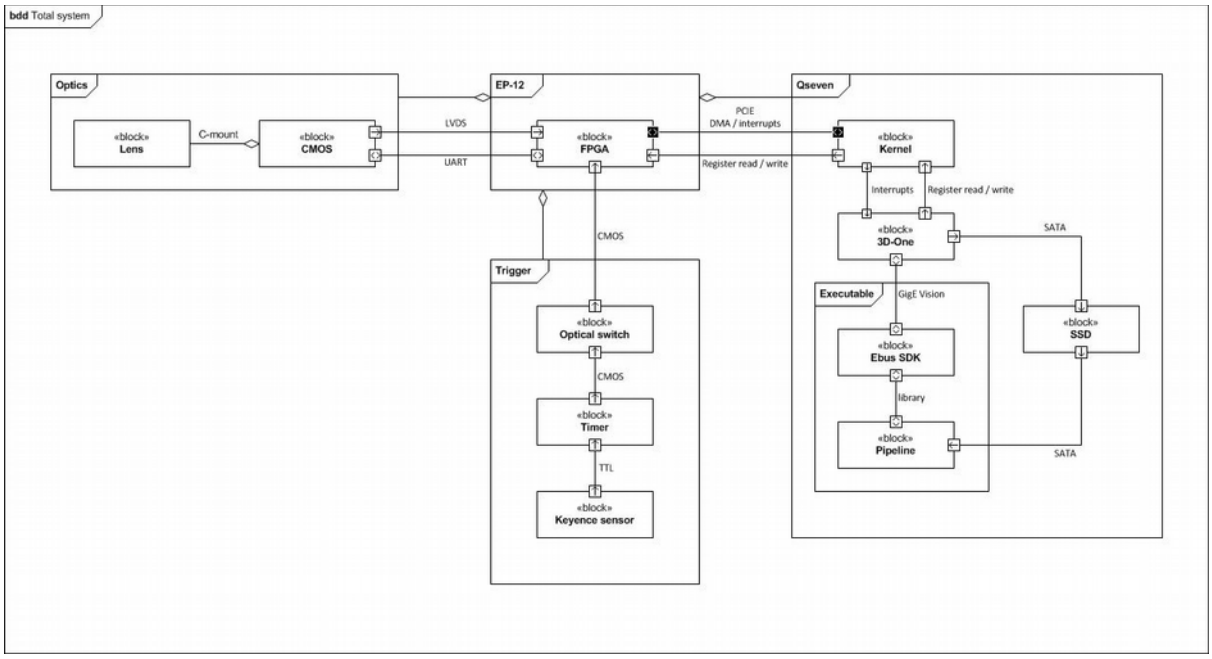


Figure 9: Block definition diagram of the connections between the components

4.2 Software architecture

The architecture of the software will describe the primary functions and behavior of the software. In order to analyze one tomato every second, the software has to run as fast as possible and finish the analysis within one second. Because of this the software need to be concurrent and run multiple threads in parallel. Also the communication between the threads need to be as fast as possible and the communication has to be lock free. The data needs to be passed on correctly between the treads and needs to be released well. Once a data object is created it needs to go through a chain of processing elements. This is called a pipeline.

4.2.1 Software design

The pipeline

The design of the pipeline is defined by the processes that need to be done. For every large task that needs to be performed a node will be implemented. Every node will perform a different task but will have the same external interface. In order to acquire the images and analyze the data a data object needs to go through the complete pipeline. On the next page in Figure 11 the flow of the pipeline and the order of the nodes and their tasks can be seen. The pipeline is constructed in the main function of the source code by connecting the output queue of one node with the input queue of another node. This is further explained in section 4.2.5, Queues. The architecture of the pipeline can easily be modified by changing the definition of the individual nodes. Depending on the needs of an application nodes can simply be added or removed from the pipeline.

Since the software is concurrent multiple nodes can be performed at the same time. Because of this images can already be processed while the camera is still recording and a new series of images already can be started while the previous series is still being processed. In Figure 10 a sequence diagram shows the expected order of the nodes. There can clearly be seen that some nodes are performed in parallel which saves time. Further there can be seen which node are the bottlenecks in the pipeline because the need to finish before any other node can go on.

