PROJECT REPORT

HØGSKOLEN I ÅLESUND
Aalesund University College

CANDIDATE NO(S):			
1825,1847			
DATE:	SUBJECT CODE:	SUBJECT NAME:	
29.05.2015	IE303612	BACHELOR THESIS	
STUDY PROGRAMME:		NO OF PAGES/APPENDIX:	BIBL. NO:
Automation Engineering		66/5	

SUPERVISOR(S):

Ottar L. Osen

TITLE:

Gangway docking assistant

ABSTRACT:

Gangways are a common facility to transport personnel in offshore operations. Docking the gangways is challenging, and system developers of Uptime International gangways, ICD Software, desires a system to assist the docking operation.

This project attempts to develop an idea or system to assist gangway docking by enhanced visualization for gangway operators. A system was developed to utilize two cameras for mapping and visualizing the gangway landing in real-time, where data from the mapping is used to visualize a 3-dimensional model in a graphics engine. In addition, a feature of displaying the camera images directly, enhancing depth perception for the gangway operator by applying 3D-tv technology.

The result is a system implementable into the pre-existing gangway control system, with two modes. Mapping the gangway landing in real-time is possible, but certain measures have to be taken to ensure a satisfying result considering performance, processing rate and detail. Displaying the images with on a 3D-tv with 3D-features enabled, enhances depth perception from the camera images dramatically.

This report describes the background theory of the applied technologies, methods, results and conclusions of the project. In addition, proposals for further development is included.

This project is handed in for evaluation and accreditation at Aalesund University College.

Postadresse/Postaladdress
Aalesund University College
Postboks 1517
N-6025 Aalesund
Norway

Besøksadresse/Main office
Larsgårdsvegen 2
Internett/Internet
www.hials.no

Telefon/Telephone
+47 70 16 12 00
Epostadresse/Emailaddress
postmottak@hials.no

Telefaks/Telefax
+47 70 16 13 00

Publication Agreement

We hereby give Aalesund University College gratuitous access to make this assignment available for publishing.	yes ✓	no
Is this assignment confidential?	yes	no ✓
Can the assignment be published after the restricted period?	yes ✓	no
Is this assignment to be excluded from the open public? (contains confidential information. Ref. Offl. §13/Fvl. §13)	yes	no ✓

Gangway Docking Assistant

Birger Skogeng Pedersen, Thorbjørn Solberg

May 2015

Abstract

Gangways are a common facility to transport personnel in offshore operations. Docking the gangways is challenging, and system developers of Uptime International gangways, ICD Software, desires a system to assist the docking operation.

This project attempts to develop an idea or system to assist gangway docking by enhanced visualization for gangway operators. A system was developed to utilize two cameras for mapping and visualizing the gangway landing in real-time, where data from the mapping is used to visualize a 3-dimensional model in a graphics engine. In addition, a feature of displaying the camera images directly, enhancing depth perception for the gangway operator by applying 3D-tv technology.

The result is a system implementable into the pre-existing gangway control system, with two modes. Mapping the gangway landing in real-time is possible, but certain measures have to be taken to ensure a satisfying result considering performance, processing rate and detail. Displaying the images with on a 3D-tv with 3D-features enabled, enhances depth perception from the camera images dramatically.

This report describes the background theory of the applied technologies, methods, results and conclusions of the project. In addition, proposals for further development is included.

Contents

Chapter 1

Introduction

1.1 Uptime Gangways

In modern offshore operations, personnel are typically transported to platforms by helicopter or by ships to other installations. To enter an offshore installation from the ship, a passage that joins the deck of the ship and the installation can be installed. This passage is designed in such a way that personnel can walk from a ship, over the gangway and to a offshore installation. This facility is known as a **gangway**. The type of gangways relevant to this project are designed to be permanently mounted on the ship in one end, with the ability to reposition the other end on whatever installation where personnel transport is desired. In this report, the desired position of where to place the second end is defined as landing.

In Ålesund, a company named Uptime International develops, markets and installs such gangways. These gangways vary in length from 4.0 up to 23.4 metres. The gangways are hydraulically controlled by a gangway operator using a controller from the ship, with a technology implemented to automatically reposition the gangway to compensate for ship movement known as "heave compensation". The gangway operation controller is positioned at the base of the gangway, which requires the operator to track the tip of the gangway to control it safely to the offshore installation. This process is known as **docking**.

1.2 Control System Challenges

For the longer gangway models, this can be challenging as the tip of the gangway is relatively far away from its base. To help the gangway operator see the tip of the gangway better, a camera is mounted on the gangway and video is sent directly to the operation controller. While this solution aids the operator by obtaining a closer view of the tip of the gangway and the landing, perceiving the depth of the video image scenery can be difficult.

1.3 Assignment

As gangway developers, Uptime International desires a system to aid the gangway operator further. From this, the assignment "Tool for gangway docking" was developed. The assignment description reads as follows (translated from norwegian): "The groups' task is to develop a system that assists the operator in any way. This could be any system with 3D-visualization, object-tracking or automatic docking.".

1.4 Initial Ideas

To solve this assignment, ideas to track the position of the desired gangway landing was developed. One of these ideas was utilizing image processing to track a specific object positioned on the landing from the images captured from the camera mounted on the gangway. The data from this tracking could be used in a visualization application or used for a system enabling automatic docking. Another idea was to obtain 3-dimensional data of objects positioned on the landing, which could also be used for tracking or visually illustrating the structure of the gangway landing. To obtain this 3-dimensional data, another camera could be mounted on the gangway and a technology known as stereo matching could be utilized. With two cameras mounted, 3-dimensional video could also be recorded. This video can be viewed by the gangway operator with depth perception enhancement technology, for instance using a 3D monitor.

1.5 Project Priorities

The priorities of this project became to enhance the visibility of the gangway operator by depth perception or by using the 3-dimensional data to visualize the gangway landing. Such a system were to be implementable into the pre-existing control system delivered by Uptime. This control system is developed by ICD Software in a application framework known as **Qt**. By this, it was decided that an application were to be developed in Qt.

1.6 Thesis

- Will mounting another camera on the gangway enable the ability to enhance depth perception for the operator?
- Can the cameras provide 3-dimensional data?
- Can this 3-dimensional data be used to visualize the gangway landing?
- Will any of these features assist gangway operation?

Chapter 2

Theory

2.1 Terminology

2D	a space in which an object or mathematical object has two dimensions, meaning it has a span in width and height.
3D	a space in which an object or mathematical object has three dimensions, meaning it has a span in width, depth and height.
CDP	"Control Design Platform".
Encapsulation	packing of data and functions into a single component.
Filter	a device or system for removing certain elements from a quantity.
GIMP	"GNU (operating system) Image Manipulation Program".
GLSL	"OpenGL Shader Language".
GPU	graphical processing unit.
GUI	graphical user interface.
HD	high definition.
I/O	input and output.
ICD	"Industrial Control Design".
IDE	integrated developer environment.
Matrix	rectangular array of quantities in rows and columns, treated as a single entity.
Noise	irregular or unwanted data, disrupting a signal or image.
OpenCV	"Open-source Computer Vision".
OpenGL	"Open-source Graphics Language".
Programming	developing a computer program which instructs a controller to perform desired tasks.
Qt	application framework.
SAD	sum of absolute differences
USB	"Universal Serial Bus".
TV	television.
Widget	a component of a graphics user interface.

2.2 Depth Perception

Depth perception is the ability to compare the images hitting the retina, and achieve binocular vision. With this technique, the brain is capable of determining objects shape and position in relevance to each other. This concept is illustrated in figure 2.1.

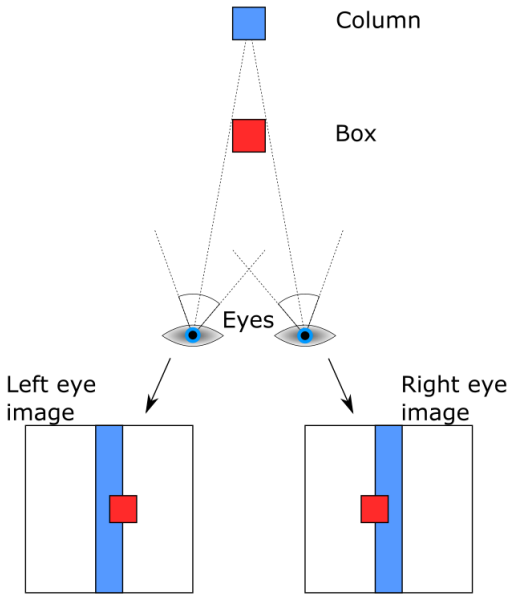


Figure 2.1: Illustration of two eyes observing an object. The objects two-dimensional position in each image differs. By comparing this difference, the distance from the eyes to the object can be estimated.

2.3 Computer Vision

Image processing is defined as any form of signal processing where the input is an image (such as a photograph or video frame) and the output either a processed image or a set of characteristics or parameters related to the image. Computer Vision is closely related to image processing, and is often considered high-level image processing out of which a machine/computer/software intends to decipher the physical contents of an image or a sequence of images.

2.3.1 OpenCV

Open Source Computer Vision (OpenCV) is a library of programming functions mainly aimed at real-time computer vision and image processing. It is developed by "Intel Russia" research center. OpenCV includes several hundreds of computer vision algorithms, including [8]:

- **core** - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- **imgproc** - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- **calib3d** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- **highgui** - an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.

2.3.2 MATLAB

MATLAB is an interactive computer environment for matrix manipulations, implementation of algorithms and image processing. It is developed and marketed by MathWorks. MATLAB can be extended with several add-on toolboxes for image processing and camera calibration. [4]

2.3.3 Digital Images

A digital image is a two-dimensional matrix consisting of "fragments" (also known as **pixels**). Each fragment has a particular value representing a single color or light intensity. Computer software can read these pixels as a signal, and by processing this signal, image processing is enabled.

2.3.4 Digital Cameras

A digital camera is a device with an array of light sensors which senses an image and stores it electronically.

2.3.5 Webcam

A webcam is a digital video camera developed to stream video conversations over the internet. The camera can be used for recording video and for capturing images.

Logitech HD Pro C920 Web Camera

The Logitech C920 is a web camera with built in auto focus capable of recording video in full HD resolution (1920 by 1080 pixels). [3]

2.3.6 Stereo Camera System

A system consisting of two or more cameras aligned horizontally pointing in the same direction. This system makes it possible to extract three-dimensional data from the images, using the same principal as human binocular vision.

2.3.7 Radial distortion

Radial distortion is the distortion in the image caused by the optical lens in the camera. Positive radial distortion makes the image bulge out from the center, often referred to as barrel distortion. Negative radial distortion makes the lines that don't go through the center bow inwards. This is often referred to as pincushion distortion. Figure 2.2 shows positive and negative radial distortion.

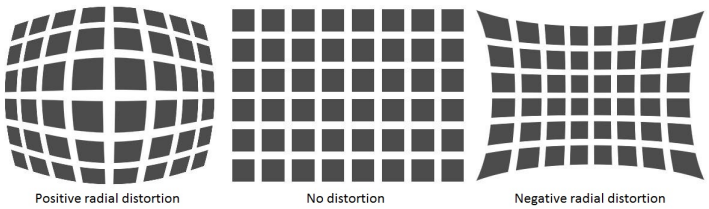


Figure 2.2: Radial distortion caused by camera lens

2.4 Camera Calibration

Camera calibration is the process of computing the intrinsic, extrinsic and distortion parameters of the camera. This is done mathematically by advanced applications detecting the corners of a specified calibration pattern. The intrinsic parameters are the internal parameters of the camera, such as the physical size of the imaging sensor, the field of view in the horizontal and vertical sensor axis, and the focal length of the camera. The extrinsic parameters are the external parameters of the system such as the position of the camera relative to the calibration patterns. The distortion parameters describe the distortion effects caused by the optical lens in the camera.

2.4.1 Calibration pattern

A calibration pattern is often an asymmetric chessboard off odd and even dimensions (e.g. 9x6). When the dimensions, size and orientation of the pattern is known, the positions of the inner corners can be detected and used to calculate calibration parameters.

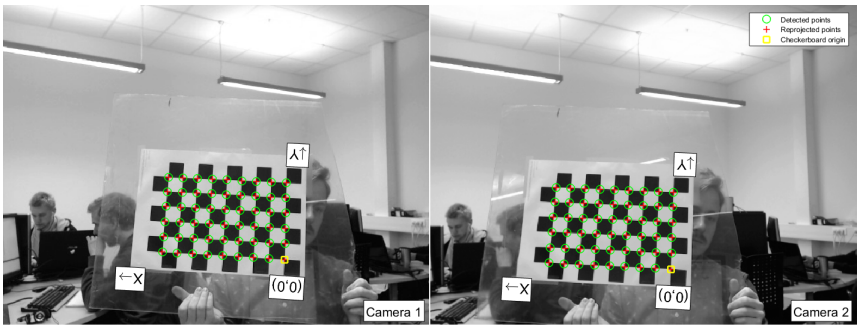


Figure 2.3: Image pair captured simultaneously by the left and the right camera containing a chessboard used for calibration and the detected inner corners.

2.4.2 Stereo Calibration

Stereo calibration is the process of computing the geometrical relationship between the two cameras in space" [15]. The computed parameters are additional extrinsic parameters describing the translation and rotation of the right camera relative to the left. The translation between the cameras is referred to as the *baseline* of the stereo camera system.

Undistortion

Undistortion is a way to mathematically remove the distortion caused by the optical camera lens, by use of the distortion parameters computed in the camera calibration.

Stereo Rectification

"*Stereo rectification* is the process of "correcting" the individual images so that they appear as if they had been taken by two cameras with row-aligned image planes." [15]. The images are rectified by use of transformation maps generated from the calibration parameters. This process makes it possible to compute stereo correspondence using stereo matching algorithms. When rectified, the stereo correspondence can be computed using stereo matching algorithms. Figure 2.4 describes the process of undistorting and rectifying a pair of stereo images.

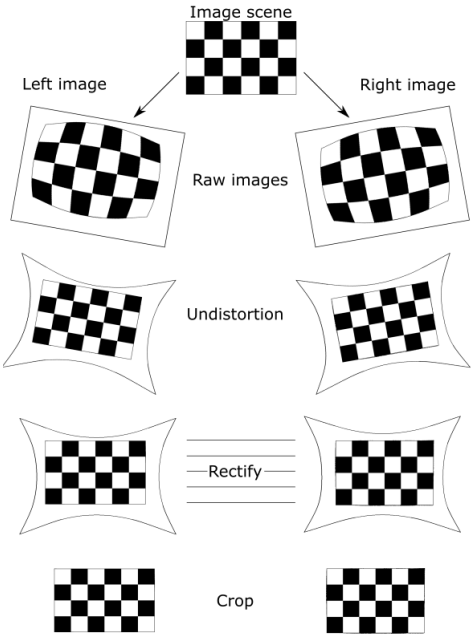


Figure 2.4: Stereo rectification and undistortion performed on a pair of stereo images. After rectification it is possible to search horizontally trough the images to detect correspondence between them

2.5 Disparity Maps

2.5.1 Disparity

Disparity is referred to as the known horizontal displacement in pixel-coordinates between the location of a certain object in a pair of stereo images. Disparity is computed by extracting the positions of the object in the left and right image. $d = x_l - x_r$ where d is the disparity, and x_l and x_r is the horizontal position of the object in the left and right image. The position of the objects close to the camera will differ more than the objects far away. Disparity is inversely related to depth, and can be used to compute real distance to the objects. Figure 2.5 show a pair of images taken by a stereo camera system, and the disparity between the objects.

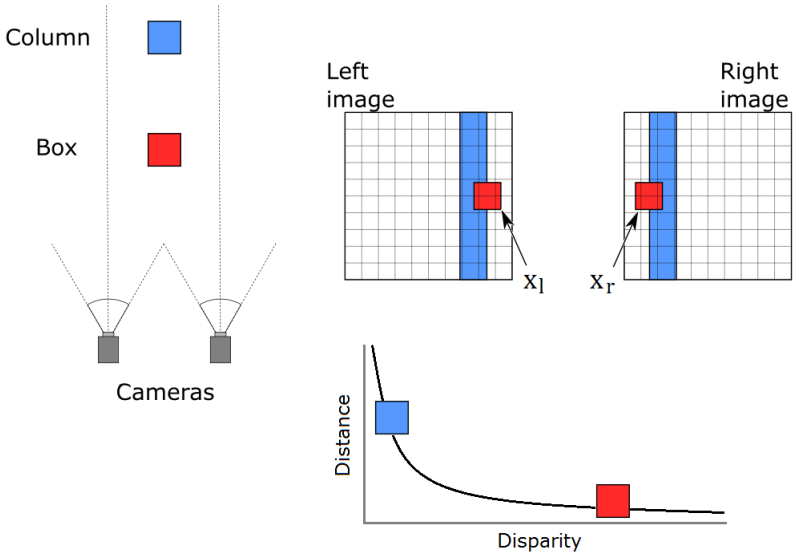


Figure 2.5: The objects two-dimensional position in each image differs. By computing the disparity, the distance from the cameras to the object can be calculated. Disparity is inversely related to distance

2.5.2 Disparity Map

A disparity map is a two-dimensional matrix with each quantity containing a single value representing a pixel disparity value. By visualizing each quantity as a colored pixel, the disparity map can be displayed as a gray-scale image. High values (high disparity) results in brighter pixels, while low values (low disparity) results in darker pixels as shown in figure 2.6.

Figure 2.6: Disparity map shown as gray-scale image. Bright features are close to the cameras, and dark features are further away. The black areas represent features as far away from the camera that correspondence is not found.

2.5.3 Stereo Correspondence

Stereo correspondence is referred to as correspondence between a pair of rectified stereo images. The disparity between features found in the left and right image can be computed and the product is a disparity map.

2.5.4 Stereo Block Matching

Matlab and OpenCv implements a fast and effective stereo block-matching algorithm to compute stereo correspondence. The implemented function is known as the **semi-global block matching**-method [7]. The images are first pre-filtered to enhance the texture so that features are easily detected. The function iterates through the images using a **SAD window** (sum of absolute differences) to find matching features between the left and right rectified images. For each feature found in the left image, the function search through the corresponding row in the right image to find a match. The algorithm use the **SAD window** to match blocks of pixels instead of single pixels. This is done to reduce the processing time of each set of images which is preferred in a real time application. Postfiltering is added to eliminate bad correspondence matches. In OpenCV this functionality is implemented in the class **StereoSGBM**. In MATLAB the same method is implemented in the function named **disparity()**. The result is a disparity map.

2.6 Computer Programming

Computers process data by instructions called computer programs, and the programs that run on a computer are referred to as software. The data used in a computer program is stored in variables, and the name of each variable is defined as an "identifier". In addition to primitive data types like numbers, variables can contain objects [16].

2.6.1 Building software

Developing a computer application can be done in a number various of various programming languages. A computer can only understand its own machine language, defined by its hardware design. These languages generally consist of strings of numbers (ultimately reduced to binary numbers), instructions for the computer to perform one at a time. The computer programs are cumbersome for humans. Programming in machine language is too slow and tedious for most programmers. Therefore, high-level-languages are developed in which single statements can be written to accomplish substantial tasks. To run software developed in high-level-languages, the source code has to be translated to machine code. This process is referred to as "compiling", and a application performing this task is known as a "compiler" [16].

2.6.2 Object-oriented programs

Objects

Object-oriented programming is a model of computer programming based on the concept of objects. An object is a data structure that contain data in the form of member fields and procedures known as methods. The objects member fields can consist different data types or even objects, which can be accessed and often modified by the objects methods. In object-oriented programming, computer programs are designed by making them out of objects that interact with each other. To interact with an object, it has to be created first. When an object is created, the program allocates some memory for this object to store its data. When an object is no longer needed, it can be destroyed, thus releasing the memory allocated [16].

Classes

Objects are defined as instances of a class. A class is defined as an extensible program-code-template providing initial values of an objects field members. Classes can inherit from each other, and the inheriting class is known as a "subclass" while the base class is known as a "superclass". With this, a subclass obtains the same functionality as the superclass, and additional methods and fields can be defined for the subclass. The classes relation with each other in regards to inheritance can be modeled as a class hierarchy. Some classes can

not be instantiated, but only inherited from. Such classes are defined as "abstract" classes. The point of such a class hierarchy when developing computer programs, is to avoid duplicating source code for classes that have fields and methods in common [14].

