NTNU

Norwegian University of
Science and Technology

Practical Experiments on the Efficiency of the Remote Presence

Remote Inspection in an Offshore Wind Turbine

Jeremias Moragues Pons

Master of Science in Engineering Cybernetics
Submission date: February 2012
Supervisor: Amund Skavhaug, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

“A great wind is blowing, and that gives you either imagination or a headache.”

Catherine the Great

Empress and Autocrat of All the Russias, 1729-1796

MSc Thesis Assignment

Assignment tittle: Practical Experiments on the Efficiency of the Remote Presence.

Remote Inspection on an Offshore Wind Turbine

Background:

- The inspections on the Offshore Wind Turbines are expensive although necessary.
- Given the need for carrying out these Inspections with a smaller budget, a solution would be to make them by Remote Presence.
- Is this possible and feasible?

Project Targets:

- Study previous work.
- Study necessary background theory.
- Identify and discuss different possible solutions for physical layout of instruments and signals to use.
- Control system for experiments.
- Suggest the user interface for remote presence.
- Select the interface.
- Plan the implementation and Implement as far as time allows.
- Evaluate how good the system with small experiments is.

Assignment given: 12. September. 2011

Submission date: 4. February. 2012

Jeremias Moragues Pons

Trondheim, 5. October. 2011

Abstract

Offshore wind power has become a growing interest in the worldwide society. New research and investigation in this kind of technology is increasing year after year. Thus, the inspection on the offshore wind turbines is expensive but necessary.

The goal of this project is to demonstrate the efficiency of a Remote Inspection System inside the nacelle of an Offshore Wind Turbine. The way of carrying out inspections on real nacelles, as well as other aspects from maintenance is discussed. Three different companies have been taken into account when developing an Inspection Plan.

This master thesis contains a theoretical design of the input device that would be used in the future for inspection and maintenance tasks. A gamepad has been implemented as a temporary solution for the input device.

It has been demonstrated how the prototype is able to detect failures in a scenario simulating a real nacelle. Although it has not been possible to test sound, heat or vibration, this kind of things would be easily detected by adding some more additional sensors.

The conclusion is that a solution for reducing maintenance tasks relies on the development of effective condition monitoring and remote control systems.

Preface

This master thesis has given me knowledge in many fields that I did not know in depth, such as: offshore wind power, java programing, and basics from ATMEL microcontroller.

I also have learnt better how to work by my own and to investigate through different tasks at the same time.

This project has made me improve my writing and understanding English skills, which was one of the reasons for coming to Norway.

I would like to thank my supervisor for the motivation transmitted and for the freedom I have had in this project. This has allowed me to raise my own goals.

I would also like to thank Øyvind Netland, PhD on the Department of Engineering Cybernetics, his support and advice in making the most complex decisions during these months.

I must thank the help from employees on the department workshop, which have provided the materials for doing the “experiments”. In this way, thanks also to Genge & Thoma Company. They have been in touch with me giving advice on the possible input devices to use.

Thanks to my family and friends, especially my sister María, my cousin Elisa and my aunt Jane, who have helped me with the redaction and correction of the thesis.

Thanks to my girlfriend Bea, who has been next to me in the most difficult moments.

And finally, thanks mom and dad for allowing me to undertake this way and helping me whenever I needed.

Jeremias Moragues Pons

Trondheim, 1. February. 2012

Table of Contents

1 INTRODUCTION 1
1.1 WIND ENERGY 1
1.1.1 Offshore Wind Power 2
1.2 WIND GENERATION IN EUROPE 3
1.2.1 Wind in Norway 4
1.3 OFFSHORE INSPECTION AND REMOTE PRESENCE 6
1.4 THE OBJECTIVE: HOW TO STUDY THE EFFICIENCY 6
1.4.1 Contribution 6

PART I. Development of the Thesis

2 PREVIOUS WORK 9
2.1 BACKGROUND 9
2.1.1 Motivation 9
2.1.2 Made Design 10
2.2 PROTOTYPES 11
2.2.1 First Option 11
2.2.2 Second Option 12
2.3 THE CHOSEN ROBOT 12
2.4 USER INTERFACE 13