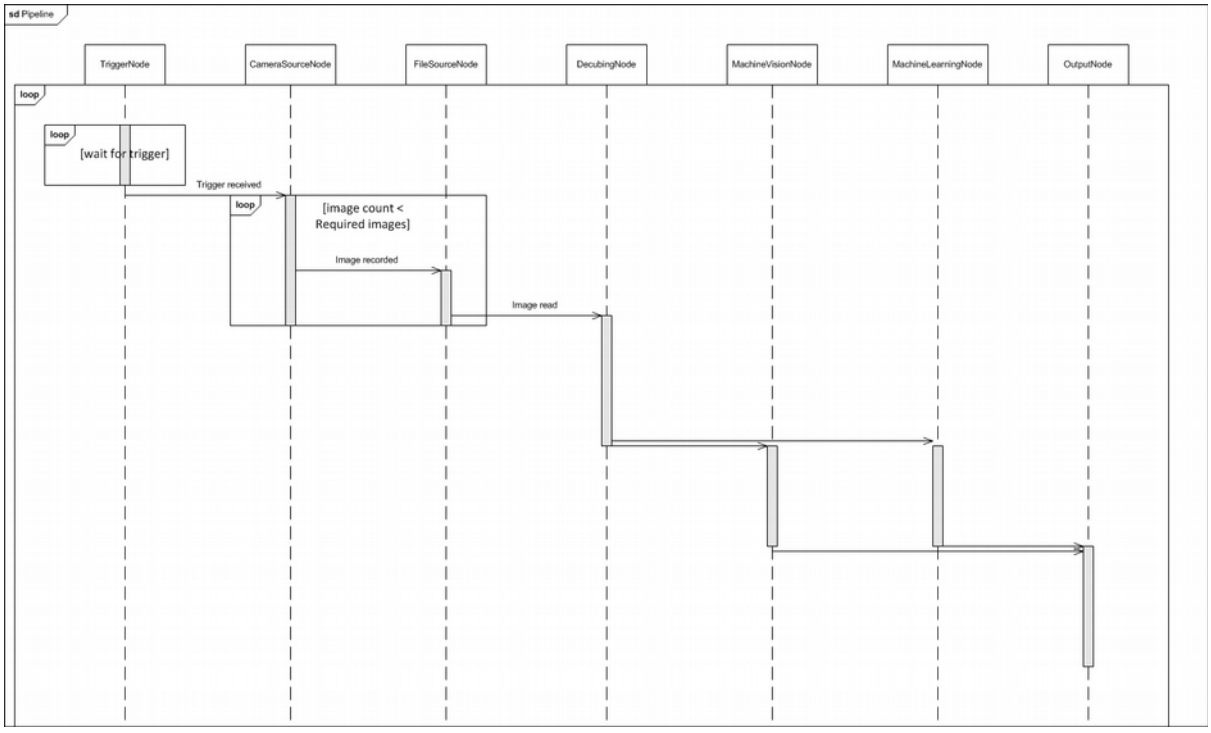


Figure 10: Sequence diagram of the pipeline

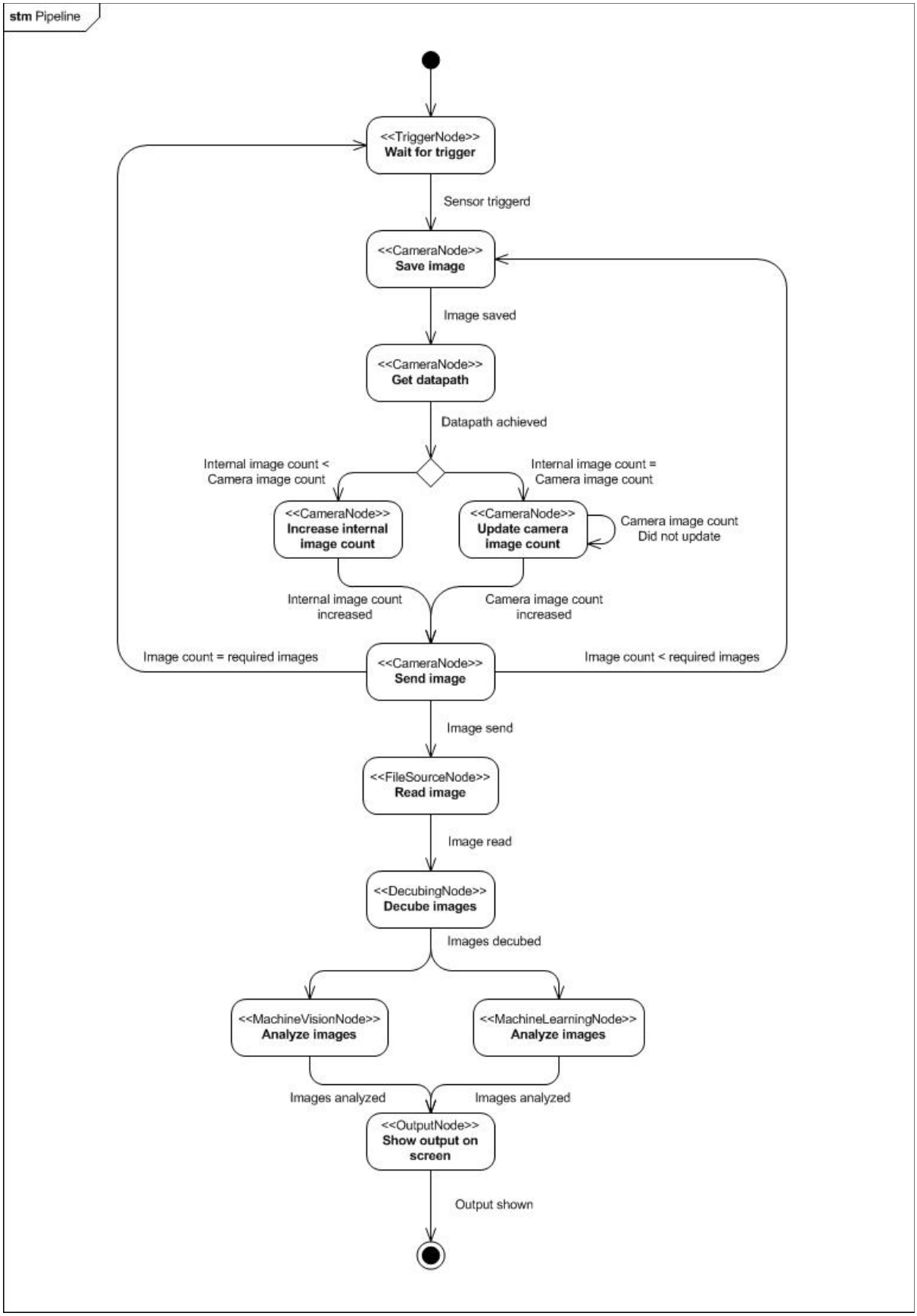


Figure 11: State machine of the pipeline

The software

When designing the software there was known what its general functions would be and what the behavior should be. The software needs to be capable to construct the pipeline and to run the different tasks of the pipeline in parallel. Besides this, there was taken into account to make the source code robust with a good readability and maintainability. The specific content of the nodes and the use of specific libraries or functions were mostly defined during the realization by trying different solutions and doing research which one would be the best solution.

The part of the software that is needed to be designed and realized is the complete pipeline. Other functionality of the software was already present. The Qseven has an operating system on it which is based on Tiny Core [6] and has extra functionalities added to it developed by 3D-One. The Qseven also already has a kernel, some user space and c binaries available on it. As a result of this it is possible to execute the source code on the Qseven. Besides this, everything that is needed to control the camera is also already available on the Qseven. On the Qseven is some code developed by 3D-One which makes use of the Pleora eBUS SDK [7]. This makes it possible to use the commands and functions of the GigE Vision library from Pleora which includes everything to acquire the images, control the camera and display the images on screen.

Next to the GigE Vision library some functionalities of other libraries are used. The libraries were not present on the Qseven by default and needed to be uploaded together with the source code. The libraries that are used are the armadillo library for use of linear algebra functions [8], the moodycammel library for the use of a fast lock free queue [9] and some of the Poco libraries [10] for the use of the logger and the configuration object. When compiling these libraries needed to be linked to the source code. The specific choices of the design of the software will be described below.

4.2.2 Non-virtual interface

In order to provide a robust code which can easily be reused and has a good readability it will be written following a design pattern. The design pattern that is used is the Template Method design pattern, often referred to as the Non-Virtual Interface idiom (NVI). The NVI idiom allows to restructure the code without changing its external behavior. The NVI idiom is based on four guidelines defined by Herb Sutter [11]. The guidelines are:

1. Prefer to make interfaces non-virtual, using Template Method design pattern.
2. Prefer to make virtual functions private.
3. Only if derived classes need to invoke the base implementation of a virtual function, make the virtual function protected.
4. A base class destructor should be either public and virtual, or protected and non-virtual.

Using this design pattern implements the external behavior of all the threads at one place, since this behavior should not be different. Which makes the code less complex and provide re-usability. This is described in the abstract base class. For every thread the abstract base class will be overridden, implementing the specific internal behavior of that thread but still using the same structure.

4.2.3 Abstract base class

The abstract base class is used to implement the basic structure and external behavior of the class. An abstract base class is a class that contains at least one or more pure virtual member functions. The destructor and the process function of this abstract base class are pure virtual. Derived classes provide the implementation of the pure virtual methods. The abstract base class will be called 'node', by overriding the abstract base class a specific node can be constructed. The abstract base class has a function called *runTask()* in which the task of the node will be performed. First the function *NodeInit()* will be called which is virtual. If it is overridden it will perform the specified initialization, otherwise it will return immediately. The nodes communicate using a message queue. The message queue functions as a buffer used to pass on data objects through the pipeline from one node to another. Whenever the node has an input queue it will try to get the first element out of the queue. The data from the input queue or a null pointer will be passed on to the *process()* function. This function will perform a node specific precess, implemented by the overriding class. The process function should return as fast as possible to overcome locking of the pipeline. After the process function there will be checked if there is an output queue and the result of the process function will be passed on to that output queue.

4.2.4 Overriding classes

The overriding classes will use the same structure as the abstract base class. Besides that, it will implement class specific functions. All overriding classes at least need to implement the pure virtual functions. It is optional to override any virtual functions. Besides that, specific functions can be created. Every specific task that needs to be done within the pipeline will be an overriding class.

4.2.5 Queues

A message queue is used to communicate and pass on data between nodes. For each node there will be an input queue and an output queue where the output queue of one node is connected to the input queue of another node. In this way one node can pass on data to the node which it is connected with. The order in which the nodes are connected to each other define the structure of the pipeline. The first node will not have an input queue, since the data object is created there. Additionally the last node will not have an output queue, because the data object is released there.

The queue is implemented to be a single-producer, single-consumer lock free queue. It uses the readerwriterqueue from the moodycamel library made by Cameron [9]. A template class is used to implement the functioning of the *try_enqueue()* and *try_dequeue()* functions which will respectively try to put a data object into the queue and try to get a data object out of the queue. The usage of the template class makes it possible not to specify the type of the data object when implementing. In this way any object type can be passed on by the queue, also more complex object types.

Whenever a node will try to read from an input queue which is empty or try to write to an output queue which is full it will block. A condition variable is used to wait in the try_dequeue function when the input queue is empty. The condition variable will be awakened whenever it receives a notify or when a timeout is expired. The try_enqueue function will call a notify_one when data is forwarded in the output queue to wake up the condition variable. The timeout of the condition variable is variable. In this way a specific timeout can be set for every node. Besides that, this supports the behavior for a node to have one input and multiple outputs. The timeout of the nodes will be set to a default value, where it will wait a while for any input before going on. Whenever an input is received it can set the timeout to zero and immediately go on whit processing when there is no input from the queue, since it is known that it should receive only one input.