When an object is instantiated, a constructor for the particular class is invoked. The constructor is, like a method, a procedure for an object. However the constructor of an object is only invoked once, when the object is created [14].

Encapsulation

A class can disallow calling code from accessing internal data and force access through methods only. Hiding information like this is known as encapsulation. Certain keywords in the code is used when defining member fields, methods or constructors for specifying the encapsulation [14].

2.6.3 C++ language

The C++ programming language is evolved from C. C, thus C++, is a hardware independent programming language developed by Bell Laboratories. C++ provides a number of features that "spruce up" the C language, including capabilities for object-oriented programming [16].

Variable scope and duration

The scope of a variable determines where it is accessible. A variables duration determines where it is created and destroyed. A variable can be local, meaning it is only accessible inside the code block it is defined in. These variables have "automatic" duration, and are destroyed when the block they are part of is exited. A variable can, however, be defined as global. This variable is accessible from any scope of the computer program [16].

Copied and referenced objects - Pointers

When defining a function in C++, certain parameters can be specified to take as input in the function. These parameters can be primitive data types or objects. When defining what input parameters a function has, objects can either be specified as a copy of or reference to a pre-existing object. When the reference of an object is passed, altering the field members of that object affects the referenced object. Passing a copy of an object prevents the function from altering the field members of the original object. The reference of an object is also known as the objects memory address. To store this address, a "pointer" variable can be instantiated as illustrated in the following source code example:

```
// C++ pointer variable example
/* Instantiating a integer variable
"number" and assigning a value of 30. */
int number = 30;
/* Instantiating a pointer variable "number_pointer" and
```

```
assigning the memory address of the "number" variable. */
int *number_pointer = &number;
```

These features are useful when optimizing performance. Since copying objects allocate memory, copying objects that are not meant to be modified but only meant for obtaining data can be avoided.

Accessibility of object members

C++ uses three keywords for specifying the accessibility of members: "private", "protected" and "public". A private member variable or function is invisible and cannot be accessed from outside the class. A protected member variable or function is like a private member with the additional benefit of being accessed in subclasses. A public member is accessible from anywhere outside the class within the program.

Struct

By default, the accessibility of all members of a class is private unless defined otherwise. A data structure, also known as a "struct" is based on the same concept as a class. All members of a struct, however, are public unless defined otherwise. While instantiating a class entity produces an "object", instantiating a struct entity produces whats known as a **product** [16].

2.6.4 Qt software

"Qt" (pronounced "cute") is a application and user interface framework developed by the Qt Company. Qt is a toolkit consisting of a set of libraries written natively in C++, which include (among others) the following features [12]:

- Core, GUI, network and XML library.
- A program to generate some boilerplate C++ code, extending C++ by adding features. Most importantly the signal/slot mechanism.
- A GUI designer tool.
- The qmake tool, used to automate an applications build process (i.e. running the necessary tool chain to build an application).
- The Qt Creator, a graphical IDE to integrate all the Qt features into a single environment.

Signals and slots

To enable interaction between objects, the typical approach for frameworks other than Qt is using callbacks. To receive a notification in a processing function (receiver), the receiver has to pass a function pointer to another function (notifier). The notifier calls this function pointer when an event occurs. The

receiving function then "calls back" to the notifying function when appropriate. This mechanism is not type safe, meaning there is no guarantee that the processing function will call the callback with the correct arguments. The callback is also coupled to the processing function since the processing function must know which callback to call [12].

With Qt, an alternative to the callback is provided; using signals and slots. A signal can be emitted when an event occurs. A slot is a function that is called in response to a particular signal. The signature of a signal must match the signature of the receiving slot, meaning the parameters in the signal and the slot has to be of the same type. This ensures type safety. The slot may have a shorter signature than the signal, in which case the remaining parameters of a signal is ignored. A class which emits a signal has no need to know which slots receive a signal [12].

Signals and slots are then connected, and the slot will be called with the signals parameters at the right time (i.e. when the signal is emitted). Any class meant to utilize this feature must be a subclass of a QObject or any of its subclasses provided in the Qt core library. Several signals can be connected to a single slot, and a single signal can be connected to several slots. Different signals can also be connected, which emits the second signal immidiately whenever the first signal is emitted [12].

2.7 Real-time systems

A typical, modern processor run at speeds much greater than the input and output devices with which they must interact. As a result, a program that wishes to perform and intensive I/O operation and do calculations sequentially will run at a relatively low speed. A solution for such programs is concurrent programming [27].

Concurrency in programming is an expression used to describe potential parallelism in application programs. It includes the techniques for dealing with communication and synchronization between potential parallel entities. In Concurrent computing, several computations are executing during overlapping time periods. This has three main advantages over sequential programming [27]:

- Full utilization of the processor.
- Allowing more than one processor core to solve a problem.
- Being able to model parallelism in the real world.

A "Real-time" system can be described as systems that have to respond to externally generated input stimuli within a specified time interval. This is a major application for concurrent programming. Such systems are inherently concurrent because they are often embedded in a larger engineering system and have to model the parallelism that exists in the real-world objects to which they are monitoring and controlling [27].

2.7.1 Computer Threads and Multithreading

In computer science, a thread of execution is the sequence of programmed instructions that can be managed independently by a scheduler, typically as part of an operating system. The implementation of threads and processes differs between operating systems. However, in most cases, a thread is a component of a process [25].

Multiple threads can exist in the same process and share resources, while different processes do not share these resources. Multithreading on a single processor is generally implemented by time-division multiplexing (or multitasking). The processor switches between different software threads. This switching generally happens frequently enough that the user perceives the threads or tasks as running at the same time [25].

If a system as several processors or cores, threads can be truly concurrent with every processor or core executing separate threads simultaneously. This is known as hardware threads, and is different from software threads which are a pure software construct. The processor or CPU has no notion of software threads, and is unaware of their existence [19].

2.7.2 Multithreading in Qt

Qt provides a base class called QThread. By inheriting from this thread, a class has the ability to run in its own thread with the abilities of using signals and slots for communication with other objects. By defining a **run()** method in the class, invoking the **start()** method of the QThread superclass starts a thread and processes the code in the **run()** method in that thread.

2.7.3 Sharing objects between threads

In comparison with a single threaded application, a multi-threaded system has a couple of mentionable pitfalls. When several threads are processing from the same data, one thread may alter data another thread is using. Consider the following example; a thread a is responsible for capturing images and storing them to an object, and another thread b is responsible for processing the captured images from the same object. A risk here is that while thread b in the middle of processing an image, thread a captures a new one and overwrites the same image thread b is processing. This may invalidate the result from thread b. Another problem is that one thread may not be able to access the image, and the thread waits or stops without ever being restarted. The latter situation is known as a "deadlock". To prevent situations like these, sharing objects between threads may require additional "safety" mechanisms.

The application developed for this particular project includes multi-threading. A thread that needs an object from a second thread, waits until the second thread is done running. Once the second thread is finished, the first thread obtains the object and processes the data. This way, threads are not altering the same objects simultaneously, thus providing thread safety.

2.8 OpenGL

Open Graphics Language (OpenGL) is a cross-language, multi-platform application programming interface for rendering vector graphics in 2D and 3D. It was initially developed by Silicon Graphics Inc. in 1991, now managed by the non-profit consortium Khronos Group. The OpenGL specification describes an abstract application programming interface (API) which can be implemented entirely in software or hardware. It defines a number of functions which may be called by a client program.

2.8.1 Context

An OpenGL context stores all of the state associated with its instance of OpenGL. it represents the default framebuffer that rendering commands will draw to when not drawing to a framebuffer object. Contexts are localized within a particular process of execution on an operating system. Such processes can create multiple OpenGL contexts, each representing seperate viewable surfaces (like a window in an application) [9].

The OpenGL Specification omits the subject of obtaining, and managing and OpenGL context. Thus, OpenGL is purely concerned by rendering. Creating an OpenGL context is a common feature of several user-interface libraries, one of such is Qt.

2.8.2 Shader programs

For rendering the graphics in a OpenGL application, a program designed to run in the graphics processor is also developed. Such a program is defined as a "shader" program. OpenGL provides its own programming language for developing such programs, known as "Graphics Language Shader Language" (GLSL). This is a high-level shading language based on the synbtax of the C programming language. By developing a shader program, developers have direct control of the graphics pipeline without having to use hardware-specific languages.

2.8.3 Vertex data

In geometry, a vertex is defined as an angular point of a figure. By submitting vertex data and describing how to interpret the data, OpenGL can draw figures appropriately. The data in a vertex can represent several things, but is stored as a vector of numbers. For instance, a vertex can represent a corner of a triangle with a specific color. By providing three vertices representing a coordinate and a color each, the program can be issued to draw the triangle with a different color in each corner.

2.8.4 Indexing

In some cases, the same vertices are used several times when drawing figures. For instance if a whole area of adjacent triangles are to be drawn, some triangles have the position of its corners in common. To avoid having to submit the same vertices several times for rendering a single frame, the vertices can be indexed. Once all the vertex data is submitted to the stream (i.e. allocated in the GPU memory), indices can be used to describe which vertices are to be used for drawing. This mechanism is known as indexing.

Some objects, like a detailed, 3-dimensional model, can contain a large number of vertices. Rather having to pass the vertices each time a frame is rendered, a buffer can store the vertex data in in the GPU memory once. The vertices are then used by indexing. This reduces the load time when rendering, thus indexing is a preferred solution when dealing with relatively large quantities of vertex data.

Rendering pipeline

In OpenGL, a sequence of steps is taken when rendering objects. This is known as the "rendering pipeline". Each step has its unique responsibility when rendering graphics in OpenGL. During these steps, at least two shader programs are run: The vertex and fragment shader.

Vertex shader

Vertex shaders are fed vertex attribute data, as specified from a vertex array object by a drawing command. The purpose of this shader is to calculate scene geometry. The shader recieves a single vertex from the vertex stream and generates a single vertex to the output stream. There must be a 1:1 mapping from input vertices to output vertices [10].

2.8.5 Fragment shader

A fragment shader (also known as pixel shader) computes the color of each pixel (referred to as a fragment) when rendering a frame.

2.8.6 Projection

To resemble a projection somewhat similar to the human eye when rendering, the vertices in the OpenGL application have to be transformed. To do this, a model, a view and a projection matrices have to be calculated. The model matrix maps from an objects local coordinate space into world space. The view matrix maps from world space to a viewpoint space. The projection matrix maps from the viewpoint space to screen. Composing all three into a single matrix enables the ability to map vertices all the way from object space to

screen space. The vertex shader uses this transformation matrix to calculate the position on the screen according to attributes of a viewpoint.

Chapter 3

Method

3.1 Planning and Administrative Preparations

In advance of this project, a pilot project was developed. The purpose of this pilot project was to investigate the task, define the problem and shape the research questions. Proposals and ideas of how to solve the assignment was also prepared. Throughout the project, progress meetings with the project supervisor and the contact person from ICD Software was held on average every 14 days. A progress report was prepared in advance of each progress meeting. After each meeting, tasks were created and distributed between the team members. Responsibilities and progress was recorded in a project-management application named "Asana". The pilot project report, progress reports and an overview of all tasks throughout this project represented as a Gantt chart are attached to this report. Note that these documents are written entirely in Norwegian, as this was initially the language this report was meant to be written in.

3.2 Qt Applications, IDE and OpenCV Library Setup

As the software created for this particular project are supposed to assist CDP, an already fully developed control system developed in Qt, the goal of this project became to create some software relatively easy to implement in CDP. As Qt GUI applications like CDP consist of widgets, the application of this project has been developed primarily as a Qt Widget. This ensures that the software is modular to any Qt GUI application, including CDP and the Uptime Gangway GUI software.

3.2.1 Getting started with Qt development

A Qt IDE license is free for community development and the IDE is obtained from the Qt Web Page[12], along with tutorials on how to create custom Qt

Applications. This application was not intented to be a complete Qt application, but rather a module (widget). However, for development and testing of this module, a simple Qt widgets application project was made in the Qt IDE.

The empty project was named **StereoVisualizer**. Initially, a empty Qt widgets application consist of a main source file ("main.cpp"), a project file ("StereoVisualizer.pro") and a default GUI form, header and source file. The default GUI is not necessary to keep for development, and the form, header and source files are removed. The "main.cpp" file is the starting point for executing source code, and invoking functions etc. is always done initially from the source code in this file. The "StereoVisualizer.pro" file defines which files should be included when building the project in Qt.

3.2.2 Importing the necessary libraries

For camera input and image processing capabilities, OpenCV is imported to the Qt project created by downloading and installing OpenCV to the controller.

OpenCV is not pre-compiled for MinGW, and to use it in the Qt application created here, it is compiled for MinGW. As MinGW is included with the Qt installation from a previous step, it can be used for compiling OpenCV aswell. The process of compiling OpenCV as done for this project is described in appendix A. Once compiled, the libraries are ready for use. To utilize the OpenCV libraries in the application, the binary files are added to the operating system as described in appendix B.

Once downloaded and installed, the OpenCV libraries are imported to the project by adding the following code snippet to the project file:

```
INCLUDEPATH += C:\opencv\build\include

LIBS += -L"C:\opencv\build\x64\mingw\lib" \
    -lopencv_core2411 \
    -lopencv_highgui2411 \
    -lopencv_calib3d2411 \
    -lopencv_imgproc2411
```

This enables the Qt Application to access the following OpenCV Modules, with its corresponding abbrevations.

- The Core Functionality, "core".
- High-level GUI and Media I/O, "highgui".
- Camera Calibration and 3D Reconstruction, "calib3d".
- Image Processing, "imgproc".

Finally, the Qt IDE and the OpenCV libraries are compiled, installed, set up and ready to be utilized in the application.

3.3 Stereo Visualizer Widget

The base application of this project was developed as a Qt Widget subclass named **Stereo Visualizer Widget (SVWidget)**. This widget is responsible for rendering the graphical user interface and controlling whatever processing necessary to in accordance with user input.

While the "SVWidget" class handles the graphical user interface, a class named **Stereo Visualizer Thread (SVThread)** was developed for capturing and processing image data from cameras connected to the controller. The **SVWidget** class has a field member of type **SVThread**, and the **SVWidget** connects the signals emitted from the **SVThread** to obtain data from the image processing executed in the **SVThread** when the thread instance is started. The **SVWidget** also contains other widgets including **QLabel** instances for displaying images and a **Stereo Visualizer Engine Widget (SVEngineWidget)** for rendering 3-dimensional graphics. Once the **SVWidget** instance receives a signal from the **SVThread**, it updates its widgets accordingly.

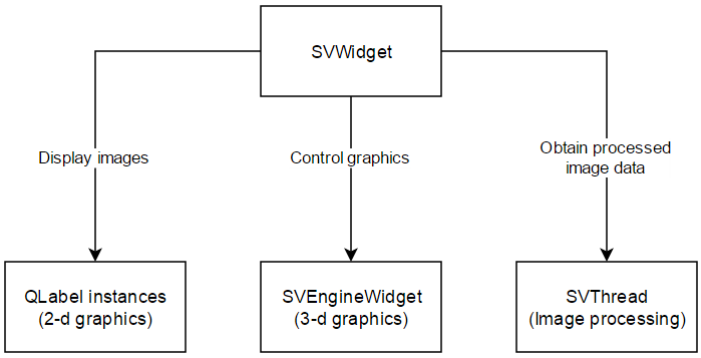


Figure 3.1: Stereo Visualizer Widget (SVWidget) class relationships

3.3.1 Desired Application Functionality

The desired functionality of the **Stereo Visualizer Widget** was to enhance depth perception from cameras or similar sensors for distance measurement. With guidance from the project supervisor the idea of feeding camera input into a 3D monitor in real-time was also developed. By this, the application has two different selectable modes for visualization. The modes are mutually exclusive, meaning that the application never has more than one mode activated at a time. To switch between these modes, the widget is developed as a subclass of the **QTabWidget** class. A **QTabWidget** instance holds selectable tabs to display or hide active or inactive GUIs within the widget. One tab is created for each mode. When a tab is selected, the **SVThread** instance is stopped and the mode corresponding with the selected mode is activated in the **SVThread**. Finally the **SVThread** instance is restarted.

Binocular Vision Utilization

One mode of the application was to utilize the concept of binocular vision by displaying camera input directly with a relatively high frame rate, or at least approximately the same frame rate as maximum for a typical camera. This mode was titled ”Stereo Video”.

Distance Measurement

Another desired functionality of the application was to measure the distance of objects in the camera images by image processing. The distance data from the image processing is meant to be used for re-projecting 3-dimensional figures resembling whatever the cameras are filming. This mode was titled **Re-project**.

3.4 Stereo Video Feed

3.4.1 Utilizing the Samsung Smart-TV 3D-functionality

A Samsung Smart 3D TV with 3D glasses can be configured to split an input frame into two separate frames from the center of the monitor input frame horizontally or vertically.

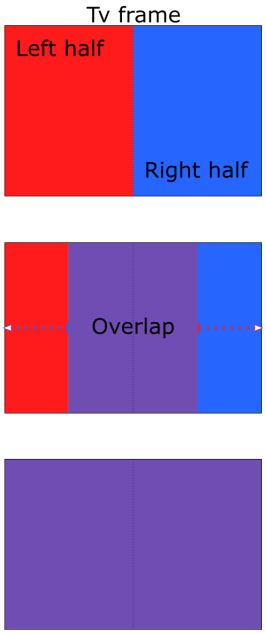


Figure 3.2: Input frame split in Samsung 3D TV

Both separate frames are then displayed in the center of the TV sequentially in synchronization with the 3D glasses. To a user with the 3D-glasses equipped

and enabled, the left half of the input frame is only visible to the left eye, while the right half is only visible to the right eye. This enables the user to perceive depth through binocular vision as described in section 2.2. To utilize this effect the **SVWidget** class implements the ability to display video feed from two cameras in real-time as described in section 3.4.2.

3.4.2 Capturing Stereo Images with OpenCV

As the position of objects are compared in both images, it is imperative that both images are captured at approximately same instance. If an object in movement has moved between left and right image being captured, the objects position cannot be used to determine the distance to the object, as this will be affected by how far the object moved. Therefore, it was decided that the application should attempt to capture the left and right image approximately simultaneously. A **Video Capture Thread (VideoCaptureThread)** class was developed for exactly this purpose. Instantiating an object of this class enables the ability to open a camera connection with a **cv::VideoCapture** instance and capture images in a multi-threaded application. Each **VideoCaptureThread** instance has its own corresponding **cv::VideoCapture** instance, which can be accessed directly. Two instances of the **VideoCaptureThread** class objects are created for each **SVThread** instance, one for each camera. The **SVThread** class accesses the **cv::VideoCapture** instance of each **VideoCaptureThread** instance, to set certain parameters before initiating image capturing. These parameters are image resolution and auto-focus. The image resolutions are varied throughout development, but auto-focus is disabled.

For the **SVThread** instance to obtain the two images, the **start()** method of each of the two **VideoCaptureThread** instances (named **videoCaptureLeft** and **videoCaptureRight** in the source code) are called. This issues the instances to starts its own **QThread** superclass thread, which in turn issues capturing an image. Once both thread instances are started, the **SVThread** instance waits for both to finish. Finally both images can be obtained by the **VideoCaptureThread getFrame()** method. As mentioned, this entire process is looped in a individual thread.

For large image resolutions, this process requires a relative high controller CPU performance. Therefore, a **QTime** timer instance is created in the **SVThread** instance for keeping track of the time between each image capturing process is issued. This enables the ability to tweak and reduce the frequency of how often this process is done, resulting in less CPU usage by the widget and the application. The miniumum time between the image capturing process to complete is defined in the **MINIMUM_TIME_BETWEEN_FRAMES** constant integer field in the **SVThread** instance.

```
// SVThread run method source code snippet

// Check time since images were most recently captured
if (timeLastImageCapture.elapsed() < MINIMUM_TIME_BETWEEN_FRAMES)
{
    // If unideal time has passed, wait until it has
    int timeDifference =
    MINIMUM_TIME_BETWEEN_FRAMES - timeLastImageCapture.elapsed();
```

```
    if (timeDifference > 0)
    {
        QThread::msleep(timeDifference);
    }
}

// Start VideoCaptureThread instances
videoCaptureLeft.start();
videoCaptureRight.start();

// Wait for both threads to finish
if (videoCaptureLeft.isRunning())
    videoCaptureLeft.wait();
if (videoCaptureRight.isRunning())
    videoCaptureRight.wait();

// Obtain resulting images
cv::Mat leftFrame = *videoCaptureLeft.getFrame();
cv::Mat rightFrame = *videoCaptureRight.getFrame();

// Reset the time since last images were captured
timeLastImageCapture.restart();
```

Due to multithreading capabilites, this issues both threads to capture an image each virtually independent of each other. The purpose of multi-threading this process is to attempt both images to be captured at approximately the exact same time, rather than capturing an image from one camera, wait until it is captured then capture an image from the next camera. A captured image is constructed as an OpenCV matrix.

3.4.3 Displaying Stereo Video in Real-Time

To display the images from the stereo video feed in the graphical user interface of the application, the images are obtained by the class handling the graphical user interface (i.e. the **SVWidget**). When the **SVThread** obtains the left and right images by its **VideoCaptureThread** instances, the images are intially obtained by the **VideoCaptureThread** instances as **cv::Mat** matrices. To notify an object trying to obtain the images, the **SVThread** emits a **leftImageReady** and **rightImageReay** signal. This signal sends a **QImage** instance, and not a **cv::Mat** instance. Therefore, a **QImage** instance is created for each **cv::Mat** instance, and the images are converted into these **QImage** instances. A method for this conversion is implemented in the **SVThread** class, and the source code is as follows:

```
QImage SVThread::cvMatToQImage(cv::Mat const& imageMatrixBGR)
{
    // Temporary matrix for storing a BGR to RGB-converted image
    cv::Mat convertedMatrix;
    // Convert the original BGR image to a the RGB image
    cvtColor(imageMatrixBGR, convertedMatrix, CV_BGR2RGB);
    /* Construct a QImage from the BGR converted image from the
     * existing memory buffer data provided by the
     * converted matrix */
    QImage result(
        (const uchar *) convertedMatrix.data,
        convertedMatrix.cols,
```

```
            convertedMatrix.rows,
            convertedMatrix.step,
            QImage::Format_RGB888);

    /* Make this image a deep-copy of the image,
     * rather than the buffer passed earlier */
    result.bits();
    return result;
}
```

Once a **SVThread** instance has captured and converted an image, the signal corresponding to that image is emitted. The **SVWidget** listens for these signals, and the images are obtained by the **SVWidget** once the signal is recieved. The **SVWidget** has two QLabel instances, one for displaying images from each of the cameras. Once an image captured and converted from one of the cameras is recieved from the **SVThread** instance, the recieved image is displayed in its correspoding **QLabel** instance as shown in figure 3.3.

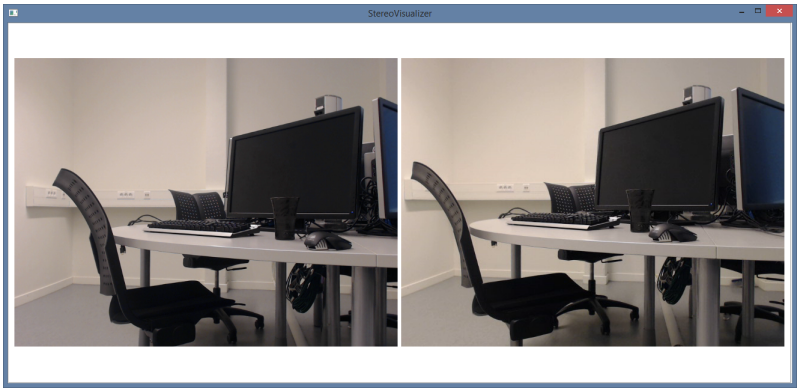


Figure 3.3: Displaying "simultaneously" captured images from both cameras

The video output of the application using this **SVWidget** is connected to the Samsung 3D TV, the 3D-functionality with vertical frame split as illustrated in figure 3.2 is enabled in the TV settings. Lastly, the active 3D glasses are equipped and enabled.

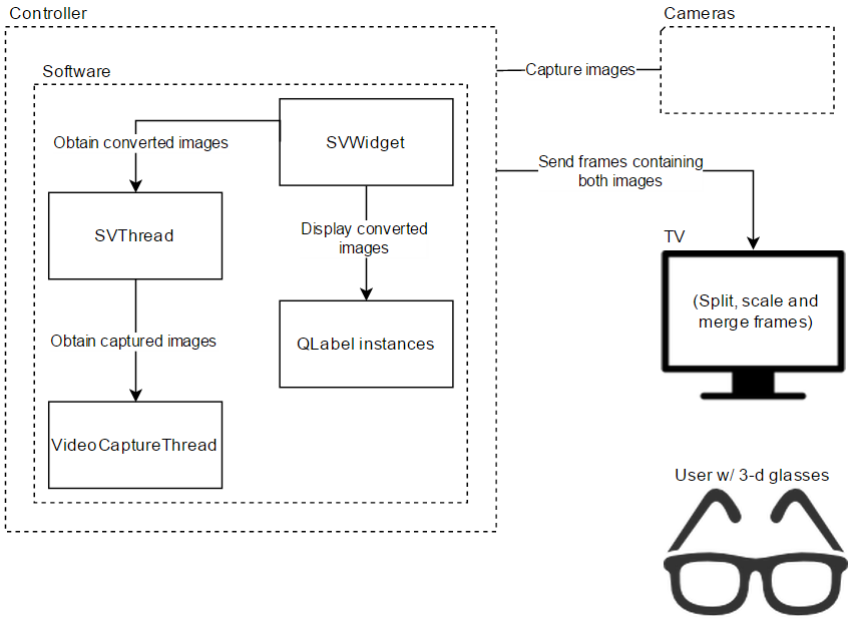


Figure 3.4: Stereo Video display concept sketch

3.5 Distance Measurement from Stereo Images

Another way to utilize the images is to use software to measure the distance from the cameras to objects in the image. In order to measure distance using a pair of cameras, the cameras have to be calibrated. The calibration is done in two steps. The cameras are first calibrated individually, to compute the intrinsic- and distortion parameters, then a stereo calibration is performed to compute the geometrical relationship between them. Camera calibration can be done in Matlab or with OpenCV.

3.5.1 Calibration Preparations

Before the calibration, some preparations is necessary. Both the Matlab and OpenCV calibration applications use an asymmetric chessboard of a known size, attached to a flat surface as a calibration pattern. The chessboard has to be asymmetric, and of odd and even dimensions (e.g 9x6) so that the orientation of the board is known, as shown in figure 3.5.

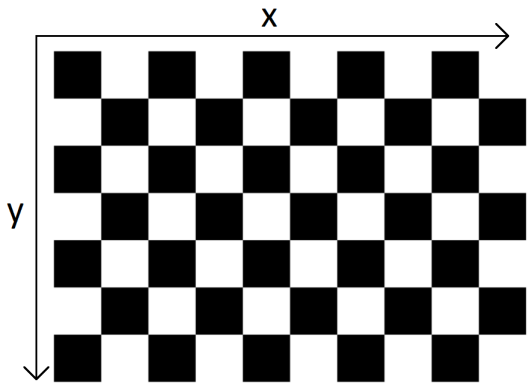


Figure 3.5: Asymmetric chessboard with 9x6 inner corners

The cameras have to be aligned horizontally with the optical axis pointing as close to straight forward as possible. This is difficult to do manually, and is one of the reasons why the stereo calibration is needed. If the selected cameras have built in auto focus, this has to be turned off. The focus should then be set manually to concentrate on the calibration pattern located at the desired distance from the cameras. It is important not to change the zoom settings between capturing images, because this will affect the focal length of the cameras. [5]

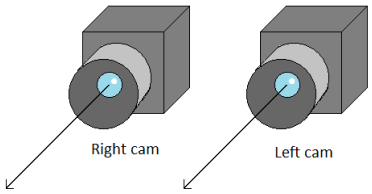


Figure 3.6: Cameras lined up horizontally pointing in the same direction

3.5.2 Calibration in Matlab

Matlab contains an built in application named "Stereo Camera Calibrator". This is an advanced, easy to use, tool to calibrate a pair of stereo cameras. The application requires between 10 and 20 gray-scale image pairs of the calibration pattern, the dimensions and the size of the pattern. Each image pair have to be captured at the same time, so that the calibration pattern stays at the same position in both images. If any of the image pairs are rejected, the application will remove these images automatically. The image pairs can then be evaluated manually, and images that affects the calibration in a negative way can be removed. The product is an object containing the intrinsic, distortion and extrinsic parameters of the stereo camera system. The object can be used to compute disparity maps and to reconstruct scenery in Matlab. Figure 3.7 displays the detected chessboards relative to the position of the two cameras in

space, known as the extrinsic parameters of the stereo camera system.

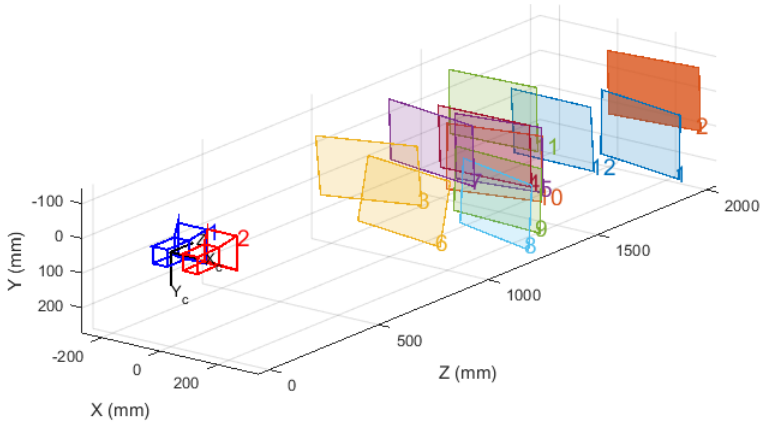


Figure 3.7: Detectet chessboards relative to the position of the cameras

3.5.3 Calibration in OpenCV

OpenCV contains a section for camera calibration and 3D reconstruction. To perform a stereo calibration with the OpenCV library, The class named **Calibration** was developed. Rather than taking all of the image pairs in advance, store them in a file, and read them in again, **Calibration** lets the user calibrate the cameras real time. The square size and the dimensions of the chessboard, how many times the chessboard is going to be detected, and how many frames to skip between each detection, is specified in the constructor. The number of frames to skip between each detection gives the user time to move the chessboard to the next position. The square size in millimeters is multiplied with the width and height to get the correct size of the chessboard. This size is used to compute the exact distance between the cameras.

An object of **Calibration** can be instantiated by the class **Depthmapper**. **Calibration** contains a method named **calibrate()**, which takes a reference to each camera as input parameters. It captures a frame from each camera and calls the OpenCV function, **cv::findChessboardCorners()**, to detect the corners of the chessboard held in front of the camera. The position of the points are stored in a point2D vector, which is a vector for storing 2D points specified by its x and y coordinates. If all of the corners are detected in the correct order, it calls **cv::drawChessboardCorners()** to draw the corners directly on the video feed. When the requested number of chessboards have been detected, the harvested corner points is used to calibrate each camera separately using **cv::calibrate()**. This function finds the camera intrinsic and extrinsic parameters from different views of a calibration pattern. The calibration parameters retrieved from each camera is then used to perform a stereo calibration calling **cv::stereoCalibrate()**. When stereo calibrated, **calibrate()** calls **cv::stereoRectify()** to compute rectification and undistortion parameters. When the calibration process is finished, the object instantiated by **De-**

phtmapper calls the function **save_calibration_result()**, which writes all the parameters to a xml file named `"camera_parameters.xml"`

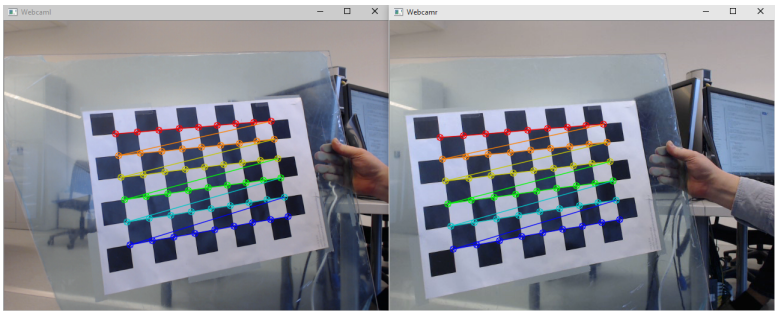


Figure 3.8: Two video feeds and the detected chessboard corners.

Note that once the stereo camera system have been calibrated, the cameras must not be moved. The geometrical relationship computed will then be incorrect, and the rectified images generated from the calibration parameters can no longer be used for computing stereo correspondence.

3.5.4 Creating and Processing a Disparity Map

In order to generate a disparity map, the image pairs have to be undistorted and stereo rectified. This is done by using the parameters from the stereo calibration. When the images are rectified, the stereo matching method mentioned in chapter 2.6, is used to find stereo correspondence between the images. The correspondence can be displayed in a gray-scale disparity map. OpenCV and Matlab contains functions to rectify images, and to compute stereo correspondence.

Matlab

Matlab contains functions to undistort and rectify images, and to create disparity maps. The rectification function is named **rectifyStereoImages()**, and takes the gray-scale images from the stereo cameras and the object, containing the camera parameters created in the calibration, as input. The function used to compute a disparity map is named **disparity()**. It takes two rectified images and optional parameters to change different settings of the block matching method as input. The disparity maps generated in Matlab have been used for comparison to create high quality disparity maps with OpenCV.

OpenCV

To compute disparity maps real time in the **SVWidget**, the class **DisparityMapper** was developed. It contains functionality of the OpenCV class

StereoSGBM, which computes stereo correspondence using the semi-global block matching algorithm. An object of **DisparityMapper** is instantiated by **Depthmapper**. **DisparityMapper** contains one initiation method, **initiate()**, and one calculating method, **calculate()**.

initiate(), reads in the required parameters from the xml-file created in the calibration process, and uses these parameters to compute an undistortion and rectification transformation map for each camera by calling the method **cv::initUndistortRectifyMap**, found in the calib3D section in OpenCV [7]. The transformation maps of type cv::Mat is stored as fields in **DisparityMapper**. Because this is a tedious operation, it is only executed once.

calculate(), removes the distortion and rectifies the images coming directly from each camera by using the OpenCV method cv::remap with the images and the transformation maps as input parameters. The rectified images is then used to find the stereo correspondence and to compute a disparity map by calling **cv::stereoSGBM.operator()**. The generated disparity map consists of pure disparity values. To be displayed like a gray-scale image, the disparity map has to be normalized. This is done using **cv::normalize**.

Both the disparity map and the gray-scale image, of type **cv::Mat**, is stored as fields in **DisparityMapper**. These fields can then be retrieved by calling the methods **getDisparityMap()**, and **getDisparityVal()**. The settings of the block matching method have been manually tuned, and set in the source code. The snippet below shows how **DepthMapper** ensures that the initiation is run before generating a disparity map.

```
void DepthMapper::initiate(cv::Mat &frame_l, cv::Mat &frame_r)
{
    //Read in required parameters from the calibration and
    //set the block matching settings
    disparity->initiate(frame_l, frame_r);

    initiated = true;
}

void DepthMapper::buildDisparityMap(
    cv::Mat &frame_l, cv::Mat &frame_r)
{
    //If initiation is not executed, run initiation
    if (!initiated)
    {
        initiate(frame_l, frame_r);
    }
    //calculate stereo correspondence and generate disparity maps.
    disparity->calculate(frame_l, frame_r);
}
```

3.5.5 Depth from disparity map

To obtain the distance to a given pixel based on the values from a disparity map, **getDepth()** was implemented in **DepthMapper**. The method computes the distance using an equation developed with the "Curve Fitting Tool" found in MATLAB. A data set of disparity values were compared to accurate laser measurement values.

The data sets were harvested by mounting the stereo cameras 25cm apart, together whit a handheld laser measure tool on a cart with wheels. Two objects were located two meters apart, so that the rearmost object was placed ten meters in front of the camera, as shown in figure 3.9. The Disparity values were recorded every five centimeters. This was done as shown in the source-code snippet below. Each time enter is pressed, the distance and the disparity value of the closest and the rearmost object is stored in a QVector. The distance to the objects is then reduced by a given value, and the cart had to be manually moved forward. This was repeated until the carriage was completely up close to the closest object. A plot of the equation and the recorded data set is displayed in figure 3.10

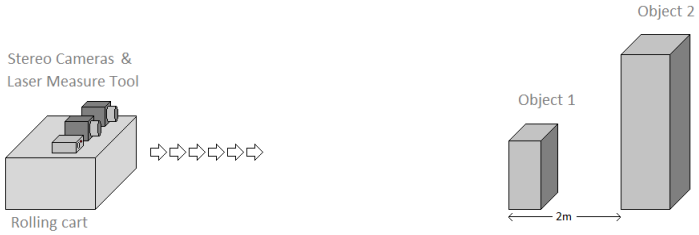


Figure 3.9: Stereo camera and laser measurement tool recording the distance to the two objects

```
//wait for 10 milliseconds
char key = cv::waitKey(10);
//If enter is pressed, record the the disparity values and the
//position. Concatinate the values in a Qvector, and reduce the
//position with a STEP.
if (key == 13) {
    QString string_high;
    string_high += QString::number(position);
    string_high += ", ";
    string_high += QString::number(depth_high);
    measurement_high << string_high;

    QString string_low;
    string_low += QString::number(position + NEARER);
    string_low += ", ";
    string_low += QString::number(depth_low);
    measurement_low << string_low;

    position += STEP;
    std::cout << string_high.toStdString() << std::endl;
    std::cout << z << std::endl;
}
```

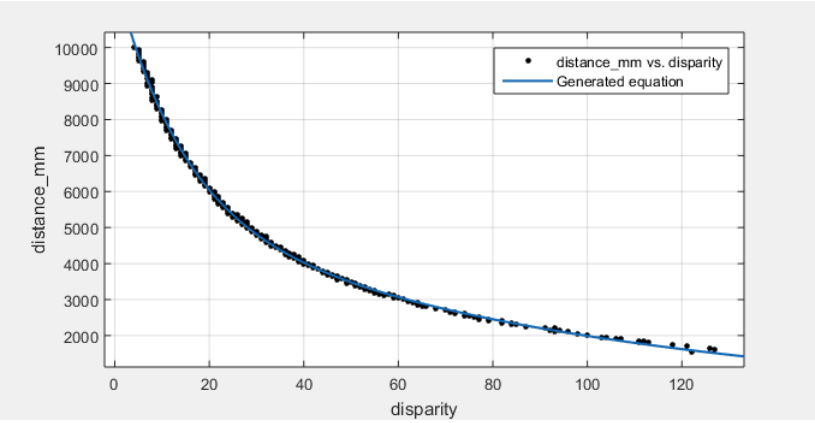


Figure 3.10: Plot showing equation and recorded distance and disparity values.

3.6 Graphics Engine Development

To display the depth information processed, it was decided that a 3D graphics engine shall be implemented in the software. The graphics engine shall be able to render 3-dimensional objects in relevance to each other, for visualization of what the cameras are filming. The idea is that if some known objects are at a fixed position in relevance to the cameras position, the objects can be rendered in the graphics engine. By this, a visualization of structures (a gangway, for instance) can be visualized in relevance with depth information from the stereo cameras. To implement such a graphics engine, it was decided that the OpenGL functionality in Qt was to be utilized.

3.6.1 Graphics engine geometry

To define the 3-dimensional components in the graphics engine, a **SphereVector** class and a Cuboid data structure was developed. The coordinate system in the graphics engine is meant to resemble a representation of a real-world scene. The coordinate system has three axes: x, y and z. The y-axis is meant to represent the height, while the x and z axes form a plane perpendicular to the y-axis.