4.2.6 Data handling

The data object

The data and messages that are passed on between the nodes are stored in a data object which is a class called NodeData. In the class the structure of the data object is implemented. The images are stored in a cube of the armadillo library [8], which is a C++ linear algebra library. The cube is a three dimensional array. When the data object is made a data type needs to be provided. The data type is an enum which can be rawdata_1d, rawdata_2d or rawdata_3d. By this the dimension of the data is specified. A single image will be 2D and only fills the first plane of the cube. When a series of images are combined they will form a 3D cube.

To store messages in the data object a Poco configuration object is used. In the configuration object variables can be created and the value can be set. On another point in the pipeline the value of the parameter can be read. In this way messages can be send between the nodes. Every class also has an integer which defines the id of the data object and which will increase every time a new data object is obtained. With the use of the id a data object can be tracked through the pipeline. The tracking can be used to see how long the data object takes to go through the pipeline.

Data handling

In the NodeData class are two public functions which handle the creation and clearing of the data object, called *obtain()* and *release()*. The way in which the data objects are created and used is called an object pool pattern. The object pool pattern makes it able to reuse the data objects. Which in this case is possible since the definition of the data object is always the same, only its content variates. The object pool pattern places the data object on a heap. The difference and advantage of the object pool pattern with a regular pool pattern is that a data object is created whenever there is none available. While whit a regular pool pattern the amount of objects need to be specified beforehand. The object pool pattern functions as a factory pattern as well, also referred to as a manager. Whenever the *obtain()* function is called there is checked if there is already a data object present in the cache. If this is the case the content of the data object will be cleared and it will be reused. When there is no data object available a new one will be created. The *release()* function does place the data object back on top of the cache. The benefit that the data object has its own manager is that it is thread safe. The different nodes can easily obtain or release a data object by using the functions of the data object itself. Because of this it is not needed for every node to have its own manager. The disadvantage of this is that it is not possible to have multiple managers. However in this case it is not necessary to have multiple managers.

5 Realization

5.1 Hardware

With the realization of the hardware all the components are put together to a test setup. On the conveyer belt a construction of aluminum profiles is made which carries the camera, EP-12 with the Qseven and the SSD. Also the optical fiber sensor is attached to the conveyor belt. The camera is attached in such way that it hangs in the middle of the conveyor belt and is looking down. With the aluminum profiles the height of the camera can easily be adjusted. The light source has its own tripod and is placed next to the conveyer belt aiming at the field of view of the camera.

5.1.1 Conveyor belt

The tomatoes in a greenhouse will most likely be transported by a conveyor belt. This is also very useful for the applications of analyzing the tomatoes, since the tomatoes need to move through the field of view of the camera because of the type of hyperspectral sensor that will be used. The conveyor belt that will be used for the test setup is a small conveyor belt of 15 cm by 100 cm. Driven by a SEW Eurodrive WF20 motor. The conveyor belt has a movement speed between 0.8 cm/s and 5.9 cm/s. The speed of the conveyor belt is regulated by a rotary knob. It does not have an encoder and there is no feedback of the speed. In order to determine the speed of the conveyor belt an object was placed on the conveyor belt. The object is moved by a set distance and the time it takes to move the distance was measured by a stopwatch. The speed can than be calculated by the following formula:

$$\text{speed} = \frac{\text{distance}}{\text{time}} \tag{1}$$

where the speed is in cm/s the distance is in cm and the time in seconds.

The speed of the conveyor belt needs to be synchronized with the recording speed of the camera in order to move one band in wavelength per image. For this reason the speed at which the conveyor belt needs to move during recording is defined by the frames per second (fps) of the camera, the field of view and the amount of bands of the hyperspectral sensor. This can be seen in Formula 2:

$$\text{speed} = \frac{\text{camera speed} * \text{field of view}}{\text{bands of the sensor}} \tag{2}$$

where the speed is in cm/s the camera speed is in fps the field of view is in cm.

Since the frames per second and the amount of bands are given by the characteristics of the camera and the sensor, the field of view is the only variable which can be adjusted to define the speed of the conveyor belt. The speed of the conveyor belt needs to be fast enough to transport one tomato per second. Although it is not preferred to move to fast since this can cause blur in the images. However, since the maximum speed of the conveyor belt is 5.9 cm/s and a beefsteak tomato is about 10 cm in size it is not possible with this conveyor belt to transport one tomato every second. For this reason the field of view will not be adjusted to match the speed of one tomato per second but the speed of the conveyor belt will be adjusted to match the field of view. Besides that, the field of view affects the speed of the conveyor belt it also affects the resolution of the image. The resolution is defined by the area that is visible per pixel or band in the image. The resolution can be calculated by the next formula:

$$\text{resolution} = \frac{\text{field of view}}{\text{amount of bands}} \tag{3}$$

where the resolution is in cm/band and the field of view is in cm.

Because of this it is preferred to have a field of view as small as possible since this results in a higher resolution. However the field of view cannot be smaller than the size of a tomato. For this reason it is chosen to have a field of view slightly larger than the width of the conveyor belt. Since the conveyor belt has a width of 15 cm it is assured that an average tomato will be completely visible.

The field of view of the camera can be adjusted by the distance from the camera to the conveyor belt and the zoom level of the lens. The lens will not be zoomed. The camera is mounted on a construction of aluminum profiles such that it can easily be adjusted to the right height. In Figure 12 the construction can be seen.

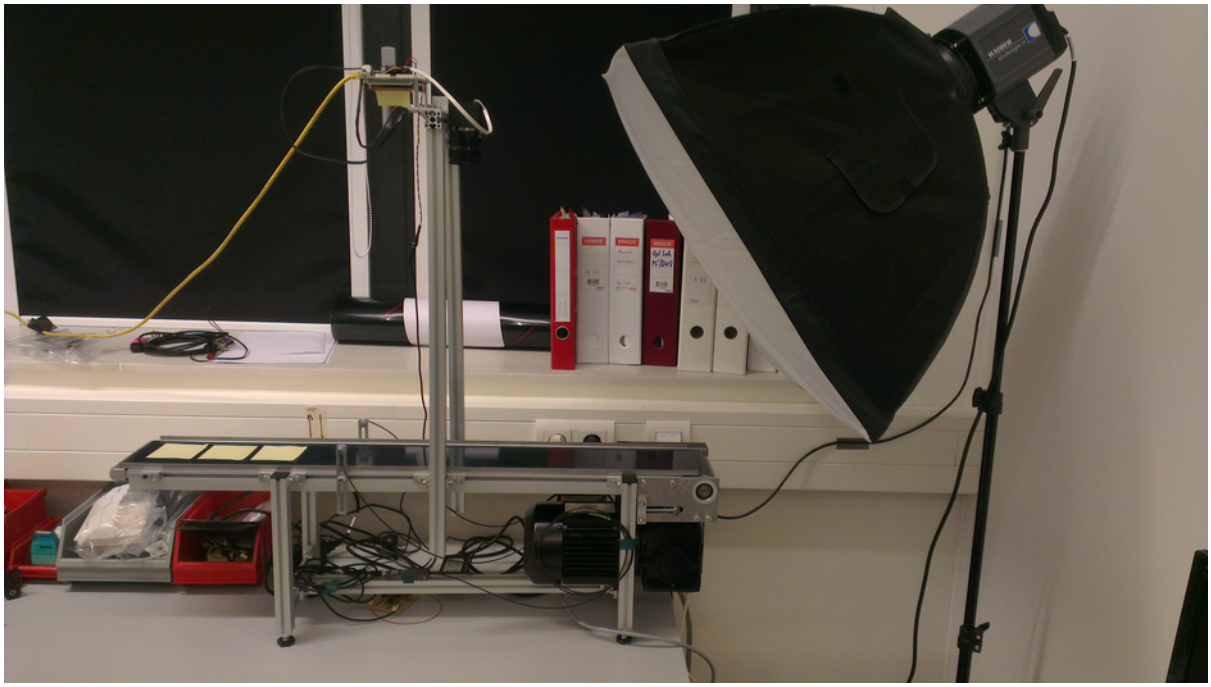


Figure 12: Setup with the conveyor belt, the camera and the light source

On the picture of the setup the conveyor belt can be seen, with the construction of aluminum profiles attached to it. The camera, the EP-12 with the Qseven and the SSD are connected to the aluminum profiles above the conveyor belt. The light source is standing next to the conveyor belt on its own tripod.