SphereVector class

A **SphereVector** is defined as an entity in a 3-dimensional space that has a position described as a vector, and a heading angle described as spherical coordinates. In addition to the ability of getting and setting the position and angles of the **SphereVector**, the "rotation matrix" of a sphere vector instance can be obtained. This rotation matrix is defined as the transformation matrix that can be applied to rotate a vector into the same heading as the **SphereVector** instance. The rotation matrix is a QMatrix4x4 instance rotated "theta" degrees

around the z-axis and "phi" degrees around the x-axis. The method of rotating a QMatrix4x4 instance is developed by Qt, thus not explained in this report.

A subclass of the **SphereVector** class is the **OpenGLCamera** class. This class is meant to describe a movable viewpoint in the graphics engine. In addition to the **SphereVector** functionality, this class defines a "up" vector. This is a unit vector to describe what vector shall be considered up with respect to the heading axis. The "up" vector of a camera is defined as a vector aligned with the y-axis (i.e. a $[x, y, z] = [0, 1, 0]$ vector). This means that the view point this camera represents is always projected with the y-axis as its vertical axis. By this, the view point cannot be rolled, only pitched and yawed.

Once a camera is instantiated, its position cannot be set by absolute values. Positioning the camera is always done by relative values, defined in the source code as "movement". To "move" the camera, one of four camera move directions can be specified to describe how to alter the camera position. The four directions are "forward", "backward", "left" and "right". "Forward" and "backward" implies that the camera new position shall be somewhere along the heading axis, where "forward" means towards its heading and "backward" meaning the opposite direction. The "left" and "right" directions are directions perpendicular to both the the y-axis and the heading axis.

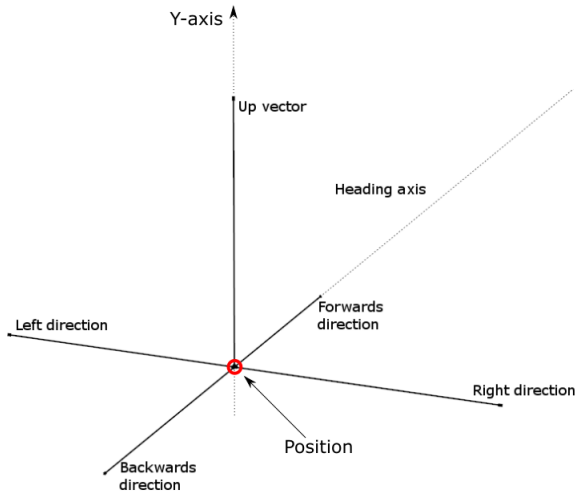


Figure 3.11: Illustration of the camera with its up vector, heading axis and movable directions

The camera can also be "moved" by specifying a relative x,y and z-value to increment the current position with. In addition the camera heading angles can also be "moved". Another key method of the camera class is the ability to obtain the "facing vector" as a 3-dimensional vector. This is a vector describing a position distanced one unit from the position of the camera aligned with the heading axis. Whenever the cameras angles or position coordinates are altered, the camera "facing" is recalculated accordingly.

Cuboid data structure

The Cuboid data structure defines a 3-dimensional cuboid consisting of six rectangular faces at right angles to each other. It has eight 3-dimensional coordinates describing the intersections of the faces. Each face is defined as a cuboid "side", as a enumeration declaration. The six sides are defined as "front", "back", "left", "right", "top" and "bottom". The sides are defined by the initial coordinate position of their four corners in relevance to the other sides corners in the three-dimensional space. By this definition the sides are identified as follows:

- "front" side: high z-value,
- "bottom" side: low z-value,
- "left" side: low x-value,
- "right" side: high x-value,
- "top" side: high y-value and
- "bottom" side: low y-value.

This way, the eight coordinates describing the intersections of the faces are identified by a combination of labels for the faces that are intersecting. This way, the eight coordinates are defined in the source code as:

- "front bottom left",
- "front bottom right",
- "front top left",
- "front top right",
- "back bottom left",
- "back bottom right",
- "back top left",
- "back top right",

The constructor of the Cuboid takes four parameters: A **SphereVector** instance describing its position and heading, a width, a height and a depth. The width represents the distance between the "left" and "right" sides of the cuboid. The height represents the distance between the "top" and "bottom" sides of the cuboid, and the depth represents the distance between the "front" and "bottom". The **SphereVector** instance describes both where the center of the cuboid is position and how the cuboid is facing, thus modifying its eight position coordinates accordingly in the Cuboid constructor.

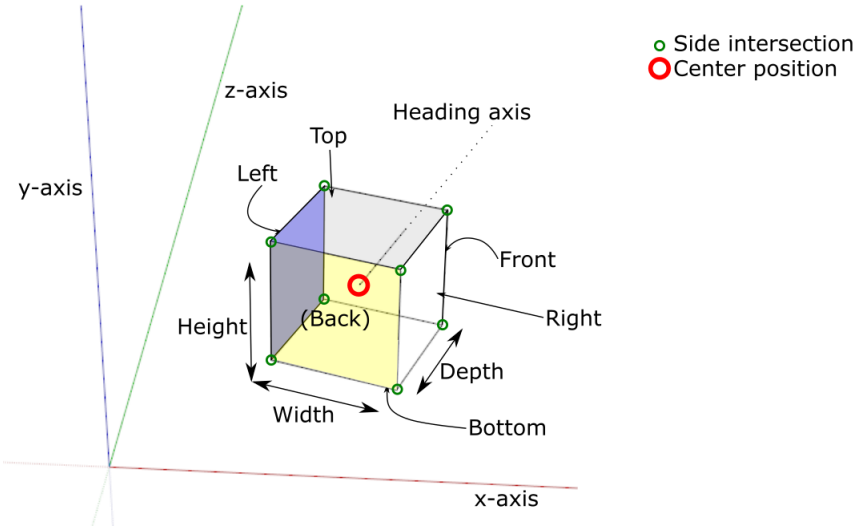


Figure 3.12: An illustration of a cuboid's six faces and eight face/side intersections in relevance to its position and heading axis

3.6.2 Using OpenGL in Qt

To create a OpenGL widget in Qt, the Qt module "opengl" is added to the project. Then, a QOpenGLWidget instance can be created. This instance is used for rendering the graphics in an OpenGL application.

A class subclass of the **QOpenGLWidget** class was then created, named **OpenGL Widget (OpenGLWidget)**. This class has the ability to draw a Scene. When an instance of this class is initially shown it is initialized, thus performing the following actions once:

- initialize Qt OpenGL functionality,
- set the default color (i.e. the background of any scene that is drawn),
- compile and load the shader programs,
- enable depth testing and
- enable face culling

The **OpenGLWidget** is also responsible for handling widget resizing events and set the projection matrix (view) accordingly.

Using OpenGL Shaders

Two shader program for the OpenGL application was created, a vertex shader and a fragment shader. Both the shader programs have the ability to draw to types of vertices; colored and textured. The additional functionality to specify and draw transparency in colored objects was also developed and implemented. The fragment shader checks the color of a fragment, and if it is approximately

entirely green with low red and blue values, its alpha value is set to 0. The vertex shader is responsible for transforming every vertex by a "model view transform" matrix to calculate the projected vertex form.

3.6.3 Scenes and various drawable objects

The Scene class was developed with the intention of holding multiple drawable entities (referred to as "scene objects"), along with a **OpenGL Camera (OpenGLCamera)**. When "drawing" a scene, it will be displayed in a specific **Qt OpenGL Shader Program (QOpenGLShaderProgram)** with a projection. This issues an iteration over each of the scene objects, drawing each one in its appropriate place/coordinate in the scene. The OpenGL camera describes what position and direction the user of the scene is looking. The source code for this process is as follows:

```
void Scene::draw(QOpenGLShaderProgram *program,
                 QMatrix4x4 *projection)
{
    /* Calculate model view transformation,
     * in accordance with the OpenGLCamera field member */
    vMatrix.setToIdentity();
    vMatrix.lookAt(camera->getPosition(),
                   camera->getFacing(),
                   camera->getUp());

    // Set modelview-projection matrix in the shader program
    program->setUniformValue("mvp_matrix", *projection * vMatrix);

    // Iterate through all objects
    QVectorIterator<Object *> iterator(objects);
    while (iterator.hasNext())
    {
        // Draw the current object in the iteration
        iterator.next()->draw(program);
    }
}
```

Note that the camera of a scene is only a reference to the scene, and the scene itself never alters any data of the camera, like the camera position or view angles. The Scene class also inherits from the **TextureHandler** class, which enables it to load and hold images as OpenGL textures, along with the ability to bind and unbind the textures when they are required for drawing objects in a scene.

Objects of type **Scene Objects (Scene::Object)** has only one purpose; being drawn in the appropriate shader program. Any object that shall be rendered in a scene needs to inherit from this class. Two classes, the **Buffered Scene Object (BufferedSceneObject)** and the **Textured Coboid (TexturedCuboid)** are derived from this class.

Buffered scene objects

The **BufferedSceneObject** has the capability of allocating data in a OpenGL Buffer Object, which can be used to store various data required to draw the

OpenGL objects.

Inheriting from this class is a abstract base class defined as a "Surface" class. A surface is defined as a vector of vertices, representing a mesh of cells. Each cell intersection or corner has one vertex, which means each cell is defined by four vertices. The vector of surface vertices is initially empty, and vertices in a surface is added one vertex at a time. The number of vertices a surface holds is an input parameter to its constructor. This is defined as the surface size as *columns* and *rows*. By this principle, the vector of vertices in a surface can be treated as a 2-dimensional matrix with each quantity containing a vertex. Despite the vertices being stored in a one-dimensional vector and not a two-dimensional matrix, the matrix model of storing the vertices is used for explanation because it can resemble the surface mesh as consisting of vertices in each mesh intersection.

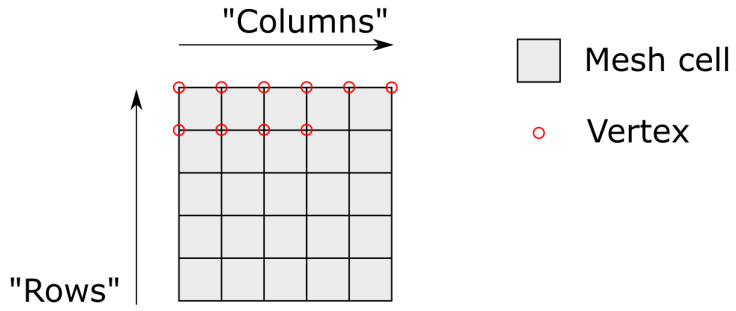


Figure 3.13: An example surface mesh with six rows and six columns, with ten of its vertices added

Surface specifications

Each time a vertex is added to the surface, the vertex gets a unique index v_n which is also stored and indices are calculated and added to the surface's list of indices. The index of a vertex corresponds with the number of vertices, meaning that the n-th vertex added has an index of n. The point of storing the surface indices is that the various surface types require these indices to specify (to the OpenGL shader program) how to draw the surface. This is defined as indexing.

In addition, a vertex counter is incremented. The vertex counter keeps track of how many vertices the vertex matrix has in a specific row, and how many rows are filled with vertices. Starting with the top row, vertices are added left to right in the vertex matrix. Once a row is filled with vertices, the empty row directly below it is selected to be filled (i.e. the vertex counter is incremented by one row). In the source code the number of rows filled with vertices in the vertex matrix is defined as y while the number of vertices filled in the current row is defined as x.

The two subclasses of surface developed in this application each draws their surface mesh as triangles, where each triangle has a facing defined by the order of the vertices that make up the triangle. The three vertices in each triangle must have a counter-clockwise winding, meaning that the order of the vertices rotate counter-clockwise around the triangle's center. Each cell with its four

corresponding vertices is drawn as two triangles with adjacent hypotenuses. These four vertices in relevance to each other are defined in the source code as the "top left", "top right", "bottom left" and "bottom right" vertices of a cell as illustrated in figure 3.14.

Note that these vertex indexing definitions are both invalid and irrelevant unless the vertex matrix has at least one row filled with vertices and at least one vertex added to the current row. By the source code definitions this occurs when $x > 0$ and $y > 0$ are true statements. If any of the $x > 0$ and $y > 0$ statements are false, indexing is skipped. The method of adding a vertex has another boolean input parameter defined as "omit". This parameter defines whether the added vertex should not be drawn. The reason for implementing this feature was to build an entire surface but hiding some vertices in the surface, for example when obscure position coordinates should be invisible.

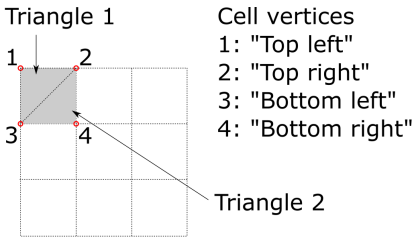


Figure 3.14: The four vertices of the first surface mesh cell in an example surface with a 3 by 3 vertex matrix

Given that a vertex with index v_n is added to the surface, identifying the four vertices in relevance to the most recently added vertex is done by the following formulas:

1. "Top left": $v_{n-3} = (y-1) * columns + x - 1$.
2. "Top right": $v_{n-2} = (y-1) * columns + x$.
3. "Bottom left": $v_{n-1} = y * columns + x - 1$.
4. "Bottom right": $v_n = y * columns + x - 1$. This is the index of the most recently added vertex.

Each time a vertex is added to a surface vertex vector a whole cell is defined for that surface mesh. This means that for each such vertex added, indices indicating vertices of two triangles has to be added. To maintain the principle of sorting the vertices in a counter-clockwise winding, the indices of the vertices from the vertex matrix to describe the first triangle are:

1. v_{n-1}
2. v_n
3. v_{n-2}

The next three indices are:

1. v_{n-1}

2. v_{n-2}
3. v_{n-3}

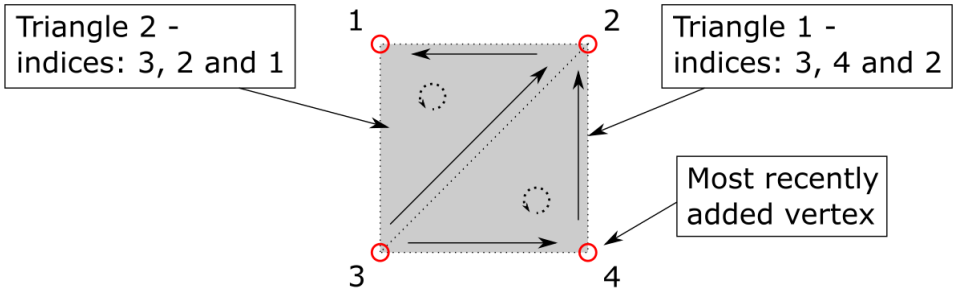


Figure 3.15: Example of the 6 indices calculated for describing the two triangles of a surface mesh cell when a new, fourth vertex is added

Next, these six indices are added to the vector of indices in the surface. Once enough vertices has been added to the surface to fill the matrix (i.e. a number of $columns * rows$ vertices has been added), the surface is considered to be "full". Then, the surface stores all its vertices and indices in its corresponding **BufferedSceneObject** buffers. This means that the surface vertex and index data is allocated in the GPU memory. This only occurs if the boolean field of a surface instance is set to true. The point of the ability to disable this is when surface data is generated without the surface being actually drawn in the graphics engine. The Surface class does not allocate the vertex data, only the index data. However the vertex data also has to be allocated in the GPU, so the Surface class has a pure virtual function to issue any subclass instance to act when the surface is full.

Note that the Surface class does not store the vertices itself, only the vertex indices. The reason for this being that different types of surfaces required different types of vertices.

Colored and textured surfaces

The **Textured Surface (TexturedSurface)** class and a **Colored Surface (ColoredSurface)** class was created, both inheriting from the **Surface** superclass. A **ColoredSurface** has the feature of having a specific color for each vertex in its surface, while a **TexturedSurface** has the feature of having a specific texture coordinate for each vertex in its surface. Initially a textured surface was used to produce a 3-dimensional model of the depth information, but a colored surface was used instead. While the **Surface** class has the responsibility for generating and adding indices, the **ColoredSurface** and **TexturedSurface** classes have the ability to add colored vertices and textured vertices respectively. Being a Surface subclass, these two classes have to implement the **filled()** method, which is called when a surface has all its vertices specified. A filled surface instance can be "re-filled" with vertices by calling the Surface

reset() method. This method clears all the surface indices, and calls the virtual Surface method **resetVertices()** to clear the surface vertices, which subclasses of the **Surface** class have to implement.

Textured cuboids

The **TexturedCuboid** class is created to be able to draw a cuboid with specific textures on its sides.

A **Sky Box (SkyBox)** class inherint from the **TexturedCuboid** was also developed. This class represents a cuboid with introverse faces. This means that the textures applied to the cuboid is only visible from inside the cuboid. Instead of defining a texture for each face, a **SkyBox** instance only uses a single texture and splits the texture into its six faces. To make a **SkyBox** texture, the texture image has to be constructed as a four by three grid by the following template:

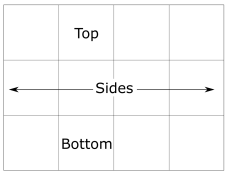


Figure 3.16: Sky box texture tile layout. Each tile represents a face inside the sky box.

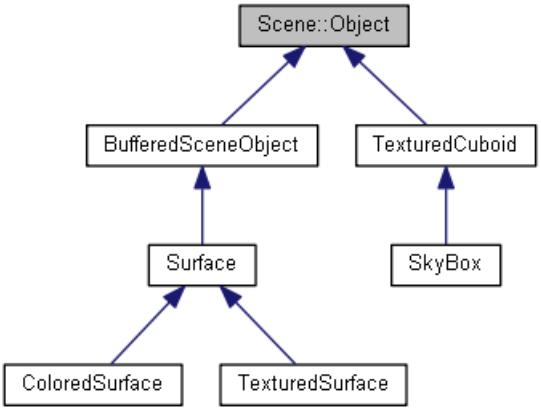


Figure 3.17: Inheritance diagram of the Scene::Object class

3.6.4 Textures

For visualizing textured objects in a scene, textures are added to a texture handler. As the Scene class inherits from the **TextureHandler** class, all textures used in this project is added to the scene.

Sky box texture

For the sky box object, a texture resembling clouds is created by masking a bright blue layer over a slightly darker blue layer with the "Difference clouds" render effect in GIMP. This texture is used for the top (i.e. the face with the highest y-valued face center coordinate) of the sky box. For the bottom (face with the lowest y-valued face center coordinate) a dark blue layer is rendered with the "Lava" effect in GIMP to resemble the upper surface of an ocean. For the remaining sides, a gradient effect between the cloudy texture and the ocean texture is made. These textures together are meant to resemble a sky fading gradually into a dark, blue water or sea surface. The images are conjoined into a single "sky box" picture file.

Gangway texture

For the gangway structure, a simplistic, orange rectangle with black edges and letters spelling "Gangway" is used to represent the gangway model texture.

3.6.5 Gangway, Atmosphere and Camera Input in 3-d

With the **Scene**, **Scene::Object** and all its derived classes implemented in the application, the **SVWidget** utilizes these classes to illustrate a scene consisting of:

- a textured cuboid resembling a gangway structure,
- a colored surface resembling whatever the cameras are filming and
- a "sky box", a seemingly realistic atmosphere.

One unit in the scene coordinate system is meant to resemble one metric centimeter, with the y-axis aligned with the distance from the bottom base to the top of an object (i.e. the height). Given that the optical axis of the two cameras are parallel, the origo of the scene coordinate system is in the center of a line between the center of projection of both the cameras. The z-axis of the scene coordinate system is aligned with the optical axis of the two cameras. By these definitions, an example gangway of 12 metres length, 2 metres height and 1.5 metres wide with the two cameras mounted underneath its base and 4 metres from the forward forward tip and tilted its optical axis tilted approximately 30° down can be projected in the scene as illustrated in figure 3.18.

Data from the disparity maps are projected as a colored surface positioned in z-axis greater than the gangway, facing the origo of the coordinate system, thus facing the gangways camera objects. In addition, a "sky box" is surrounding the gangway and the colored surface, to illustrate an atmosphere for the gangway to be in.

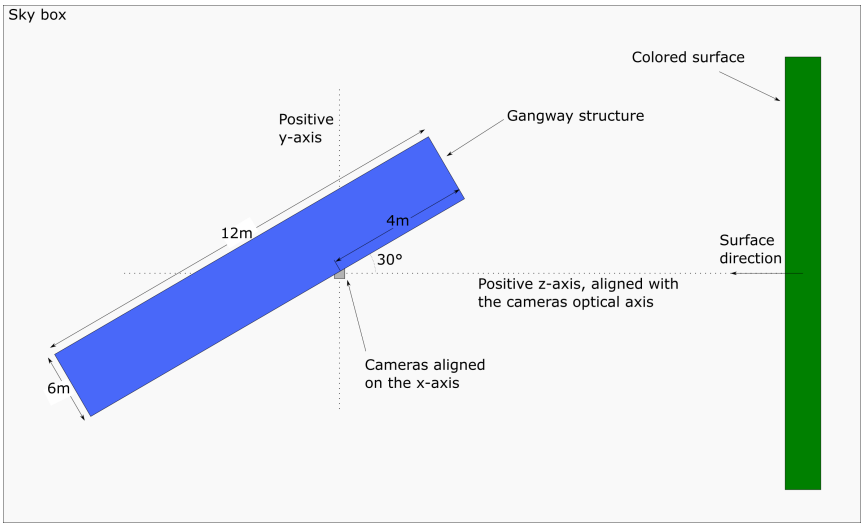


Figure 3.18: Sketch of example gangway (blue), colored surface (green) and a surrounding sky box (white)

3.7 Rendering Disparity Maps in 3D

3.7.1 Building and Rendering the Stereo Visualizer scene

Scene setup

The application widget **(SVWidget)** class defines (in its constructor) the size, position and facing angles of the disparity map colored surface as a Cuboid instance. Once defined, the Cuboid instance is passed to the **SVThread**. The **SVEngineWidget** field of the **SVWidget** defines the gangway and sky box objects are constructed once the **SVEngineWidget** is started by the **SVWidget**.

3.7.2 Generating colored surface data

When in **Re-project** mode, the **SVThread** issues a **DepthMapper** instance to calculate a disparity map from the two images the **SVThread** obtains. Once the **DepthMapper** instance has calculated a disparity map from the images, the **SVThread** obtains data for the **ColoredSurface** instance by calling a method in the **SurfaceBuilder** class. This class is non-instantiable, and contains a set of static methods for specifying the data a previously created target surface from disparity map data.

Generating from a disparity map

One method in the **SurfaceBuilder** class takes a disparity map as a single channel (gray-scaled) OpenCV matrix ("cv::Mat") instance and iterates over all the pixels in the matrix. For each pixel in this matrix, a **ColoredVertex** object is instantiated and positioned in a position mapped to represent the pixel. This means that the corners This means that the upper left pixel has a corresponding upper left colored vertex in the surface, the lower left pixel has a corresponding lower left colored vertex, and so forth. This principle is illustrated in figure 3.19.

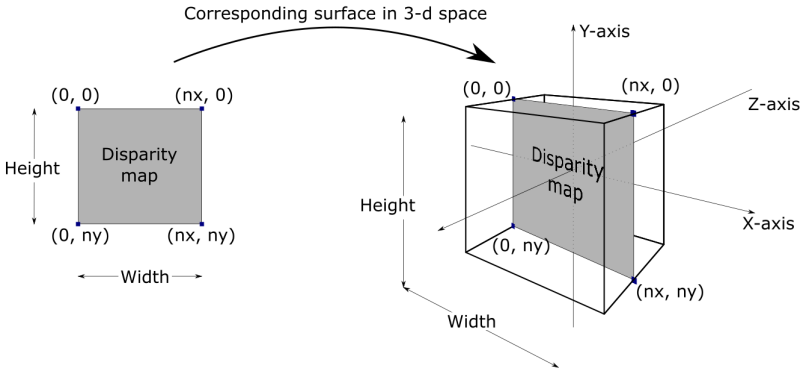


Figure 3.19: Disparity map with a corresponding 3D surface

The 3-dimensional boundary sizes of the surface is specified as a **Cuboid** instance with a "width", "height" and "depth" referred to as the surface "space". These are the distances of the cuboids sides along the x, y and z-axis' respectively. For a (n_x, y_n) pixel of a width by height disparity matrix, the corresponding 3-dimensional vertex position is calculated by the following formula:

$$(-scale_x * (width/2) + scale_x * x_n, -scale_y * (height/2) + scale_y * y_n, z_n)$$

The $scale_x$ and $scale_y$ is a remapping factor depending on the "width" ($space_w$) and "height" ($space_h$) of the cuboid instance. This calculated by the following formulas:

$scale_x = space_w/width$ and,
$scale_y = space_h/height$

Each pixel in the single-channel disparity map contains a 0-255 value. This value corresponds with the calculated distance measurement of that pixel from when the disparity map was processed, thus indicating the distance from the camera to this pixel. With this pixel value, the position z-coordinate (z_n) of the corresponding vertex is calculated. Discovering the formula for calculating the "real-world" distance from the pixel value was attempted, but for this particular method the z-value formula was simplified. Given that $space_d$ is defined as the surface space cuboid "depth" and a variable v_p is the remapped pixel value (from [0-255] to [1-0]), the formula is simplified to:

$$z_n = space_d/2 - v_p^{1/2} * scape_d$$

Surface data generating from a DepthMapper instance

Another static method of generating surface data was developed in the **SurfaceBuilder** class, with minor differences from the previous one. Instead of calculating the z-value of the surface vertices from a complete disparity map, this method uses a DepthMapper instance and requests the calculated depths of each possible coordinate for a disparity map retained by the **DepthMapper** instance itself. This enables the **DepthMapper** instance to be responsible for calculating the "real-world" z-value rather than the **SurfaceBuilder** class to define it.

In this method, the width and height (in pixels) of the disparity map the **DepthMapper** instance has to be obtained. After this, a y-value (y_n) is iterated from the disparity map height to 0, and in each sequence a x-value (x_n) is iterated from 0 to the width of the disparity map. For each sequence the pixel distance (in millimetres) is obtained from the **DepthMapper** instance. Given that this distance value is v_p millimetres, the z-value of the vertex is calculated by the following formula:

$z_n = space_d/2 - v_p/10$

Note that each axis integer in the coordinate system of the scene represents a centimeter. Therefore the distance value v_p is divided by 10. This distance then represents the z-value of the vertex position, from the "front" side of the space cuboid. The x and y-values are calculated the same way as described in section 3.7.2.

Transform vertex position

The position of each colored vertex is transformed to correspond with the surface "space" cuboid position and its facing angles.

Generating the surface color

For both methods of the **SurfaceBuilder** class described here, a key input parameter is a 3-channel OpenCV matrix representing the color values of each pixel that is generated in a disparity map. When a colored vertex is constructed with a position z-value obtained in the **DepthMapper** instance by a (x_n, y_n) value, the same (x_n, y_n) specifies the pixel of the color matrix which sets the vertex color. By this the colored surface projecting the distance measurements gets the same color as the objects present in the image. The left frame is used for coloring the surface, as the left image is used as the base when generating the disparity maps.

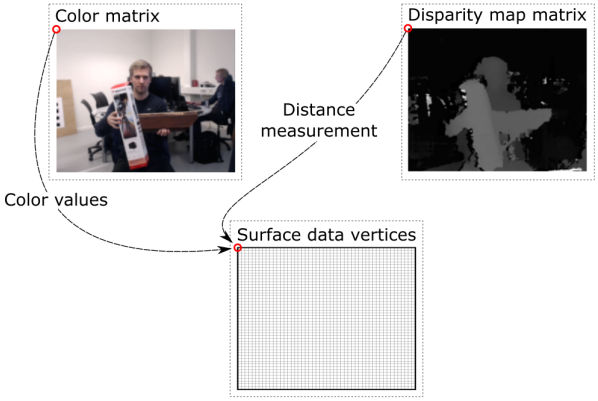


Figure 3.20: Example of generating surface data from pixels in a disparity map.

Filtering the data

Noise in the disparity maps occurs because of various causes. This noise has a dramatic impact on the vertices calculated when the surface is generated. A thresholding filter is implemented into the surface data generating methods, to prevent relatively high or low values (also referred to as "spikes") in the disparity maps from increasing or decreasing the calculated position coordinate z-value of a vertex. When calculating a vertex for the surface, the corresponding pixel from the disparity map is obtained. In addition, the pixels directly adjacent to the selected pixel is also obtained. All these pixels are checked to see if any contains a "spike". If any of these pixels does contain a spike, a boolean variable named **omit** is set to true. This variable is passed to the surface when adding the vertex. If this variable is true, the vertex is added but the indices for this vertex is omitted. This prevents the vertex from being drawn, thus filtering spikes from the surface.

Using the surface data

Once the surface data is set according to a disparity map matrix or a **DepthMapper** instance, a **surfaceReady** signal is emitted from the **SVThread** instance. The **SVWidget** listens to this signal, and obtains the surface data once it is received. Then, it updates the data in the surface that exists in the scene. The scene is re-drawn, and the surface with its new data is projected in the graphics engine.

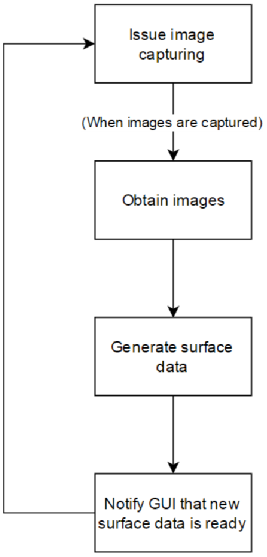


Figure 3.21: The tasks performed by a SVThread instance when **re-project** mode is activated.

3.8 Implementing the Stereo Visualizer Widget in a Qt Application

To use the **Stereo Visualizer Widget** developed, a Qt Application created. The main method of this application instantiates a **QApplication** and **SVWidget** object. The **SVWidget show()** method is called for the **SVWidget** object, and the the "exec" method is called for the **QApplication** object. Thus, showing the widget in a Qt application.

3.9 Test Setup

The rate of how much information the application can process are documented in this report. To measure this rate, some functionality was added to the **run()** method of the **SVThread** class. The time since images were previously captured also represents the time it has taken to process the images. Therefore, the rate (r) of process loops per second can be measure with the following formula:

$r = 1/t$

Where t represents the time since images were previously captured in seconds, thus r calculates number of process loops per second. Note that this rate is calculated after a time constraint, therefore the rate will never exceed to a point where minimum time between image capturing is exceeded.

To test the processing rate of certain process steps, others are excluded from the application by commenting out source code. The rate of each processing step is

also calculated with different image resolutions to document how the resolutions affect the processing rate.

3.10 Documentations and Source Code

To make the widget and its features easy to use and maintain, all software developed in this project were documented using a documentation generator. The documentations were generated as a complete web page readable to any modern web browser. This web page along with all the source code and textures/images developed during this project was stored on a USB flash drive.

Chapter 4

Results

4.1 Stereo Visualizer Widget

Running a Qt application with the Stereo Visualizer Widget showing is functional. If the cameras are not connected, a warning is displayed in the console output of the application. The default mode activated is the **Stereo video** mode, thus the application shows the images captured from both cameras. Selecting tabs and starting/restarting the **SVThread** instance has all the planned functionality.

4.2 Stereo Video Feed

Images from left and right cameras are shown in the GUI as shown in figure 4.1. The images are captured at approximately the same time, even if the total frame rate is reduced. Images from both left and right cameras displayed simultaneously in each half of the GUI as planned. Connecting the controller to a Samsung 3D-TV, enabling 3D mode with vertical split and using the 3D-glasses gives the user depth perception from the cameras. If the cameras are aligned or not facing the same objects, depth perception is not achieved.

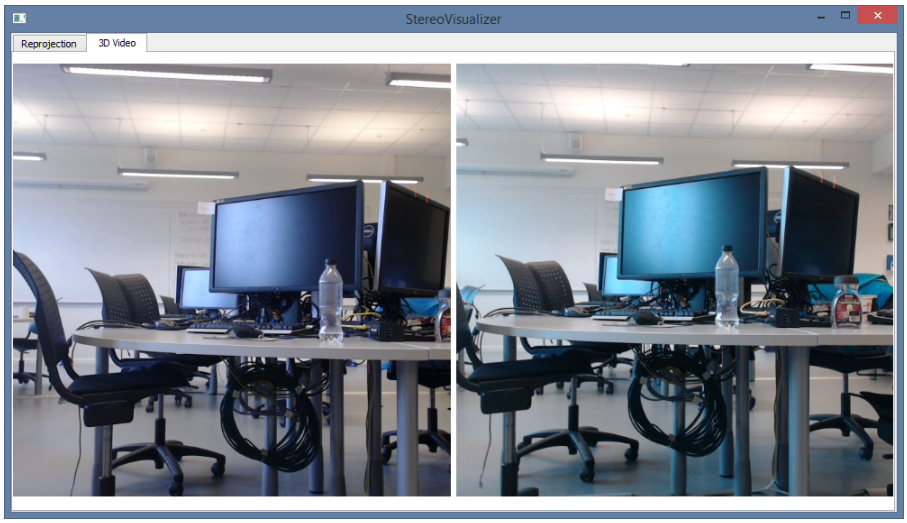


Figure 4.1: Screen shot from the Stereo Visualizer Widget shown, with its **Stereo video** mode activated

4.3 Disparity Maps

The stereo images are rectified using the parameters from a successful calibration, and stereo correspondence is found. The disparity maps that are generated clearly show differences in depth, but contain a lot of noise in areas where stereo correspondence is incorrectly detected. This is usually due to repetitive patterns in the picture. Distance is computed from the disparity values, and the results is quite accurate in the areas without noise. A disparity map is shown in figure 4.2.

Figure 4.2: A complete disparity map calculated from someone sitting in front of the camera holding two objects at different distances from the cameras. Notice the noise as completely black and white areas in the disparity map.

4.4 Rendering the Gangway Structure, Sky Box and Surface in the Graphics Engine

Instantiated **Scene::Object** objects can be added to a scene and displayed in the developed OpenGL graphics engine. The scene created for visualizing a gangway structure, sky box and disparity map surface is displayed, and the user can move the camera around in the scene to visualize the scene from different viewing angles with a mouse and a keyboard. Moving the camera position is concurrent with the image processing producing the disparity maps and altering the shape of the disparity map surface.

Figure 4.3: Screen shot of a Qt application showing a SVWidget instance.

4.5 Processsing Rates

These tests are carried out on a Asus N43SN laptop with the following specifications:

- Intel Core i5 2410M CPU
- 8 Gb DDR3 1333MHz SDRAM memory
- NVidia GeForce GT 550M 1GB DDR VRAM graphics processing unit

The cameras connected are two Logitech HD Pro Webcam C920 with a maximum frame rate of 30 frames per second. The processing rates here are represented with a preset ($width * height$) frame resolution in pixels with a corresponding result.

4.5.1 Stereo video

The rate of capturing and displaying images from both cameras in the widget are limited by the hardware of the cameras. For these tests the maximum image frame rate for each camera is 30 frames per second, thus this is the same rate of which the graphical user interface displays the captured images. The rate of this process is constrained in the source code to 30 frame per second for each camera. If this constraining value is reduced to produce more than 30 frames per second, images from the the cameras are duplicated and the same images are processed and displayed more than once. Because of this, a processing rate higher than the camera frame rate is considered inappropriate. This is the case for any frame resolutions possible for the cameras.

4.5.2 Generating disparity maps

The processing rate of capturing images and generating disparity maps from the images are as follows.

- 480 * 320 pixels: 8.5 - 9 disparity maps per second
- 640 * 480 pixels: 2.4 - 2.7 disparity maps per second
- 800 * 600 pixels: 1.55 - 1.6 disparity maps per second
- 1280 * 720 pixels: 0.65 - 0.7 disparity maps per second

4.5.3 Generating surface data

The processing rate of capturing images, generating disparity maps and generating surface data from disparity maps into surface data instances are as follows:

- 480 * 320 pixels: 6.4 - 7 instances per second
- 640 * 480 pixels: 1.92 - 2.14 instances per second
- 800 * 600 pixels: 1.26 - 1.3 instances per second
- 1280 * 720 pixels: 0.54 - 0.59 instances per second

4.5.4 Inserting surface data

After surface data is generated, the data is inserted into the disparity map surface existing in the scene. This is the final step for a **SVThread** processing loop to finish. After this step is included in the SVThread process loop, the processing rate becomes:

- 480 * 320 pixels: 6.0 - 6.7 loops per second
- 640 * 480 pixels: 2.1 - 2.2 loops per second
- 800 * 600 pixels: 1.17 - 1.26 loops per second
- 1280 * 720 pixels: 0.54 - 0.59 loops per second

Chapter 5

Discussion

5.1 Camera Rig

If the cameras are physically repositioned in relevance to each other after a calibration, the calibration becomes invalid unless the cameras can be moved back to their approximately exact position. During the development of this application, the cameras had to be calibrated 50 times which proved quite tedious. The formula calculated for distance measurement differs for different camera positions, and calculating this formula is an even more tedious process than the calibration. For further development of this project, a rigid camera rig which enables permanent fastening of the cameras should be made.

5.2 Noise in Disparity Maps

Disparity maps calculated with this application contains some areas with invalid distance measurements, referred to as noise. The noise influences the resulting 3-dimensional representation of the disparity map quite dramatically, and filtering the noise in the disparity maps seems quite crucial for a useful and satisfying end result.

The filtering process should not decrease application performance too much. As the surface building process iterates through every pixel in the disparity map, filtering could perhaps be implemented here without damaging the processing rate too much. A relatively primitive filter was added here. However this filter was not thoroughly developed, and did not improve the details of the disparity map. Another possibility to remove the noise could be to use some sort of edge detection algorithm on the rectified images before computing stereo correspondence, to clarify the areas of interest for the disparity map.

5.3 Graphical Enhancements

The gangway structure cuboid could advantageously be further developed to resemble a gangway structure more than it currently is. This can enhance the realism of the visualization, thus make it look more professional. Dynamic scene objects, like a textured cuboid with animated movement could also be added to enhance the look and feel of the graphical environment.

Controlling the viewing angle in the graphics engine is currently only done by a mouse and keyboard. For use in an actual gangway, controlling the camera should be a less occupying for the operator. The best solution here could perhaps be an automatic movement based on what viewing angle gives the best overview of the landing and the gangway, with the feature of manual override. That way, the gangway operator would not have to worry about repositioning the camera when operating the gangway with aid from the **Stereo Visualizer Widget**.

5.4 Processing Rate Optimization

5.4.1 Image resolution versus processing rate

From the resulting processing rates, it seems clear that the resolution of the images captured for processing influence the processing rate quite dramatically. A relatively high resolution results in a relatively low processing rate, and vice versa.

Consider that the disparity map is not completed until images are captured and the disparity mapping process is completed. This means that using t seconds to generate the disparity maps means the disparity maps represent images captured t seconds ago. For low processing rates, the gangway or the gangway landing may have moved a significant distance before the disparity is completed. The gangway operator is dependant on real-time data from the visualization. If the processing rate is too low, the data can be considered "outdated" before its fully processed.

Relatively low resolution images results in a less detailed disparity map where noise filtering have to be fine tuned not to invalidate the distance measurements. Therefore it seems crucial to find a balance between a high resolution and high processing rate.

5.4.2 Multi-threading

As multi-cored processing units are common, process of building disparity maps should be multi-threaded to utilize all cores of a CPU . This can be achieved by splitting the images into different sections, and each thread can process its own section of an image. As the algorithm search for stereo correspondence horizontally from left to right, the sections have to be split horizontally. This is to ensure that objects positioned in the intersections of these sections are visible in both images.

Multi-threaded software for generating disparity maps was discovered during the development of this application. However this software splits the images sections both horizontally and vertically into tiles, which did not seem approprioate. Developing a multi-threaded solution for generating disparity maps was planned, but omitted for this application. This feature could be added in further development of the application/solution.

For processing rate optimization, shader programs could also be developed to utilize a controllers GPU to increase the processing rate. This would enable the CPU and GPU to cooperate in the processing, and seems like a reasonable way to increase processing rate.

5.4.3 Benefits of a custom graphics engine

By developing a custom graphics engine, the graphics engine was developed to be as light-weight as possible. Implementing a pre-developed graphics engine means adding a lot of excess functionality that may slow down performance.