The height of the camera is adjusted in such way that the field of view is slightly larger than the width of the conveyor belt. This is set by looking at the image of the camera and adjusting it such that the conveyor belt with some extra space next to it is visible in the field of view. This results in a height of 50.2 cm from the lens to the conveyor belt. When the height of the camera is set the field of view is measured by putting down a ruler in the field of view and reading the measurement. In the current setup the field of view is 15.5 cm. Based on the set field of view the resolution of the setup can be calculated by the following formula:

$$\text{resolution} = \frac{15.5\,cm}{100\,bands} = 0.155\,cm/band \tag{4}$$

Since the maximum speed of the camera is 15 fps and the sensor has one hundred bands the speed of the conveyor belt ends up to be:

$$\text{conveyor belt speed} = \frac{15\text{ fps} * 15.5\,cm}{100\,bands} = 2.325\,cm/s \tag{5}$$

5.1.2 Synchronization

In order to synchronize a tomato on the conveyor belt with the recording of the camera a transmissive fiber sensor is used. The sensor is a Keyence FU-77V sensor. In Figure 13 the sensor can be seen.

Figure 13: Keyence FU-77V transmissive fiber unit

The sensor works like a light gate. One end of the sensor will transmit a signal which will be received by the other end. The sensor will trigger as soon as something is between the two ends of the sensor. The sensor will send a signal through an electrical circuit which will send a signal to the EP-12. The EP-12 will handle the signal and the camera will be triggered to start recording. When the timing of the trigger is correct the tomato will be exactly on the edge of the field of view of the camera when the recording starts. If the speed of the conveyor belt is adjusted right, the tomato will move through the entire field of view of the camera with a speed of one band per image, as described in 5.1.1, conveyor belt.

5.1.3 Camera

The camera that will be used for the setup consist of different components composed for the project. The most important component of the camera is the sensor. The sensor is a CMOSIS CMV4000 sensor with an IMEC coated filter with a linearly variable filter (LVF) configuration. In Appendix 1 the specifications of the sensor can be seen. The sensor is a hyperspectral sensor where the center wavelength of the filter varies linearly across one axis of the sensor, such that the sensor has multiple bands which image different wavelengths, as described in 1.1, background.

The sensor is put on a FPGA which processes the data from the sensor and record it as images. The FPGA with the sensor are placed in a housing. The FPGA is connected with the EP-12 with a LVDS connection. The housing of the camera has a C-mount on which a FUJINON CF35HA-1 lens is connected. The specifications of the lens can be found in Appendix 2

5.1.4 Light source

The light source that will be used needs to transmit a light as equally spread as possible and it should have a broad flat light spectrum. The normal room light, which are fluorescent tubes should not be used since their spectrum has a lot op peaks and is not broad and flat. The spectrum of a fluorescent tube can be seen in Figure 14 A large Kaiser studio lamp was available and was an excellent possibility. The studio lamp has a halogen lamp which source is very bright and has a nice broad and flat spectrum. The spectrum of the halogen lamp can be seen in Figure 15 The lamp has a large shade around it which completely enclose the lamp and a canvas is hanging in front of the lamp such that the light is refracted. The lamp has its own tripod and can easily be adjusted in height and angle. When images are recorded the regular lights in the optics lab would be turned of. The windows of the optics lab are blinded so there is no distortion from sunlight.

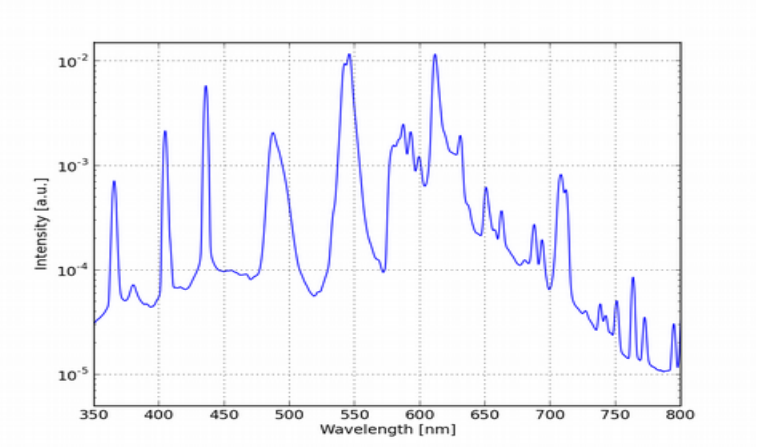


Figure 14: Spectrum of a fluorescent tube

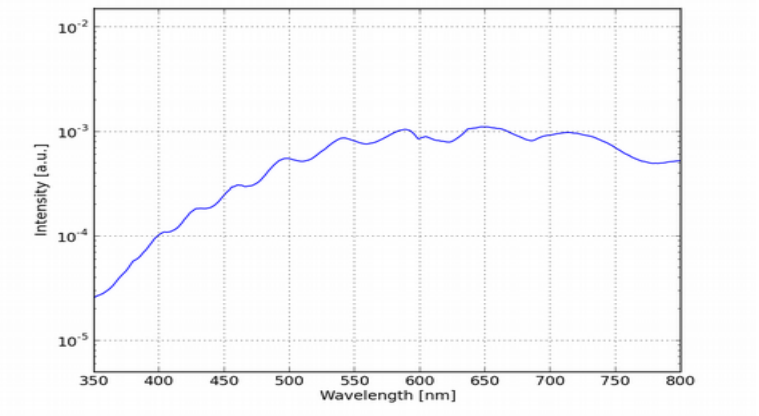


Figure 15: Spectrum of a halogen lamp

5.2 Software

With the realization of the software, the behavior and functions of every specific node in the pipeline got implemented. The main tasks of the pipeline are to give the camera the command to start recording, manage the recorded images which are saved to disk, load the files from disk, preprocess the files, analyze them with vision and machine learning algorithms and show the output on screen.

5.2.1 Camera triggering

As described in the hardware section, an optical fiber sensor will be used for triggering. The EP-12 will process the signal of the optical fiber sensor. The first node of the pipeline will check in a loop if the EP-12 received an input signal corresponding to the trigger. When there is an input signal the TriggerNode will create a data object and set a trigger variable to true. The data object will be passed on to the next node through the message queue. While testing a simple version of the TriggerNode is used, which just send a data object through the message queue every 20 seconds.

5.2.2 Image recording

Start recording

The streaming of the images gets handled by the CameraSourceNode. Before the camera can start streaming, it needs to be initialized. The *nodeInit()* function of the abstract base class will be overridden to implement the initialization. In the initialization some parameters of the camera need to be set, for instance the trigger interval, the auto exposure and the shutter time. The value for every parameter is read from a configuration file. In this way the values can be changed in the configuration file, without having to compile the code.

In the abstract base class the CameraSourceNode checks if there is any input from the TriggerNode. The timeout of the input queue has a default value of 500 milliseconds. When a messages is received the camera needs to start recording. After triggering, the *process()* function of the CameraSourceNode needs to be called continuously in order to create multiple data objects and move them to the output queue as fast as possible. For this reason the timeout of the input queue is set to zero. In this way the Node supports single input, multiple output behavior as explained in section 4.2.5, Queues.