5.5 Other Areas of Utilizing this Technology

This application or at least its ideas or technology could perhaps be applied to other disciplines where depth perception through stereo cameras is appropriate. Cranes or other construction machinery where the depth perception for the operator is crucial could benefit from this technology just as much as gangway operation.

5.6 Comparing other technologies

When comparing this technology to other technologies for 3-dimensional mapping by distance measurement, the cameras and the controller needed for this application is a relatively low-budget solution. For the cameras to be mounted in a offshore or other hazardous environment (like on the tip of a gangway), the cameras would perhaps have to be of a more rigid type. This could result in increased costs for the system, but still considered to be a cheaper alternative than for instance laser 3-dimensional mapping or ultasound sensors. Producers of high-definition cameras are commercial and competitive, thus providing high-quality cameras for a relatively low cost.

Chapter 6

Conclusion

6.1 Stereo Video

Compared to ordinary camera systems which only display a regular image, stereo video displayed on a 3D-screen can absolutely be used to enhance depth perception from cameras.

6.2 Distance Measurement from Disparity

Distance measurement using disparity maps is fully possible. In order get more precise measurements and useful results, the disparity maps have to contain less noise. In addition, the processing rate should be increased. This feature needs further development to be considered a useful solution for docking assistance.

6.3 Visualizing The Gangway

From the distance measurement data referred to as disparity maps, 3-dimensional models can be rendered. Regardless of the quality of disparity maps, visualizing the landing and a representation of the gangway in a graphics engine is a good way of providing the gangway operator visual aid for docking. To use such a system, changing the viewing angle of the visualization automatically for better overview should be implemented.

6.4 Assisting Gangway Docking

The current results of disparity maps and processing rate along with the other desired functionality for the applications re-projecting mode needs more development before it can be used for assisting a gangway docking. However, depth

perception is satisfyingly enhanced in the application's stereo video mode and the application can definitely be used for this purpose.

6.5 Proposals for Further Work

- Create a rigid camera rig to avoid the problems caused by moving the cameras.
- Implement the feature of filtering the disparity maps. This should be implemented in either the disparity map generating process or the process of generating surface data to display the disparity map surface with.
- Develop different ways of controlling the viewing angle in the graphics engine, perhaps automating the viewing angle movement.
- Accomplish higher processing rate by use of multi-threading in the disparity map process, or by utilizing the GPU for processing.
- Research different cameras, and attempt to discover what cameras are best suited for the purpose of assisting gangway docking.

Bibliography

[1] Cmake. http://www.cmake.org/.

[2] Icd official website. http://www.icd.no/.

[3] Logitech hd pro webcam c920. http://www.logitech.com/en-us/product/hd-pro-webcam-c920.

[4] Matlab description. http://se.mathworks.com/products/matlab/?s_tid=hp_fp_ml.

[5] Matlab stereo calibration app. http://se.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html.

[6] Open graphics languange. https://www.opengl.org/wiki/Main_Page.

[7] Opencv documentation. http://docs.opencv.org/.

[8] Opencv introduction. http://docs.opencv.org/modules/core/doc/intro.html.

[9] Opengl context. https://www.opengl.org/wiki/OpenGL_Context.

[10] Opengl vertex shader. https://www.opengl.org/wiki/Vertex_Shader.

[11] Oxford dictionaries. http://www.oxforddictionaries.com/.

[12] Qt project web page. https://www.qt.io/.

[13] Uptime international official website. http://www.uptime.no/.

[14] David J. Barnes and Michael Kölling. *Objects First with Java: A Practical Introduction Using BlueJ*. Pearson, 2011.

[15] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly Media, 2008.

[16] Paul Deitel and Harvei Deitel. *C++ How to Program (8th Edition)*. Prentice Hall, 2011.

[17] Qt Learning Digia. Opengl tutorial. ftp://ftp.informatik.hu-berlin.de/pub/Linux/Qt/QT/developerguides/qtopengltutorial/OpenGLTutorial.pdf, 2013.

[18] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, 2007.

[19] Steve Kleiman. *Programming With Threads*. Prentice Hall, 1996.

[20] Nils Olsson. *Praktisk rapportskriving*. Fagbokforlaget, 2014.

[21] Alan Peters. Lectures on image processing. `http://eeweb.poly.edu/~onur/lectures/lectures.html`.

[22] Amund Ringvold. Dybdesyn. In *Store Norske Leksikon*. Foreningen SNL, 2009.

[23] Knut A Rosvold. Tredimensjonal. In *Store Norske Leksikon*. Foreningen SNL, 2015.

[24] Ling Shao and Jungong Han. *Computer Vision and Machine Learning with RGB-D Sensors*. Springer, 2014.

[25] Abraham Silberschatz. *Operating System Concepts*. Wiley, 2008.

[26] Paul van Walree. Distortion. `http://toothwalker.org/optics/distortion.html`.

[27] Andrew Wellings. *Concurrent and Real-Time Programming in Java*. Wiley, 2004.

Appendix A

Compiling OpenCV 2 with MinGW for Windows

This document describes a step-by-step tutorial on how to install and compile OpenCV with MinGW compiler for Windows.
For this particular project, OpenCV was installed to the C drive of a Windows operating system, thus creating a folder structure as illustrated in A.1.

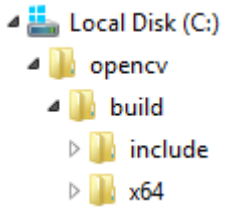


Figure A.1: OpenCV Installation folder structure

Once OpenCV has been installed, the following steps are followed to compile OpenCV.

1. Download and install CMake from the official CMake web page[1].
2. Start CMake GUI.
3. Select "Browse Source" and navigate to the OpenCV "sources" folder. For this particular project this is "C:\opencv\sources".
4. Select "Browse Build" and navigate to the Opencv "build" folder . For this particular project this is "C:\opencv\build".
5. Select "Configure" to select the compilers.
6. Under "Specify the generator for this project", select "MinGW Makefiles".
7. Select the "Specify native compilers" choice and hit "Next ¿".

8. Specify the "C" and "C++" compilers. Note that for this particular project the Qt IDE was installed to the C drive, thus making the following directories apply for the compilers.
9. The "C" compiler is located at "C:\Qt\Tools\mingw491_32\bin\gcc.exe".
10. The "C++" compiler is located at "C:\Qt\Tools\mingw491_32\bin\g++.exe".
11. Hit "Finish" and wait for the inital configuration to complete.
12. Hit "Generate" to generate the MinGW makefiles needed for compilation. Once this is done, CMake can be exited.
13. Open a Windows Command Prompt, and navigate to the OpenCV build directory using the command "cd C:\opencv\build\".
14. Start the compilation with the command prompt command "C:\Qt\Tools\mingw491_32\bin\mingw32-make -j5" and wait for the compilation to finish. Note that the "-j5" argument enables 5 different threads to be applied for this compilation process, thus utilizing a multi-core system as used in this project.
15. Finally, build the OpenCV binaries by the command prompt command "C:\Qt\Tools\mingw491_32\bin\mingw32-make install -j5".

Appendix B

Adding OpenCV binaries to the system PATH

To utilize the OpenCV libraries, the library binary files is added to the controllers "PATH" environment variable. This is done by adding the "C:\opencv\build\x86\mingw\bin" directory to the "PATH" variable by the following steps:

1. Enter computer properties by right-clicking "This PC" icon in the Windows Explorer and selecting "Properties".
2. Enter "Advanced system settings" on the left menu.
3. Select "Environment Variables...".
4. Navigate down to the "Path" variable, select it and click "Edit...".
5. In the "Variable value:" field, add the following to the end of the text: C:\opencv\build\x86\mingw\bin;
6. Apply all changes by hitting "Ok" on all the configuration windows.
7. Restart or log out the current user to reload the PATH environment variable.

Appendix C

Progress Reports

ID301702 Hovedprosjekt	Prosjekt Verktøy for dokking av Uptime gangvei	Antall møter denne periode 1). 1	Firma - Oppdragsgiver Høgskolen i Ålesund /	Side 1 av 1
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 27.01 til 13.02.2015	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Thorbjørn Solberg & Birger S. Pedersen	Dato 12.02.2015

Hovedhensikt / fokus for arbeidet i denne perioden

Drøfte og vurdere alternative forslag til løsninger

Planlagte aktiviteter i denne perioden

- Lage en liste med forslag til løsninger
- Diskutere alternativene og plukke ut de foretrukkede alternativene
- Presentere løsningene for veileder og ICD
- Utføre nødvendig testing av alternativene
- Bastemme hvilke alternativer som skal prioriteres i videre arbeid
- Få tak i nødvendig utstyr, og tilegne seg nødvendig informasjon og kunnskap

Virkelig gjennomførte aktiviteter i denne perioden

- Lage en liste med forslag til løsninger
- Diskutere alternativene og plukke ut de foretrukkede alternativene
- Utføre nødvendig testing av alternativene

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Har behov for å teste forskjellige alternativer litt mer før gruppen kan ta en endelig beslutning og videre prioriteringer.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Hovederfaring fra denne perioden

Objektgjennkjenning krever høy oppløsning og stor datakraft, foreløpig testing har gitt ustabile resultater. Finnes veldig gode løsninger for 3D-kartlegging av objekter med laser og bilde.

Hovedhensikt/fokus neste periode

Bruke stereo-bilder (disparity) for 3D-kartlegging av landing, teste objektgjennkjenning litt mer

Planlagte aktiviteter neste periode

Utvikling av èn løsning: Spesifikasjon, planlegging av utviklingsfase, utvikling og testing.

Annet

Andre vurderte alternativer:

- IR + 3 stk. lys på landing
- 3stk. antenner for triangulærposisjonering
- 3D-modellering av landing med laser

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Tilbakemelding på gruppens plan om framtidig arbeid på å utvikle en løsning

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere

ID301702 Hovedprosjekt	Prosjekt Verktøy for dokking av Uptime gangvei	Antall møter denne periode 1). 1	Firma - Oppdragsgiver Høgskolen i Ålesund /	Side 1 av 1
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 13.02 til 27.02.2015	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Thorbjørn Solberg & Birger S. Pedersen	Dato 26.02.2015

Hovedhensikt / fokus for arbeidet i denne perioden

Spesifikasjon og utvikling av løsning

Planlagte aktiviteter i denne perioden

Spesifisere løsning, benytte 3D-API for visualisering, benytte bildekalibrering og hente dybdeinformasjon fra disparitymaps, utvikle 3D-kart fra stereo-bilder, utvikle algoritmer for objektsporing, utvikle gjennkjenning av svinginger fra objektsporing

Virkelig gjennomførte aktiviteter i denne perioden

Spesifiserte løsning, utviklet 3D-miljø i Qt, benyttet bildekalibrering i Matlab, utviklet 3D-kart fra stereo-bilder

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Bildekalibrering utenfor Matlab (dvs. bildekalibrering i C++ med OpenCV) har vært mer krevd mer tid enn antatt, objekt-gjennkjenning og objektsporing har blitt forskjøvet pga. manglende tid.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Objektsporing og gjennkjenning av svinginger gangveiens bevegelser må forskyves til neste uke.

Hovederfaring fra denne perioden

Planlagte aktiviteter kan ta lengre tid enn forespeilet, Qt har god støtte for 3D-visualisering, enkelt å lage 3D-visualisering for føreren av gangveien. Det mest komplekse med utviklingen av verktøyet later til å bli bildebehandlingen fra stereo-kamera.

Hovedhensikt/fokus neste periode

Videreutvikling av 3D-visualisering og dybdeinformasjon fra disparitymaps.

Planlagte aktiviteter neste periode

Få på plass bildekalibrering og dybdeinfo fra stereo-kamera (disparitymaps) uten bruk av Matlab, utvikle objektsporing og gjennkjenning av svinginger.

Annet

(Ingen tekst)

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Tilbakemelding på prosjektets progresjon og gruppens arbeid så langt.

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere

ID301702 Hovedprosjekt	Prosjekt Verktøy for dokking av Uptime gangvei	Antall møter denne periode 1). 1	Firma - Oppdragsgiver Høgskolen i Ålesund /	Side 1 av 1
Rapport fra prosess Framdriftsrapport	Periode/uke(r) 26.02 til 10.03.2015	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Thorbjørn Solberg & Birger S. Pedersen	Dato 09.10.2015

Hovedhensikt / fokus for arbeidet i denne perioden

Videreutvikling av verktøy

Planlagte aktiviteter i denne perioden

Kalibrere kamera, implementere egengenererte disparity maps til 3D-kart i sanntid, videreutvikle uthenting av dybdeinformasjon fra disparity maps, visualisere gangvei sammen med (i samme rom som) disparity map, objekt-tracking fra egengenererte disparity maps, detektere svingninger i videobilde (mao. i gangveiens posisjon).

Virkelig gjennomførte aktiviteter i denne perioden

Implementert generering av disparity maps til 3D-kart i sanntid, implementert tegning av blokker for å illustrere gangvei foran disparity map.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Å generere disparity maps (tilsynelatende) uten støy er utfordrende, foreløpige resultater på egengenererte disparity maps er vanskelige å bruke da de inneholder mye støy.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

(Ingen)

Hovederfaring fra denne perioden

Bedre forståelse for hva disparity maps inneholder samt hvordan kameraene skal kalibreres.

Hovedhensikt/fokus neste periode

Objeckt-tracking i stereo vision.

Planlagte aktiviteter neste periode

Få på plass en akseptabelt resultat mtp bildekalibrering og egengenererte disparity maps.
Objekt-tracking i 3D, detektering av svinginger i gangveiens bevegelser.

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Tilbakemelding på prosjektets progresjon og gruppens arbeid så langt.

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere

ID301702 Hovedprosjekt	Prosjekt Verktøy for dokking av Uptime gangvei	Antall møter denne periode 1	Firma - Oppdragsgiver Høgskolen i Ålesund /	Side 1 av 1
Rapport fra prosess Fremdriftsrapport	Periode/uke(r) 10.03 til 24.03.2015	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Thorbjørn Solberg & Birger S. Pedersen	Dato 23.03.2015

Hovedhensikt / fokus for arbeidet i denne perioden

Videreutvikling / ferdigstilling av verktøy

Planlagte aktiviteter i denne perioden

Framstille (filtrere) gode disparitymaps fra stereo-kameraer, tracking av objekter i 3D, implementere sann avstandsbedømmelse fra disparitymaps, teste verktøyet på realistiske avstander, bestille inn 3D-briller og teste 3D-vision (bruk av 3D-videofeed i sanntid).

Virkelig gjennomførte aktiviteter i denne perioden

Har fremstillt bedre, men ikke perfekte disparitymaps.
Implementert sann avstastandsbedømmelse av disparitymaps.
Bestillt inn og testet 3D-briller på 3D-TV.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Utfordrende å definere støy og ujevnheter i disparitymaps. Utvikling av 3D-tracking av objekter utgår da det vil ta for lang tid, men løsningen og alternativer vil bli drøftet i rapporten.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

(Ingen)

Hovederfaring fra denne perioden

3D-skjerm med 3D-briller kan være en gunstig løsning for operatøren. Våre tester av denne løsningen viste at denne teknologien gjør det svært enkelt å bedømme avstander i videobildet.

Hovedhensikt/fokus neste periode

Ferdigstilling av verktøy.
Informasjonsinnhenting, drøfting og rapportskriving.

Planlagte aktiviteter neste periode

Implementering av en ferdig løsning.
Oppstart av rapportskrivingsfase, planlegge spesifikke oppgaver for rapportskrivingen.

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Tilbakemelding på prosjektets progresjon og gruppens arbeid så langt.

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere

ID301702 Hovedprosjekt	Prosjekt Verktøy for dokking av Uptime gangvei	Antall møter denne periode 1	Firma - Oppdragsgiver Høgskolen i Ålesund /	Side 1 av 1
Rapport fra prosess Fremdriftsrapport	Periode/uke(r) 24.03 til 15.04.2015	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Thorbjørn Solberg & Birger S. Pedersen	Dato 15.04.2015

Hovedhensikt / fokus for arbeidet i denne perioden

Ferdigstille verktøy, samle resultater, oppstart av rapportskriving

Planlagte aktiviteter i denne perioden

Teste verktøy på realistiske avstander, samle data for verktøyets sanntids- (hastighet og pålitelighet), oppstart av rapportskriving (primært teoridel).

Virkelig gjennomførte aktiviteter i denne perioden

Rapportskrivingen er igang, funnet en løsning for realistisk avstandsbedømmelse. Teoridelen av rapporten er utsatt da gruppen anser det som mer fornuftig å starte med metodikk og resultat.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Programvaren må skrives litt om for å kunne brukes skikkelig til testing. Også noe fravær pga. sykdom.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

(Ingen)

Hovederfaring fra denne perioden

Hovedhensikt/fokus neste periode

Fortsette med rapporten, paralellt med avsluttende utvikling og testing av programvare.

Planlagte aktiviteter neste periode

Implementere ferdig løsning.
Fortsette med metode og resultat i rapporten.
Sanke inn informasjon til teoridelen.

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Tilbakemelding på prosjektets progresjon og gruppens arbeid så langt.

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere

ID301702 Hovedprosjekt	Prosjekt Verktøy for dokking av Uptime gangvei	Antall møter denne periode 1	Firma - Oppdragsgiver Høgskolen i Ålesund /	Side 1 av 1
Rapport fra prosess Fremdriftsrapport	Periode/uke(r) 15.04 til 07.05.2015	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Thorbjørn Solberg & Birger S. Pedersen	Dato 07.05.2015

Hovedhensikt / fokus for arbeidet i denne perioden

Fortsette med rapporten, paralellt med avsluttende utvikling og testing av programvare.

Planlagte aktiviteter i denne perioden

Implementere realistisk avstandsmåling og avslutte programvareutviklingen.
Samle data fra avstandsmåling med bruk av disparity og lasermåler.
Fullføre rapportens metode-kapittel.

Virkelig gjennomførte aktiviteter i denne perioden

Implementert realistisk avstandsmåling.
Samlet data fra avstandsmåling.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Mye av tiden siste periode har blitt brukt til eksamensforbereding.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Opprinnelig plan var å fullføre rapporten ila. uke 19 (før 11. mai). Det avsluttende arbeidet med programvaren har tatt mye lengre tid enn først antatt, så rapportskriving har blitt utsatt. Vi ser oss nødt til å sette strek for utviklingen nå for å rekke å fullføre en god rapport.

Hovederfaring fra denne perioden

Hovedhensikt/fokus neste periode

Fullføre rapporten, holde presentasjon på ICD, holde presentasjon for veileder m.fl.

Planlagte aktiviteter neste periode

Fullføre rapporten i følgende rekkefølge; metodedel, resultater, diskusjon, konlusjon, sammendrag, teoridel og referanser ol.
Planlegge og gjennomføre presentasjoner.