In order to track the state of the camera, an enum is created which consist of the different states the camera can be in. A switch case handles the different states and different behavior will be performed corresponding to the matching state. In Figure 16 a state machine diagram shows the different states the camera can take and the conditions that are required to change the state.

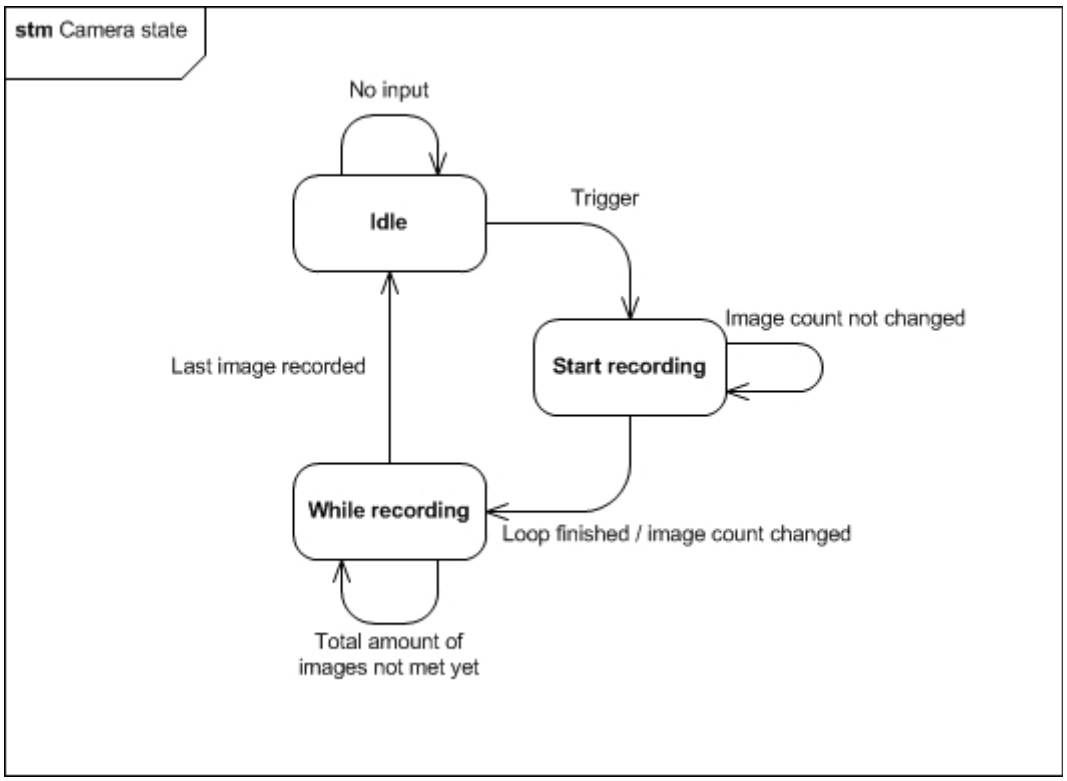


Figure 16: State machine diagram of the states of the camera

Whenever the camera is not recording or triggered yet the state is idle, which is the default state. The process function will return a nullpointer whenever the state is idle. As soon as the CameraSourceNode receives a message from the TriggerNode the state will be set to start_record. In this state the *startRecord()* function will be called. In the *startRecord()* function the command RecordStreamStart will be send to the camera. After that the data path where the images get saved will be asked from the camera. Also the image count of the camera gets asked. The image count is used to keep track of how many images are saved. It might take a few hundred milliseconds before the image count gets updated. Because of this the image count gets asked twice to make sure the image count is updated and not the value of the previous session is taken. If the *startRecord()* function returns successfully the state will be set to recording and a data object will be created. The data path and the image count will be set in the message of the data object and the object will be passed on to the output queue.

While recording

While the state of the camera is recording, there will be checked if the image count already reached the required amount of images. If this is not the case the function *whileRecord()* will be called. The required amount of images will be gathered from the configuration file, which makes it able to change it without compiling the code. The *whileRecord()* function is a slightly shorter version of the *startRecord()* function. It does not need to give the RecordStreamStart command since the camera is already recording. The function will ask for the data path and the image count. Since the image count of the camera gets cached there will be checked if the internal image count is corresponding to the camera image count. If this is the case the camera image count needs to be updated. Whenever the internal image count is lower than the camera image count it is not needed to ask the camera for its image count and the internal image count will be increased. When the *whileRecord()* function returns successfully a new data object will be created with the data path and the image count as message and it will be passed on to the output queue. This will continue until the required amount of images are recorded. Once this happens the command RecordStreamStop will be given and all the parameters will be set to default. The image counts will be set to zero, the timeout of the input queue will be set to 500 milliseconds and the state to idle. In Figure 17 a state machine diagram can be seen which shows the entire flow of the CameraSourceNode.

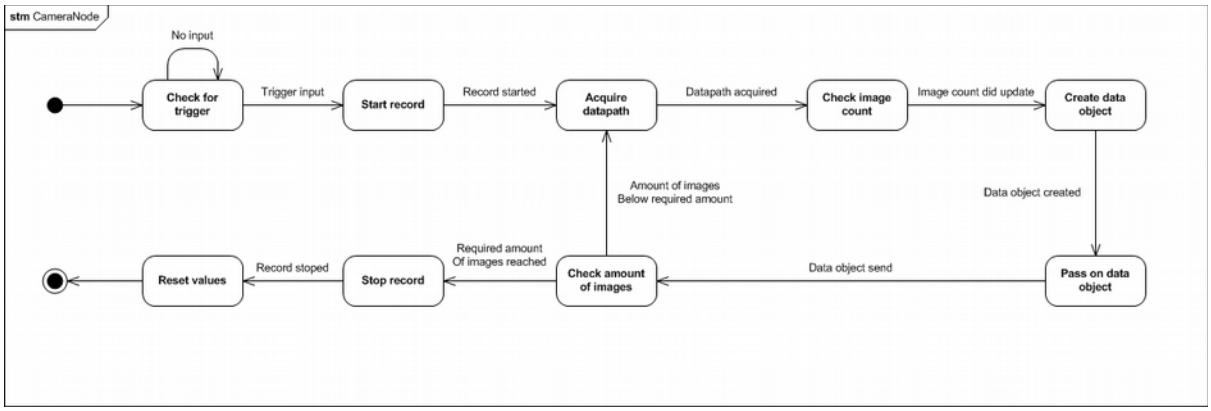


Figure 17: State machine diagram of the CameraSourceNode

5.2.3 File reading

The camera will save the images on a SSD. The files need to be loaded from the disk in order to get access to the images for processing. Every time the CameraSourceNode sends a data object to the message queue, the FileSourceNode will read this data object from the queue. The data path and the image number can be read from the message of the data object. In this way the FileSourceNode will start reading a file as soon as the data object is passed on. The CameraSourceNode and FileSourceNode will run in parallel such that the FileSourceNode already can read the images while the CameraSourceNode is still streaming. The data path from the message will be cast to a Poco Path and the image number will be extended with zeros until a total of nine numbers. At the end the string '.raw' will be added. In this way the file that needs to be loaded is specified. The data of the image will be loaded to the data variable of the same data object that was passed on by the CameraSourceNode. If this succeed the data object will be passed on to the next node.

5.2.4 Decubing

Once the images are gathered they can be used for analysis. Since a specific type of sensor is used, where one image consist of multiple bands all of a different wavelength, it is needed to reorder the images in order to get images that show only one wavelength. This is called decubing. The DecubingNode did not get implemented, but it is known what its behavior should be. There will be described how this presumable could be implemented. The DecubingNode needs to process multiple images before a datacube can be passed on to the next node. As a result, the DecubingNode has the behavior of a multiple input, single output node. Once the first image gets in, a new data object should be created with the data type *rawdata_3d* as described in 4.2.6, Data handling. The incoming data objects can be released once they are processed. Every band of all the images with the same wavelength need to be placed next to each other, such that a new image is formed consisting of only one wavelength. Because of the moving of the object a different part of the object will be gathered by every band in every new image. To correctly perform the decubing some linear algebra function need to loop through the images, such that the functions go diagonal through the images.