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

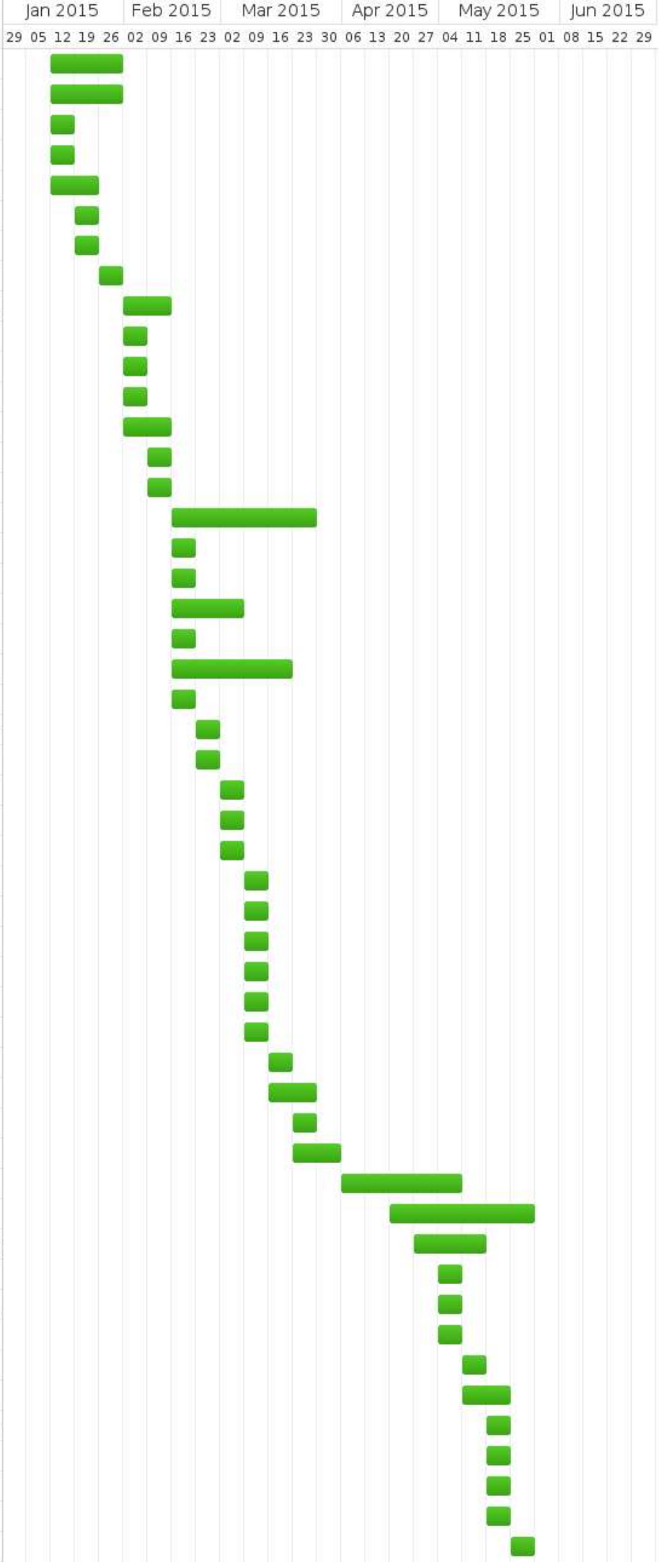
Tilbakemelding på prosjektets progresjon og gruppens arbeid så langt.

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere

Appendix D

Task Overview

Gangway Docking Assista...



1	Complete pre-project and hand in report
2	DEFINE AND FORMULATE ASSIGNMENT, PRE-WORK AND..
3	Initial meeting with company and counsellor
4	Identify the companys preparations
5	Define task
6	Define the project subtasks
7	Develop a plan of the projects progress
8	Acquire necessary basic skills
9	EXAMINE AND DISCUSS ALTERNATIVE SOLUTIONS FOR T..
10	List all solution alternatives
11	Discuss alternatives and select favourable solution(s)
12	Present and discuss alternative solutions with ICD
13	Execute necessary testing of alternatives
14	Decide which solution to prioritize further in the proje..
15	Obtain necessary equipment, acquire necessary info...
16	DEVELOP, TEST AND IMPLEMENT A SOLUTION
17	Plan the solution development
18	Applying 3D-API in Qt
19	Stereo cameras to 3D-map
20	Plan specific subtasks for development
21	Develop and test solution
22	Mapping 3D coordinates from disparity map
23	Calibrate cameras and compute disparity map in real...
24	Develop depth map GUI
25	Acquire depth information from disparity maps
26	Projecting gangway model in 3D
27	Document and illustrate current solution
28	Applying proper filters on disparity maps
29	Tracking objects in 3D
30	Develop algorithm to compensate for sway-motion
31	Complete the disparity maps 3D projecting software
32	Testing disparity maps on realistic distances (~6m)
33	Research 3D monitor/glasses for visualization, order ...
34	Develop software for 3D vision
35	Implement the solution
36	Collect data for result
37	EASTER HOLIDAYS
38	Project break for exam preparations
39	FINISH THE PROJECT WITH DOCUMENTATION
40	Finish documenting methods for the progress of solv...
41	Fasten cameras to a rig
42	Take depth measurements from cameras and finish a..
43	Implement multihreadingn optimize application speed
44	Finish the result chapter
45	Finish the discussion and conclusion chapter
46	Write theory and background information about the e..
47	Create a poster
48	Plan presentation at ICD Software
49	Finish the appendices chapter
50	Finish rest of the report and hand in

Appendix E

Pre-project Report

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

HØGSKOLEN I ÅLESUND

Aalesund University College

TITTEL:
Verktøy for dokking av hiv-kompensert gangvei

GRUPPEMEDLEMMER:
Birger Skogeng Pedersen, Thorbjørn Solberg

DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
	IE303612	**Bacheloroppgave**	- Åpen

STUDIUM:	ANT SIDER/VEDLEGG:	BIBL. NR:
AUTOMASJON	12/5	- Ikke i bruk -

OPPDRAGSGIVER:
ICD SOFTWARE – V/INGOLF SALEN
Veileder:
Ottar L. Osen

OPPGAVE:

ICD og Uptime har utviklet en hiv-kompensert gangvei for bording av skip fra skip eller fra skip til faste installasjoner. Disse gangveiene er styrt av en gangvei-operatør fra gangveiens egen cockpit. Det å styre disse gangveiene er utfordrende, da marginene for en vellykket sammenkobling er små, samt at tuppen på gangveien kan være langt unna.

Prosjektets overordnede målsetting er å utvikle et system eller verktøy som kan bistå føreren i å styre gangveien på en trygg og effektiv måte. Prosessen for å oppnå denne målsetningen skal dokumenteres grundig.

Løsningen må kunne tjene sin hensikt uten å gå på bekostning av sikkerhet eller effektivitet. Verktøyet som blir utviklet må ikke kunne anses som en unødig belastning for produsenten eller føreren av gangveien.

OK. Ottar L. Osen

1. Innholdsfortegnelse

1.1. Innhold

2. Innledning

ICD og Uptime har utviklet en hiv-kompensert gangvei for bording av skip fra skip eller fra skip til faste installasjoner. Disse gangveiene er styrt av en gangvei-operatør fra gangveiens egen cockpit. Det å styre disse gangveiene er utfordrende, da marginene for en vellykket sammenkobling er små, samt at tuppen på gangveien kan være langt unna.

Dette prosjektet har som formål å vurdere mulige løsninger og lage et forslag til løsning eller prototype for verktøy for dokking av gangvei.

Ettersom ICD -og Uptimes gangveier er installerte på en rekke fartøy, samt at flere slike gangveier skal installeres i nær framtid, er det en mulighet for at prosjektgruppens løsning og/eller prototyp kan implementeres i et eksisterende gangvei-system eller et som skal installeres i nær framtid. Dette avhenger av hvorvidt verktøyet blir vellykket ifht målsetningen.

3. Begreper og forkortelser

Begrep eller forkortelse	**Betydning**
Gangvei	Innretning for å transportere personal mellom skip eller mellom skip og offshore-installasjoner, oljeplatformer ol.
Dokking	Prosessen med å styre en gangvei i korrekt posisjon før personell kan borde installasjon eller skip.
Hivkompensering	Teknikk som brukes på løfteutstyr for å redusere påvirkning av bølger under offshore operasjoner
ICD	Industrial Control Design, leverandør av styresystemer til installasjoner i industrien.
Uptime	Firma som leverer gangveier til offshorenæringen.
Asana	Nettbasert prosjektkoordineringsprogram som brukes til å organisere og koordinere arbeidsoppgaver i et prosjekt.

4. Prosjektsorgranisasjon

4.1. Prosjektgruppe

4.1.1. Medlemmer

Rollene som prosjektleder og sekretær vil rullere mellom gruppemedlemmene. Prosjektet starter med Thorbjørn Solberg som prosjektleder, og Birger Skogeng Pedersen som sekretær.

Navn/Studentnummer	Addresse	Mobil	E-post
Birger Skogeng Pedersen 120152	Ystenesgata 33. 6007 ÅLESUND	959 44 571	birger.sp@gmail.com
Thorbjørn Solberg 120147	Verrvgata 8. 6003 ÅLESUND	986 36 114	thorbjornsol@gmail.com

4.1.2. Oppgaver for prosjektgruppen - organisering

Hovedansvar for oppgaver og delmål for prosjektet blir fordelt mellom gruppemedlemmene. Ansvarsområde for de forskjellige aktiviteter som vil foregå i prosjektet er som følger.

Ansvarsområde	Ansvarlig gruppemedlem
Fullføre og levere en forprosjektsrapport	Thorbjørn Solberg
Tildele prosjektgruppen oppgaver	Thorbjørn Solberg
Lage fremdriftsplaner	Thorbjørn Solberg
Arrangere møter	Birger Skogeng Pedersen
Refere og loggføre fra møter	Birger Skogeng Pedersen
Lage liste over alternative oppgaveløsninger	Birger Skogeng Pedersen
Presentere og drøfte alternativer med ICD og veileder	Birger Skogeng Pedersen
Testing og utvikling av alternativer	Thorbjørn Solberg
Kjøpe eller lage nødvendig utstyr for utvikling og testing	Thorbjørn Solberg

Samle inn data om utstyr, til rapportens teori-del	Birger Skogeng Pedersen
Samle inn data fra utvikling og testing	Birger Skogeng Pedersen
Loggføring av utvikling og testing	Birger Skogeng Pedersen
Loggføring av fortløpende drøfting	Thorbjørn Solberg
Fullføre hovedrapporten	Thorbjørn Solberg

4.1.3. Oppgaver for prosjektleder

Prosjektleder har hovedansvaret for fremdriftsplanlegging, overordnede mål (løsninger til oppgaven) og utstyr for utvikling.

4.1.4. Oppgaver for sekretær

Sekretær har hovedansvaret for møter (alt for og etterarbeid), dokumentere arbeid som blir gjort underveis og ha hovedansvaret for rapportering.

4.2. Styringsgruppe

Styringsgruppen består av èn veileder fra høgskolen, og kontaktperson fra ICD.

Stilling	**Navn**	**Mail**	**Mobil**
Veileder	Ottar L. Osen	oo@hials.no	926 61 272
Kontaktperson	Ingolf Salen	is@icd.no	928 68 418

5. Avtaler

5.1. Arbeidssted og ressurser

Prosjektgruppen vil i hovedsak sitte på Høgskolen i Ålesund, men har anledning til å sitte i ICD sine lokaler om det skulle være nødvendig. Møter vil med fordel holdes i ICD sine lokaler.

Kontaktperson i ICD, Ingolf Salen, vil bistå prosjektgruppen med nødvendig informasjon. Han vil også kunne svare på eventuelle spørsmål via epost.

Budsjettet for utstyr til prosjektet er ikke spesifisert. Veileder kan gi tillatelse til å kjøpe utstyr for 3000,- men alt utstyr skal godkjennes og bestilles av prosjektets veileder, Ottar Osen. Mer kostbart utstyr må eventuelt søkes om og begrunnes godt for å få innvilget.

5.2. Gruppernormer - samarbeidsregler - holdninger

Gruppemedlemmene skal være pliktoppfyllende og punktlige mtp oppmøte og arbeidsinnsats.

Gruppemedlemmene skal tillegg behandle hverandre med respekt, opptre saklig og lytte til hverandres forslag og argumenter. Ansvarsfordelingen internt i gruppen er veiledende, og fraskriver ingen av gruppemedlemmene ansvaret for en helhetlig god gjennomførelse av prosjektet.

6. Prosjektbeskrivelse

6.1. Problemstilling - målsetting - hensikt

Prosjektets overordnede målsetting er å utvikle et system eller verktøy som kan implementeres i kontrollsystemet som eksisterer for gangveiene levert av ICD og Uptime. Dette verktøyet skal bistå føreren i å styre gangveien på en trygg og effektiv måte. Prosessen for å oppnå denne målsetningen skal dokumenteres grundig.

6.2. Krav til løsning eller prosjektresultat - spesifikasjon

Løsningen må kunne tjene sin hensikt uten å gå på bekostning av gangvei-dokkingens sikkerhet eller effektivitet. Verktøyet som blir utviklet må ikke kunne anses som en unødig belastning for produsenten eller føreren av gangveien. Kostnad for verktøyet må være såpass lav at verktøyet blir benyttet.

6.3. Planlagt framgangsmåte for utviklingsarbeidet – metode

Ved informasjonsinnsamling samt testing og drøfting av forskjellige løsninger og produkter for problemstillingen skal prosjektgruppen kunne bestemme seg for èn eller flere løsning(er) som skal prioriteres. Etter dette skal denne/disse løsning(ene) videreutvikles.

6.4. Informasjonsinnsamling - utført og planlagt

Ved dialog med utviklere, installatører og brukere av det eksisterende gangvei-systemet, samt undersøkelse og forskning skal prosjektgruppen kunne få et grundig innblikk i hvordan oppgaven kan løses. prosjektgruppen er også interessert i å besøke en båt eller installasjon som har og bruker en av disse gangveiene.

6.5. Vurdering - analyse av risiko

Etter prosjektgruppens vurdering skal en løsning for oppgaven kunne utbedres og utvikles. Om løsningen blir realisert og installert på en av ICD eller Uptimes gangvei er vanskelig å fastslå før det er større klarhet i hvilken løsning som blir prioritert. prosjektgruppen vil velge å prioritere en mest mulig gjennomført og god løsning foran noe som tar kortere tid å utvikle for å installere hurtigere.

For å lykkes med å fullføre oppgaven på en best mulig måte, må gruppemedlemmene ha høy innsats, planlegge prosjektet godt og ha god dialog med bedriften(e) og veilederen.

6.6. Hovedaktiviteter i videre arbeid

Nr	**Aktivitet**	**Ansvarlig**	**Tid/omfang (dager)**
A1	Definere og spissformulere oppgave, forarbeid og oppstart av prosjektet	T.S.	13
A11	Møte bedriften og veileder	B.S.P.	1
A12	Definere oppgave; spissformulere prosjektets hovedmål i tråd med bedriftens ønsker	B.S.P.	3
A13	Kartlegge bedriftens eventuelle arbeid med problemløsningen, samt bedriftens egne ideer og visjoner til eventuelle løsninger	T.S.	2
A14	Utvikle en plan for prosjektets fremdrift	T.S.	2
A15	Definere delmål	B.S.	1
A15	Tilegne prosjektgruppen nødvendig grunnkompetanse for å kunne arbeide med bedriftens systemer og verktøy	T.S.	5
A2	Gjennomgå og drøft alternative løsninger for oppgaven	B.S.P.	10

A21	Liste opp alle alternativer for løsninger, gjør nødvendig forskning (samle inn data) og eventuelt forkaste ugunstige ideer	B.S.P.	1
A22	Drøfte alternativer og velge ut (tilsynelatende) mest gunstige løsninger	B.S.P.	3
A23	Presentere og drøfte alle alternative løsninger med bedriften	B.S.P.	1
A24	Utføre nødvendig testing av alternativer til oppgaveløsning	T.S.	3
A25	Bestemme hvilket alternativ som skal prioriteres videre i prosjektet.	T.S.	3
A26	Skaffe nødvendig utstyr, samt tilegne prosjektgruppen nødvendig informasjon/kunnskap	T.S.	3
A3	Utvikle, teste og implementere en løsning	T.S.	30
A31	Spesifisere løsningen	B.S.P.	2
A32	Planlegge spesifikke delmål for utviklingen	B.S.P.	3
A33	Utvikle, teste og eventuelt implementere løsningen	B.S.P.	22
A34	Samle inn data for resultat	T.S.	5
A4	Fullføre prosjektet med rapporten	B.S.P.	25
A41	Skrive teori og bakgrunnsinformasjon om utstyret	B.S.P.	10
A42	Føre inn metodikk for framgang av oppgaveløsning	B.S.P.	10
A43	Dokumentere bakgrunn for forkasting av “ugunstige” løsninger	B.S.P.	10
A44	Drøfte data og resultat	T.S.	10
A45	Fullføre en helhetlig rapport	T.S.	10

6.7. Framdriftsplan - styring av prosjektet

6.7.1. Hovedplan

Nr	Aktivitet	Start	Slutt
A1	Definere og spissformulere oppgave, forarbeid og oppstart	6. Jan.	30. Jan.
A2	Gjennomgå og drøfte alternative løsninger for oppgaven	2. Feb.	13. Feb.
A3	Utvikle, teste og implementere en løsning	16. Feb.	27. Feb.
A4	Fullføre prosjektet med dokumentasjon	6. Apr.	8. Mai.

6.7.2. Detaljplan

Definere og spissformulere oppgave, forarbeid og oppstart.

Nr	Aktivitet	Start	Slutt
A11	Møte bedriften og veileder	15. Jan.	15. Jan.
A12	Definere oppgave; spissformulere prosjektets hovedmål i tråd med bedriftens ønsker	16. Jan.	20. Jan.
A13	Kartlegge bedriftens eventuelle arbeid med problemløsningen, samt bedriftens egne ideer og visjoner til eventuelle løsninger	15. Jan.	16. Jan.
A14	Utvikle en plan for prosjektets fremdrift	22. Jan.	23. Jan.
A15	Definere delmål	21. Jan.	21. Jan.
A15	Tilegne prosjektgruppen nødvendig grunnkompetanse for å kunne arbeide med bedriftens systemer og verktøy	26. Jan.	30. Jan.

Gjennomgå og drøfte alternative løsninger for oppgaven

Nr	Aktivitet	Start	Slutt
A21	Liste opp alle alternativer for løsninger, gjør nødvendig forskning (samle inn data)	2. Feb.	2. Feb.

	og eventuelt forkaste ugunstige ideer		
A22	Drøfte alternativer og velge ut (tilsynelatende) mest gunstige løsninger	2. Feb.	4. Feb.
A23	Presentere og drøfte alle alternative løsninger med bedriften	5. Feb.	5. Feb.
A24	Utføre nødvendig testing av alternativer til oppgaveløsning	5. Feb.	9. Feb.
A25	Bestemme hvilket alternativ som skal prioriteres videre i prosjektet.	9. Feb.	11. Feb.
A26	Skaffe nødvendig utstyr, samt tilegne prosjektgruppen nødvendig informasjon/kunnskap	11. Feb.	13. Feb.

Utvikle, teste og implementere en løsning

Nr	**Aktivitet**	**Start**	**Slutt**
A3 1	Spesifisere løsningen	16. Feb.	17. Feb.
A3 2	Planlegge spesifikke delmål for utviklingen	18. Feb.	18. Feb.
A3 3	Utvikle, teste og eventuelt implementere løsningen	19. Feb.	20. Mars
A3 4	Samle inn data for resultat	23. Mars	27. Mars

Fullføre prosjektet med dokumentasjon

Nr	**Aktivitet**	**Start**	**Slutt**
A4 1	Skrive teori og bakgrunnsinformasjon om utstyret	6. Apr.	17. Apr.
A4 2	Føre inn metodikk for framgang av oppgaveløsning	13. Apr.	24. Apr.
A4 3	Dokumentere bakgrunn for forkasting av “ugunstige” løsninger	20. Apr.	1. Mai

A44	Drøfte data og resultat	20. Apr.	1. Mai
A45	Fullføre en helhetlig rapport	27. Apr.	8. Mai

6.7.3. Styringshjelpemidler

Prosjektgruppen bruker et gratis web-basert verktøy som heter Asanaa sammen en plug-in som heter Instagantt til å organisere arbeidet, tilordne arbeidsoppgaver, og å lage gantt-diagram. Dokumentering og rapportering foregår i Google Disc.

6.7.4. Utviklingshjelpemidler

- ProVisual Studio
- Qt Creator
- OpenCV
- MATLAB
- Eclipse

6.7.5. Intern kontroll - evaluering

Alle oppgaver og deloppgaver blir opprettet i Asana og fordelt mellom gruppemedlemmene. Etterhvert som oppgavene blir utført blir de kvittert ut, og gruppemedlemmene har med dette verktøyet en tydelig oversikt over hvilke oppgaver som gjenstår til hvilken tid.

6.8. Beslutninger – beslutningsprosess

Medlemmene i prosjektgruppen har diskutert og blitt enige før beslutninger har blitt tatt underveis i arbeidet med forprosjektet.
Prosjektgruppen kommer til å bruke samme metode i hovedprosjektet, men ved store beslutninger, eller ved uenigheter vil gruppen rådføre seg med veiledere.

7. Dokumentasjon

7.1. Rapporter og tekniske dokumenter

Gruppen skal før prosjektets slutt ha produsert en hovedrapport med tillegg som inneholder en detaljert beskrivelse av metodikken underveis i prosjektet. Rapporten skal også ha vedlagt datablader, priser og kontaktinfo for leverandører for produkter som brukes til en løsning, samt kildekode for programvare prosjektgruppen utvikler ifm. prosjektet.
ICD har ikke spesifikke krav til dokumentasjon i prosjektet utover hva som allerede er nevnt.
Hovedrapporten blir skrevet på engelsk, selv om noen av vedleggene i hovedrapporten (spesielt møteinnkallelser og møtereferat) vil være skrevet på norsk.

8. Planlagte Møter og Rapporter

8.1. Møter

8.1.1. Møter med styringsgruppen

Møter kalles inn ved behov, normalt innen hver 14. dag. Kontaktpersonen i ICD har begrenset med tid til prosjektet, prosjektgruppen må kunne jobbe selvstendig og unngå å kalle inn til unødvendige møter.

8.1.2. Prosjektmøter

Prosjektgruppen møtes normalt hver dag og jobber side om side med prosjektets arbeidsoppgaver. Formelle møter internt i prosjektgruppen vil bare bli holdt ved spesielle anledninger, eksempelvis ved større interne beslutninger.

8.2. Periodiske rapporter

8.2.1. Framdriftsapporter

Framdriftsrapport vil bli levert til veileder minimum hver 14. dag. Den skal minimum bestå av en utfylt rapport utfra framdriftsapport-malen samt gantt-diagram som illustrerer hvilke oppgaver som er fullført, underveis og gjenstående.

9. Planlagt Avviksbehandling

Dersom prosjektgruppen ikke klarer å holde frister for noen av oppgavenes sentrale oppgaver, eller prosjektgruppen innser grove feilvurderinger som kan sette prosjektets suksess i fare, vil prosjektgruppen kalle inn veileder for råd. Alle mål og/eller forutsetninger som ikke blir gjennomført eller vellykket vil bli grundig drøftet og dokumentert i hovedrapporten.

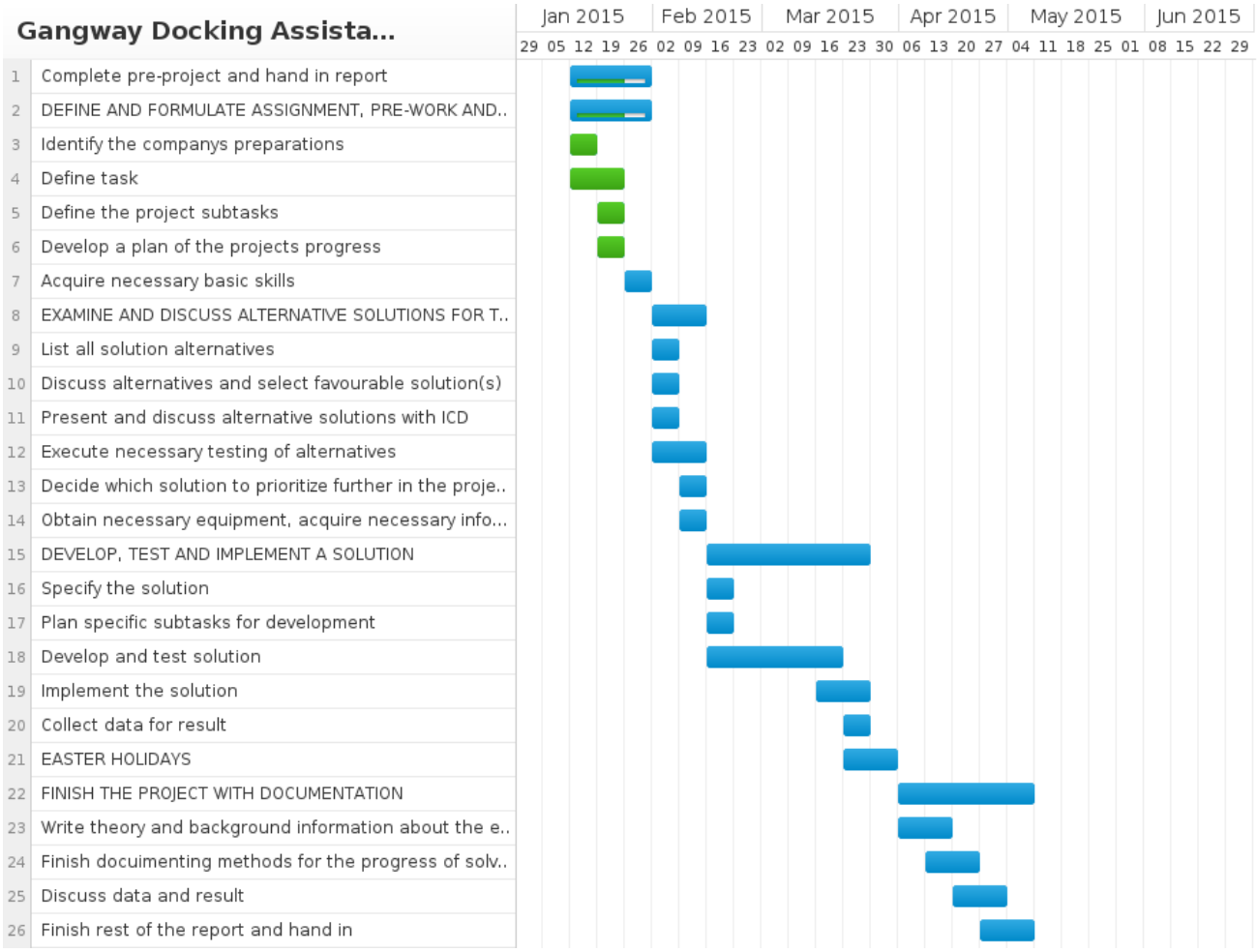
10. Utstyrsbehov/Forutsetninger for Gjennomføring

Utstyrsbehov avhenger av hvilken løsning på problemstillingen prosjektgruppen velger å gå videre med. Som utgangspunkt vil et kamera være nødvendig. Til organisering av prosjektet brukes Asana, Instagantt og Google Docs samt diverse programvare-utviklerverktøy. Alle disse produktene er gratis til ikke-kommersielt bruk.

11. Vedlegg

- Vedlegg 1: Gantt-diagram
- Vedlegg 2: Saksliste fra møte 15.01.2015
- Vedlegg 3: Møterefferat fra møte 15.01.2015
- Vedlegg 4: Saksliste fra møte 27.01.2015
- Vedlegg 5: Møterefferat fra møte 27.01.2015

Vedlegg 1


Gangway Docking Assista...
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
29 05 12 19 26 02 09 16 23 02 09 16 23 30 06 13 20 27 04 11 18 25 01 08 15 22 29
1 Complete pre-project and hand in report
2 DEFINE AND FORMULATE ASSIGNMENT, PRE-WORK AND..
3 Identify the companys preparations
4 Define task
5 Define the project subtasks
6 Develop a plan of the projects progress
7 Acquire necessary basic skills
8 EXAMINE AND DISCUSS ALTERNATIVE SOLUTIONS FOR T..
9 List all solution alternatives
10 Discuss alternatives and select favourable solution(s)
11 Present and discuss alternative solutions with ICD
12 Execute necessary testing of alternatives
13 Decide which solution to prioritize further in the proje..
14 Obtain necessary equipment, acquire necessary info...
15 DEVELOP, TEST AND IMPLEMENT A SOLUTION
16 Specify the solution
17 Plan specific subtasks for development
18 Develop and test solution
19 Implement the solution
20 Collect data for result
21 EASTER HOLIDAYS
22 FINISH THE PROJECT WITH DOCUMENTATION
23 Write theory and background information about the e..
24 Finish docuimenting methods for the progress of solv..
25 Discuss data and result
26 Finish rest of the report and hand in

Vedlegg 2

For gruppen

- Definere oppgaven
    - Gjennomgå problemstillingen
    - Sette rammer for oppgaven
    - Målsetninger (hovedmål og evt. delmål)
    - Krav til resultat - Spesifikasjon
    - Muligheten for å montere noe på landing?
- Tidligere arbeid/idèer/planer?
    - Montert kamera på toppen? (senere med ingolf)
- Styring av prosjekt
    - Fremdriftsplan
    - Rapportering
- Tilgang til ressurser
    - Arbeidsplass, (evt. på ICD?)
    - Personer (Ingolf)
    - Utstyr (Billigst mulig (helst under 3000))
    - Budsjett (alt går gjennom Ottar)
- Planlagte gangvei-installasjoner før sommeren? spør ingol
- Datasikkerhet/informasjon unndratt offentlighet (åpent ottar kommer med tilbakemelding)
- Våre første idèer til eventuelle løsninger
    - avstandsmåling (med avstandsmålere), illustrere posisjonen til landingen ifht gangveien på en gui
    - montere kamera under tuppen av gangveien, vise videofeed med beregnet landingsposisjon inntegnet i skjermbildet

Spørsmål

- Definere oppgaven
    - Hvilke forventninger/forhåpninger har dere til resultatet
        - forventer dere et ferdig produkt som kan taes i bruk allerede til sommeren? (nei)
- hvor ofte skal vi ha møte med veiledere?

Vedlegg 3

Styringsmøte for Verktøy For Dokking Av Gangvei

Avholdt 15.01.2015 kl 11:30 på ICD Software, Hundsværgata 8, 6008 ÅLESUND.

Tilstede på møtet: Thorbjørn Solberg, Birger Skogeng Pedersen og Ottar Osen (herunder betegnet som veileder).
Fraværende: Ingolf Salen (herunder betegnet som ICD kontakperson).

Besvarelse av sakslisten

(Saker ihht innkalling er listet opp som kulepunkter, gjennomgang av sakene følger hvert punkt).

- Definere oppgaven
    - Gjennomgå problemstillingen
    - Sette rammer for oppgaven
    - Målsetninger (hovedmål og evt. delmål)
    - Krav til resultat - Spesifikasjon
    - Muligheten for å montere noe på landing?

- Problemstillingen går ut på å forenkle dokking av ICD/Uptime gangbru. Gruppen står i utgangspunktet fritt til å velge løsning, men bør teste ut forskjellige alternativer. Her ble nevnt:
    - sanntidsdata fra videokamera eller annen avstandsmåling (eksempelvis med laser) som sammenlignes med 3D-modell av landing eller oppgitt størrelse på objekt på landing,
    - bedre visualisering for gangveiens fører (mulighet å bruke 3D-visualisering av landing eller inntegning i av landingspoisjon for gangvei i videobilde) og
    - automatisk dokking.
- Viktig å få med i rapporten allle de alternativene gruppen IKKE velger og dokumentere hvorfor disse er forkastet eller nedprioriterte.
- Vurder alternativene og kom på flere. Test alternativer og vurder hvilke som kanskje er best egnet.

- Tidligere arbeid/idèer/planer?
    - Montert kamera på toppen?
- (Ble ikke besvart, dette må gruppen ta med Ingolf ved neste møte).

- Styring av prosjekt
    - Fremdriftsplan
    - Rapportering

- Fremdriftsplan lages om mulig i Asana, og veileder ønsker tilbakemelding på hvorvidt Asana egner seg.
- Rapportering skjer via dokument på Fronter, og møte med veileder hver 14. dag.

- Tilgang til ressurser
    - Arbeidsplass
    - Personer
    - Utstyr
    - Budsjett
- Arbeidsplass blir primært på skolen, men arbeidsgruppen kan ha dager på ICD etterhvert dersom det er ønskelig. Dersom gruppen ønsker å reservere et kontor på ICD, må dette klareres med Ottar Osen som kan ta det med daglig leder.
- Kontaktperson på ICD blir primært Ingolf Salen.
- Det eksisterer foreløpig lite/ingenting utstyr med direkte relevans til oppgaveløsning på ICD som gruppen kan bruke, bortsett fra et kamera. Budsjett for utstyr er foreløpig uspesifisert, men all bestilling av utstyr skal gå gjennom veileder. Veileder ønsker at gruppen unngår utstyr dyrere enn 3000 kroner, men er åpen for dyrere utstyr dersom det kan begrunnes godt.

- Planlagte gangvei-installasjoner før sommeren?
- (Ble ikke besvart, må snakke med kontaktperson på ICD om dette).

- Datasikkerhet/informasjon unndratt offentlighet
- Ikke avklart, veileder kommer tilbake med mer info om dette punktet senere. Til da holder gruppen alle spesifikke detaljer om oppgaven skjult for offentligheten.
- Våre (gruppens) første idèer til eventuelle løsninger
    - avstandsmåling (med avstandsmålere), illustrere posisjonen til landingen ifht gangveien på en gui
    - montere kamera under tuppen av gangveien, vise videofeed med beregnet landingsposisjon inntegnet i skjermbildet
- Idèer ble diskutert innledningsvis, se foranliggende punkt.

Annet

- Ivar Blindheim var tidligere også satt opp som veileder, men er trukket grunnet for mange grupper.
- Tips fra Ottar; Drøft og dokumenter alternativene veldig godt. Alle forkastede og foretrukne idèer skal begrunnes grundig. Bruk heller tiden på rapportskriving og drøfting enn utvikling av selve prosjektet, da rapporten er den som teller aller mest.
- Ikke la ICDs prioriteringer ta helt over oppgaven, gruppen må prøve å prioritere bacheloroppgaven og tenke (mest mulig) på seg selv.
- Veileder sa at vi burde begynne med planlegging og lage framdriftsplaner så snart som mulig. Dette skal framlegges før neste møte (dvs. innen to uker).

- Etterhvert kan gruppen undersøke og få navn på båter med installert 23,4m gangbro, finne de via AIS track på nett og avtale eventuelt besøk.
- Veileder anbefaler engelsk hovedrapport. Samtidig er det litt mer krevende å skrive rapporten på engelsk, så dette blir en vurderingssak for gruppen.
- Neste styremøte blir senest innen to uker (Dvs. innen torsdag 29-01-2015).

Møtereferent: Birger Skogeng Pedersen.

Vedlegg 4

Hei,

Kaller herved inn til møte ang. bacheloroppgave; verktøy for dokking av gangvei. Flere ting på sakslisten ble gjennomgått forrige møte, men vi ønsker å ta det kort opp igjen for å få innspill fra Ingolf. Fint om Ingolf også kan kommentere forslagene til en løsning fra forrige møte.

Saksliste

1. Definere oppgaven
    1.1. Sette rammer for oppgaven
    1.2. Målsetninger (hovedmål og evt. delmål)
    1.3. Krav til resultat - Spesifikasjon
    1.4. Muligheten for å montere noe på landing
2. Tidligere arbeid/idèer/planer
    2.1. Montert kamera på toppen/bunnen?
3. Tilgang til ressurser
    3.1. Personer
    3.2. Utstyr
    3.3. Budsjett
    3.4. Hyppighet mellom møter med ICD
    3.5. Rapportering til ICD?
    3.6. Navn på båter med Uptime gangveier?
4. Planlagte gangvei-installasjoner før sommeren
5. Datasikkerhet/informasjon unndratt offentlighet
6. Våre første idèer til eventuelle løsninger
    6.1. Kommentarer fra Ingolf på forslag til løsninger fra forrige møte
7. Foreløpig status og fremdriftsplan (ligger som vedlegg)
    7.1. Tilbakemelding på tidsbruk
8. Dokumentason
    8.1. Rapporter og tekniske dokumenter
    8.2. Hva slags dokumentasjon skal utarbeides – utforming, innhold
9. Eventuelt

Forslag til løsninger (fra forrige møte, 2015-01-15)

- sanntidsdata fra videokamera eller annen avstandsmåling (eksempelvis med laser) som sammenlignes med 3D-modell av landing eller oppgitt størrelse på objekt på landing,
- bedre visualisering for gangveiens fører (mulighet å bruke 3D-visualisering av landing eller inntegning i av landingspoisjon for gangvei i videobilde) og

- automatisk dokking.

Vedlegg 5

Styringsmøte for Verktøy For Dokking Av Gangvei

Avholdt 27.01.2015 kl 10:30 på ICD Software, Hundsværgata 8, 6008 ÅLESUND.

Tilstede på møtet: Thorbjørn Solberg, Birger Skogeng Pedersen og Ottar Osen (herunder betegnet som veileder), Ingolf Salen (herunder betegnet som ICD kontakperson).

Besvarelse av sakslisten

(Saker ihht innkalling er listet opp som kulepunkter, gjennomgang av sakene følger hvert punkt).

- Definere oppgaven
    - Sette rammer for oppgaven
    - Målsetninger (hovedmål og evt. delmål)
    - Krav til resultat - Spesifikasjon
    - Muligheten for å montere noe på landing
- Det ble foreslått at oppgaven skulle deles opp for å adskille de to dokkingmetodene.
    - De to scenarioene bør defineres hver for seg.
    - Gruppen må velge ett av scenarioene, finne en løsning, og derretter sammenligne og se hva som kan brukes fra det første alternativet til det andre. (for videre arbeid)
- Det ble diskutert om det kunne males på en glyph eller lignende på landingen.
    - Det ble konkludert med at dette kan kan bli veldig dyrt, men at det bør nevnes i rapporten. (Skriv hvorfor det ikke kan brukes i rapporten)
- Tidligere arbeid/idèer/planer
    - Montert kamera på toppen/bunnen?
- Gangbroene leveres med to kameraer som standard. Ett kamera er montert på oversiden, og ett er montert på undersiden litt lengre inn.
- Tilgang til ressurser
    - Personer
    - Utstyr
    - Budsjett
    - Hyppighet mellom møter med ICD
    - Rapportering til ICD?
    - Navn på båter med Uptime gangveier?

- Det ble nevnt at om Ingolf skulle være utilgjengelig, så kunne Arnet kontaktes for spørsmål.
- Ingolf vil delta på møter annehver uke om ikke annet avklares på forhånd.

- Rapportering direkte til ICD kun hvis det er noe spesielt. (Hvis det kommmer noen kunder som vil ha en slik løsning)
- Island Crown og Siem Moxie, samt en på land i Haugesund.

- Planlagte gangvei-installasjoner før sommeren
- ICD skal teste en gangbro i Haugesund i Februar. Dersom gruppen vil montere noe eller teste noe på en gangvei, må dette diskuteres med Uptime.
- Datasikkerhet/informasjon unndratt offentlighet
- Gruppemedlemene må signere en NDA (Non-disclosure agreement) hos ICD.
- Våre første idèer til eventuelle løsninger
    - Kommentarer fra Ingolf på forslag til løsninger fra forrige møte
- Visualisering med 3D-briller er trolig ingen optimal løsning for å installere på gangveiene, men for prosjektet sin del (bacheloroppgaven) er 3D-visualisering innen industrien et relevant område som bør forskes på.
- Foreløpig status og fremdriftsplan (ligger som vedlegg)
    - Tilbakemelding på tidsbruk
- Veileder: Fremdriftsplanen virker til å være et fornuftig utgangspunkt for prosjektet. Fremdriftsplanen må være dynamisk, flere og flere delmål må komme på plass underveis.
- Dokumentason
    - Rapporter og tekniske dokumenter
    - Hva slags dokumentasjon skal utarbeides – utforming, innhold
- I all hovedsak rapporten, datablader og kildeode, priser og leverandører. ICD har ingen egne, spesifikke krav til dokumentasjon.
- Eventuelt
- Tips fra Ingolf; Gruppen bør sikte seg inn på “I-bjelke”-løsningen
- For å implementere en løsning i ICDs allerede utviklede styresystemer må gruppen lage “CDP-komponenter” som kan brukes i softwaren. Gruppen skal mao. ikke forandre noe i kjerner på styresystemet, men lage en modulær løsning.
- Det ble diskutert flere forslag til løsninger, og nevnt noen utfordinger.
    - Ta bilde, måle på bildet for å finne avstanden.
    - Korrigering med kamera
    - MPC, for å forrutsi bevegelsen til landing
    - Uptime krever at alt utstyr som skal monteres på gangveiene skal være ex-godkjent
    - Problem med vannsprut
    - Lage en meny hvor opperatøren kan velge mellom hvilke input/sensorteknologier som skal være i bruk mtp utfordringer som kan sette noen alternativer ut av spill.
- Det ble foreslått at gruppen i rapporten skriver litt om hvordan noe av det som utvikles kan bli brukt videre til “Ship to ship” mtp MRYU-data fra en annen båt.

Møtereferent: Thorbjørn Solberg