5.2.5 Machine vision

When the decubing is done the data cube can be used for analysis of the object. Also the nodes that will perform analysis did not get implemented. The node which will perform machine vision algorithms can be a single input, single output node, where the data cube gets passed on with some extra information about the tomato in the message of the data object. It is also possible to let the node create new data objects with edited single images which show specific characteristics of the tomato. A few simple machine vision algorithms will be used in this node to gather general information of the tomato. For instance the size and the position of the tomato. Also characteristics like the average color and the center can be collected. The values of these characteristics can be added to the message of the data object.

When some more complex machine vision algorithms are performed it is needed to create new data objects where the results of the analysis is a single image which will be added to the data variable of the data object. The extra information that is available because of the different wavelengths can be used to acquire specific characteristics of the tomato. For instance, images with two different wavelengths can be divided over each other which can result in a good contrast between the tomato and artifacts when the right wavelengths are chosen. The result of this will be a single image from which the size and the position of a potential artifact can be acquired.

5.2.6 Machine learning

Next to the machine vision algorithms it is also possible to gather information of the tomato by using machine learning algorithms. Machine learning algorithms are more complex but will be able to acquire different characteristics, like the ripeness of the tomato. The machine learning algorithm node did not get implemented as well. To pass on the data cube to the machine learning node it is possible to pass it on from the machine vision node instead of releasing it there. But it would be better to pass it on straight from the decubing node. In order to do this it is needed to copy the outcome of the decubing node and send it to two output queues. Better would be to create a specific node which is capable of making copies of data objects and pass it on to two ore more nodes. In this way the copy node can be used at other points in the pipeline as well.

There are different kind of machine learning algorithms. There are two algorithms that are probably the most useful for analyzing tomatoes, classification and support vector machine (SVM). A different node should be created for each algorithm. In the classification node a data set of different cultivars of tomatoes will be used to learn to distinguish between the cultivars. Previous research [4] shows that the cultivars clearly can be separated by using classification, see 1.1, Background. The result of the classification will be the name of the type of tomato cultivar, which can be added to the message of the data object. Once the classification is done the result can be passed on to the next node.

Another interesting feature of a tomato is its ripeness. It is a lot more complex to determine the ripeness of a tomato. Probably SVM can be used for this. The behavior of the SVM node would be the same as with the classification node. Where the optimal result would be the age of the tomato in days. The result also can be saved in the message of the data object and will be passed on to the next node.

5.2.7 Output

The last task of the pipeline is showing the output of the analysis on screen. Also the OutputNode did not get implemented. The OutputNode needs to be a multiple input, single output node, since the output will be passed on by both the machine vision node and the machine learning node. Once both data objects are received the output can be shown on screen. This can be simple the text of the result of the analysis from the message of the data object. It is also possible to show an image with the text overlaying on the image. However this would probably not add much interesting information since most images would be very similar for most tomatoes. After the output is shown the data object can be released since there is not another node after the OutputNode.

6 Results

Once the setup and the software is realized the performance needs to be tested. Since the software needs to run as fast as possible in order to analyze one tomato every second the timing is the most important performance to test.

6.1 Timing

The timing of the code is very important since the setup needs to be capable to analyze one tomato every second. However with the current setup this is not possible. The maximum speed of the conveyor belt is not fast enough to transport a tomato per second. Besides this, also the camera cannot record fast enough to capture a full tomato in a second. The tomato needs to move through the entire field of view in order to acquire information of the tomato in every wavelength. Since the sensor has one hundred bands and the whole tomato needs to move through every band with steps of one band per image two hundred images need to be taken to capture a full tomato. Because the camera has a frame speed of 15 fps this would take 13.3 seconds. However even though the hardware is not capable to capture a tomato every second the true industrial scenario will be much quicker and the software should still be capable to do the analysis within one second.

In order to measure the timing of the software the logger of the Poco library is used. The logger makes it possible to log messages with a certain priority level. These messages can be read during execution or after execution in a log file. With the use of the logger debugging information and error information can be reported. It is possible to let the logger also show the time of the processor with each message. The time of the logger has a microseconds accuracy which is perfect to do accurate time measurement. At interesting points in the pipeline a message is logged which makes it possible to see the time on the moment the message gets logged. The messages that will be logged shows information of the point in the pipeline at which it did log. For example at the beginning and ending of every node. Besides this, also the id of the data object gets logged with the message. Because of this it is possible to track a data object through the pipeline and see how long it takes to complete every step and the total time to go through the pipeline.

To measure the time, the pipeline was executed once where it did record 50 images. When the time of the first log message of the first image is subtracted from the time of the last log message from the last image the difference is rounded 3.62 seconds. To see how long it would take to record 200 images the time is multiplied by four which ends up to be 14.5 seconds. Which is slightly larger than the 13.3 seconds which it should be when 200 images are recorded with a record speed of 15 fps. This means that there is still a very small delay in the process of every image which adds up to a bit more than one second when 200 images are recorded.

To analyze how long it takes for each node to process and how long it takes for a data object to go through the pipeline, the time values are put in a table. This is done for 20 data objects. The first time value is subtracted from every other value in order to let the time start at zero and see the proceeding of the time from that point on. In Table 1 the timing of 10 data objects can be seen with the ids of 20 to 30.

	CameraSource node				FileSource node				Inspection node				Sink node			
Node	start Record	data created	end process	outq	inq	start process	end process	outq	inq	start process	end process	outq	inq	start process	data released	Total time
20	0.000000	0.000460	0.000570	0.000670	0.013180	0.013280	0.059650	0.059940	0.060160	0.060300	0.087030	0.087130	0.087210	0.087290	0.087370	0.087370
21	0.152060	0.152320	0.152510	0.174140	0.174140	0.174340	0.198130	0.198360	0.198610	0.198810	0.251580	0.251680	0.251760	0.251820	0.251900	0.099840
22	0.152750	0.254560	0.254670	0.254740	0.254860	0.255020	0.266240	0.266330	0.266440	0.266500	0.294510	0.294600	0.294680	0.294740	0.294820	0.142070
23	0.254810	0.255260	0.255380	0.255440	0.266400	0.266490	0.276030	0.276170	0.295060	0.295150	0.319650	0.319750	0.319830	0.319890	0.319970	0.065160
24	0.255510	0.356290	0.356400	0.356460	0.356560	0.356670	0.365430	0.365520	0.365610	0.365670	0.395380	0.395470	0.395560	0.395620	0.395700	0.140190
25	0.356540	0.357000	0.357110	0.357190	0.367340	0.367440	0.391470	0.391620	0.395830	0.395890	0.422390	0.422480	0.422570	0.422630	0.422710	0.066170
26	0.357300	0.458390	0.458640	0.458840	0.459130	0.459490	0.464650	0.464790	0.464910	0.464970	0.497200	0.497310	0.497400	0.497460	0.497540	0.140240
27	0.459070	0.560680	0.560840	0.560920	0.561060	0.561160	0.570650	0.570800	0.570940	0.571090	0.597100	0.597210	0.597300	0.597370	0.597450	0.138380
28	0.561030	0.561520	0.561630	0.561720	0.570890	0.570980	0.582580	0.582700	0.597600	0.597690	0.626980	0.627080	0.627170	0.627230	0.627330	0.066300
29	0.561810	0.662710	0.662820	0.662880	0.663020	0.663180	0.709950	0.710040	0.710130	0.710370	0.736210	0.736300	0.736390	0.736450	0.736530	0.174720
30	0.662960	0.764010	0.764120	0.764180	0.764280	0.764400	0.774440	0.774540	0.774650	0.774710	0.800760	0.800860	0.800940	0.801000	0.801070	0.138110

Table 1: Timing of the nodes in the pipeline

In the table the different places in the pipeline where a message did log can be seen. Also the timing of every node and the total time of one data object to go through the pipeline can be seen. For every node a message get logged directly when the node starts, right before the input queue which is indicated with inq, at the start of the process function where the node specific task is performed, at the end of the process function and at the end of the node just before the output queue would send the data object which is indicated with outq. The SinkNode does not have an output queue since it is the last node of the pipeline. Here the time when the data got released can be seen.

The total time of each data object differs quite a lot with values between 0.06 and 0.15 seconds. The average total time of the 50 data objects is 0.12 seconds. The average time for each node to complete is 0.067 seconds for the CameraSourceNode, 0.02 seconds for the FileSourcenode, 0.03 seconds for the InspectionNode and 0.0002 seconds for the SinkNode. It is very clear that the CameraSourceNode takes the longest and is the bottleneck in this part of the pipeline.

This measurement only shows the timing of the realized part of the pipeline. It would be necessary to measure the timing of the other parts of the pipeline as well in order to see how well the pipeline really performs. In that case the different parts of the pipeline could be compared with each other and there could be seen where the bottleneck is.

7 Conclusion

With the use of the results and the progress of the realization a conclusion about the project can be made. First there can be concluded that the initial goal of the project to analyze tomatoes was not achieved. Due to time issues the nodes that should process and analyze the images did not get implemented. Although the realization of the pipeline did start relatively soon and there was a continuous work flow it was still not possible to realize the analysis of the tomatoes. Since the time of the project was efficiently spent there can be concluded that the realization of the structure of the pipeline was underrated and that the initial goal of the project was to much to be finished within time.

Even though not the complete goal was achieved, the part that did get finished was performed well. The pipeline that did get implemented has a nice structure and behavior. The use of the abstract base class makes it possible to implement specific tasks in the pipeline while still using the same structure. It is very easy to create new nodes and it simple to add the to the pipeline. The data gets efficiently passed on between the nodes and the nodes can easily pass on messages to other nodes. The data is obtained and released right and with the use of the object pool pattern there is an efficient use of the data objects and not every node needs its own manager. The nodes take ownership of the data object once passed on with the use of the unique pointer and when the data object is released at the end of the pipeline there will be no memory leak.

Also the recording part of the pipeline was implemented well. It is possible to wait for a trigger and once triggered to start recording. The CameraSourceNode handles the recording of the images correctly. It is possible to have nodes with a single input single output behavior, to have nodes with a single input and a multiple output behavior and to have nodes with a multiple input single output behavior. The images are read from the SSD and after that it is possible to process the images. With the use of the armadillo cube it is possible to have a single 2D image in a data object as well as a 3D cube of multiple images.

With the use of the measurements also the timing of the pipeline can be concluded. The time the recording of the images takes is slightly slower, with 10 percent of the total amount of time it should take. It is not possible to say how much analysis can be done within one second since the analysis did not get implemented.

8 Recommendation

Based on the experience gained some recommendations will be given for the continuing of the project. Now that the pipeline is realized it is recommended to continue the project by implementing the processing of the images and vision algorithms to analyze tomatoes. First the decubing of the images needs to be implemented in order to do any analysis. After that some basic machine vision algorithms and some machine learning algorithms can be implemented. For the new parts of the pipeline the nodes can be used and in this way the parts can simply be added to the pipeline.

When the pipeline is complete it is recommended to look further at the timing of the pipeline to see how much analysis can be done within one second. It would be interesting to compare the timing of the recording, the decubing and the analysis. The results of the timing can be used to determine the bottleneck of the pipeline and this can be used to increase the time needed to go through the pipeline. When the analysis requires to much time it is recommended to look at a splitter node which is capable to split the data and pass it on to two nodes which perform the same task. In this way the task that is the bottleneck of the pipeline can be performed performed in parallel for two different series of data. This would not improve the total time that is needed to go through the pipeline but it would increase the throughput of the pipeline since two data series can be analyzed at the same time.

When continuing the project it is also recommended to secure a simple demo setup first which can be used to show the recording and processing of the tomatoes. This demo should be simple to start such that it is easy to show the functioning to customers or employees who want to get an understanding about the technique. From that point on the pipeline can be expanded with more analyzing algorithms which improve the functionality of the demo.

At last it is recommended to do further research to which machine learning algorithms can be used to analyze the tomatoes. It would be useful to know which algorithms are available and to know which ones can be used to analyze the tomatoes and which ones add new interesting information about the tomatoes. It would also be needed to find out which machine learning libraries are available for C++ and what their advantages and disadvantages are.

References

[1] http://commons.wikimedia.org/wiki/File:Cone-fundamentals-with-srgb-spectrum.svg, last seen on 04-06-2015
[2] IMEC 2014 Activity Overview, *personal communication*
[3] http://commons.wikimedia.org/wiki/File:AcquisitionTechniques.jpg, last seen on 04-06-2015
[4] cosine internal study
[5] S. Kawano, H. Wantanabe, M Iwamoto, J. Japan Soc. Hort. Sci. 61 (2), 445, 1992
[6] http://distro.ibiblio.org/tinycorelinux/, last seen on 04-06-2015
[7] http://www.pleora.com/our-products/ebus-sdk, last seen on 04-06-2015
[8] Armadillo, a C++ linear algebra library, http://arma.sourceforge.net/, last seen on 04-06-2015
[9] Cameron, A Fast Lock-Free Queue for C++, 2013, http://moodycamel.com/blog/2013/a-fast-lock-free-queue-for-C++, last seen on 04-06-2015
[10] http://pocoproject.org/, last seen on 04-06-2015
[11] Herb Sutter, Virtuality, September 2001, http://www.gotw.ca/publications/mill18.htm, last seen on 04-06-2015

Books
Martin Reddy, API design for C++, ISBN: 978-0123850034, 2011
Bjarne Stroustrup, The C++ programming language, fourth edition, ISBN: 978-0321563842, 2013

Appendix 1: Specifications of the hyperspectral sensor

CMV4000

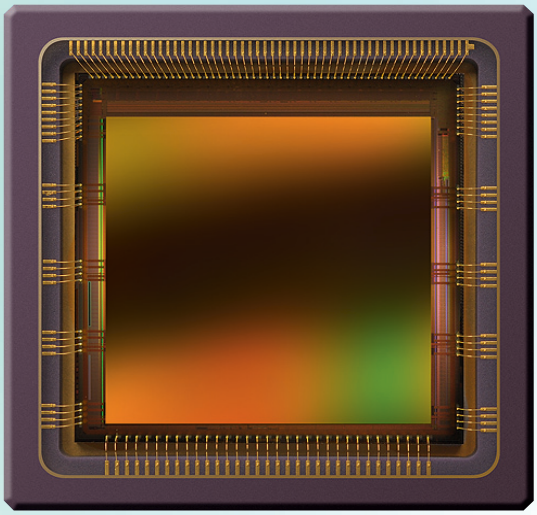
4MP high speed global shutter image sensor

SENSOR DESCRIPTION

The CMV4000 is a global shutter CMOS image sensor with 2048 by 2048 pixels in a 1" optical format. The image array consists of 5.5 um by 5.5 um pipelined global shutter pixels, which allow exposure during read out while performing CDS operation reducing fixed pattern and dark noise significantly. The CMV4000 has 16 12-bit digital LVDS outputs (serial) each running at 480 Mbps. The image sensor also integrates a programmable gain amplifier and offset regulation. Each channel runs at 480 Mbps maximum, which results in 180 fps frame rate at full resolution in 10-bit mode. Higher frame rates can be achieved in row-windowing mode or row-subsampling mode. All operation modes are all programmable using a SPI interface. A programmable on-board sequencer generates all internal exposure and read out timings. External triggering and exposure programming is also possible. Extended optical dynamic range can be achieved by multiple integrated high dynamic range modes. A 12-bit per pixel mode is available at reduced frame rates.

APPLICATION FIELDS

- Machine vision
- Motion control
- Traffic monitoring
- High speed inspection
- Security



SENSOR FEATURES

- Pipelined global shutter with CDS
- 2048 (H) * 2048 (V) active pixels on a 5.5 µm pitch
- Optical format of 1"
- 180 frames/s at full resolution in 10 bit mode
- 37 frames/s at full resolution in 12 bit mode
- ROI windowing capability
  (up to 8 separate ROIs - row based only)
- X-Y mirroring function
- 16 LVDS-outputs @ 480 Mbps multiplexable
  to 8, 4 and 2 at reduced frame rate
- Multiple High Dynamic Range (HDR) modes up to 90 dB
- On chip temperature sensor
- On chip timing generation
- SPI-control
- 3.3V and 1.8V signaling
- Monochrome and Bayer (RGB) configuration
- Ceramic 95-pins µPGA package (18.6 mm x 18.6 mm)

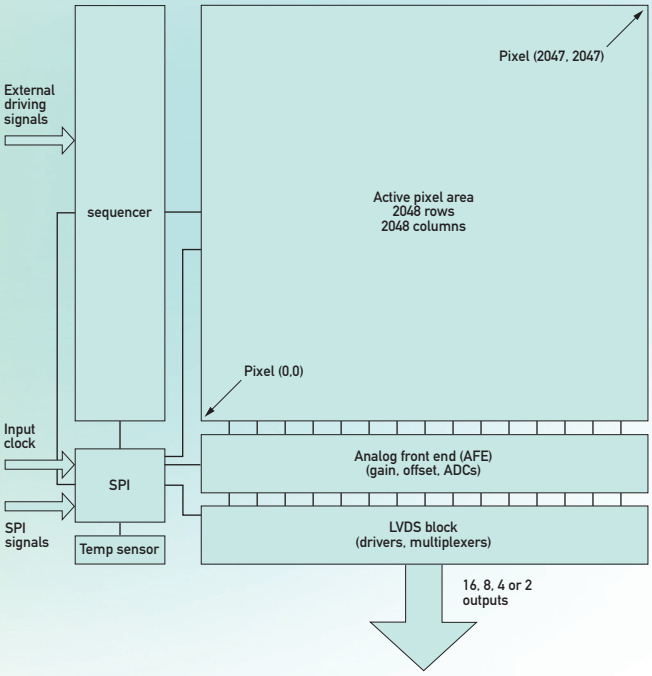
Please address all product inqueries and ordering information to:
CMOSIS NV · Coveliersstraat 15 · B-2600 Antwerpen · Phone: +32 3 260 17 33 · Fax: +32 3 260 17 79 · info@cmosis.com

CMV4000

4MP high speed global shutter image sensor

SENSOR SPECIFICATIONS

Specification	Value
Part status	Production
Resolution	4MP - 2048(H) x 2048 (V)
Pixel size	5.5 x 5.5 µm2
Optical Format	1''
Shutter Type	Pipelined global shutter with true CDS
Frame Rate	180 fps (10 bit) 37 fps (12 bit)
Output Interface	16 LVDS outputs @ 480 Mbps
Sensitivity	4.64 V/lux.s
Conversion gain	0.075 LSB/e-
Full well charge	13500 e-
Dark noise	13 e- (RMS)
Dynamic range	60 dB
SNR max	41.3 dB
Parasitic light sensitivity	1/50000
Extended dynamic range	Yes, up to 90 dB
Dark current	125 e-/s (25 degC)
Fixed pattern noise	< 1 LSB (<0,1% of full swing)
Chroma	Mono and RGB
Supply voltage	1.8V / 3.3V
Power	600 mW
Operating temperature range	-30 to +70 degC
RoHS compliance	Yes
Package	95 pins uPGA



ORDERING INFORMATION

Product number	Description
CMV4000-3E5M1PP	Monrochome version with micro lenses
CMV4000-3E5C1PP	RGB Bayer color version with micro lenses
CMV4000-3E12M1PP	NIR enhanced sensitivity

Appendix 2: Specifications of the lens

Applicable camera (model)

1 | 2/3 | 1/2 | 1/3 | 1/4

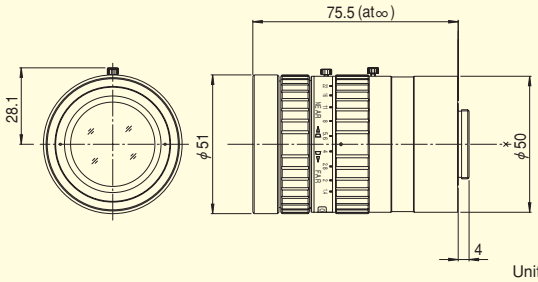
For FA/Machine Vision
Fixed Focal

CF25HA-1

FIXED Fixed Focal | 1.5 Mega For Megapixel Camera | MANUAL Manual Iris | C-mt C Mount | METAL Metal Mount | F1.4 Wide Aperture Rate

NEW

- High-resolution design, providing support for up to 1.5 megapixel camera resolution.
- Wide-aperture (F1.4) design achieves clear images under low light intensity.
- Low-distortion design achieving accurate image input.
- Robust enclosure resistant to vibrations and shocks. Equipped with locking knobs for the iris and the focus.



Unit : mm

Focal Length (mm)		25
Iris Range		F1.4 ~ F22
Operation	Focus	Manual
	Iris	Manual
Angle Of View (H×V)	1"	28°43′ × 21°44′
	2/3"	19°58′ × 15°02′
	1/2"	14°35′ × 10°58′
Focusing Range (From Front Of The Lens) (m)		∞ ~ 0.1
Object Dimensions at M.O.D. (H×V) (mm)	1"	65 × 48
	2/3"	44 × 33
	1/2"	32 × 24
Back Focal Distance (in air) (mm)		22.32
Exit Pupil Position (From Image Plane) (mm)		−140
Filter Thread (mm)		M49 × 0.75
Mount		C
Mass (g)		310

Remarks
- With Metal Mount
- With Locking Knob for Iris and Focus

Applicable camera (model)

1 | 2/3 | 1/2 | 1/3 | 1/4

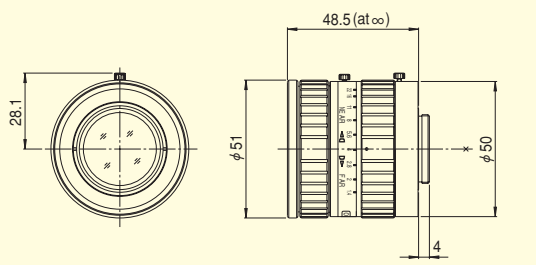
For FA/Machine Vision
Fixed Focal

CF35HA-1

FIXED Fixed Focal | 1.5 Mega For Megapixel Camera | MANUAL Manual Iris | C-mt C Mount | METAL Metal Mount | F1.4 Wide Aperture Rate

NEW

- High-resolution design, providing support for up to 1.5 megapixel camera resolution.
- Wide-aperture (F1.4) design achieves clear images under low light intensity.
- Low-distortion design achieving accurate image input.
- Robust enclosure resistant to vibrations and shocks. Equipped with locking knobs for the iris and the focus.



Unit : mm

Focal Length (mm)		35
Iris Range		F1.4 ~ F22
Operation	Focus	Manual
	Iris	Manual
Angle Of View (H×V)	1"	20°43′ × 15°37′
	2/3"	14°20′ × 10°46′
	1/2"	10°27′ × 7°51′
Focusing Range (From Front Of The Lens) (m)		∞ ~ 0.2
Object Dimensions at M.O.D. (H×V) (mm)	1"	73 × 55
	2/3"	50 × 38
	1/2"	37 × 27
Back Focal Distance (in air) (mm)		14.99
Exit Pupil Position (From Image Plane) (mm)		−37
Filter Thread (mm)		M49 × 0.75
Mount		C
Mass (g)		180

Remarks
- With Metal Mount
- With Locking Knob for Iris and Focus