3 NEW DEVICE: JOYSTICK 15
3.1 CONTRIBUTION 15
3.1.1 Technical details 16
3.1.2 Industrial applications 16
3.1.3 Demanded features 17
3.2 LIST OF POSSIBLE DEVICES 18
3.2.1 Final device 18
3.3 ASPECTS TO CONSIDER IN MAKING THE INTERFACE 19
3.3.1 Delay 19
3.3.2 Collision 19
3.3.3 Vibration 19
3.3.4 Placing & Augmented reality 19
3.4 FINAL IMPLEMENTATION 22
3.4.1 Device used 22
3.4.2 Java Programing 23
3.4.2.1 JX Input – Input Devices for Java 23
3.4.2.2 Client-Inspection 25

4 INSIDE THE NACELLE 27
4.1 ANATOMY OF A WIND TURBINE 27
4.1.1 Blades *28*
4.1.2 Nacelle *28*
4.1.3 Tower *28*
4.2 MAIN PARTS OF THE NACELLE 29
4.2.1 Rotor hub *29*
4.2.1.1 Main Shaft 29
4.2.1.2 Small Shaft 29
4.2.2 Gearbox *30*
4.2.3 Generator *30*
4.2.3.1 Cooling System 30
4.2.4 Mechanical brake *30*
4.2.5 Yaw Drive *30*
4.2.5.1 Yaw motor 31
4.2.5.2 Yaw bearing 31
4.2.6 Anemometer and Wind Vane *31*
4.2.7 Controller *32*

5 OPERATION AND MAINTENANCE 33
5.1 INTRODUCTION 33
5.1.1 Land Based Comparative Data *34*
5.2 MAINTENANCE STRATEGIES 34
5.3 O&M OFFSHORE EXPERIENCE 35
5.3.1 Availability *35*
5.3.2 Operational expenditure *35*
5.3.3 Serviceability *35*
5.3.4 Access for maintenance *36*
5.4 DESIGNS FOR REDUCED MAINTENANCE 36
5.4.1 Component reliability *36*
5.4.1.1 Gearbox 37
5.4.1.2 Generator 37
5.4.1.3 Direct Drive System 37
5.4.1.4 Electrical and Electronic Control System 37
5.4.1.5 Hydraulic System 37
5.4.2 Corrosion protection *38*
5.4.3 Control and condition monitoring *38*
5.4.4 Back-up power *38*
5.5 O&M CONCLUSIONS 39

6 WHAT TO INSPECT 41
6.1 GENERAL OUTLINES ABOUT THE INSPECTION 41
6.2 INSPECTING THE NACELLE 42
6.2.1 Structural System *42*
6.2.2 Electrical and Mechanical System *43*
6.3 COMPANIES GENERAL PROCEDURE 43

7 DIFFERENT POSSIBLE SOLUTIONS 45
7.1 INSPECTION PLAN 45
7.1.1 The Component List *46*
7.2 LAYOUT 46
7.2.1 Lighting conditions *47*
7.2.2 Analysed scenarios *48*
7.3 EXPERIMENTAL CONTROL SYSTEM 48

8 TESTS 49
8.1 INTRODUCTION 49
8.1.1 Modus operandi *49*
8.2 TEST 1. GENERAL SEARCH 50
8.2.1 Visible cables *50*
8.2.2 Oil leakage *52*
8.3 TEST 2. COMPARISON SEARCH 53
8.3.1 Control panel *53*
8.3.2 Motor connection *54*
8.3.3 Bolts connections *54*

9 DISCUSSION 55
9.1 THE INSPECTION CONTROLLER 55
9.2 AUGMENTED REALITY 56
9.3 INSPECTION TESTS 56
9.4 FUTURE WORK 57

10 CONCLUSION 59

PART II. Appendix

A. MAP OF WIND FARMS IN SCANDINAVIA 63
B. JOYSTICKS FROM GENGE & THOMA 65
C. JXINPUT – INPUT DEVICES FOR JAVA 67
D. CLIENT – INSPECTION APPLICATION 83
E. INSPECTION SUMMARY 95
F. INSPECTION CHECKLIST 97
G. THE DVD 99
REFERENCES 101

List of Figures

Figure 1. Hywind in Åmøy Fjord, the first full scale Offshore Floating Windmill 2
Figure 2. Average wind farm size in MW 3
Figure 3. The whole system with the track 10
Figure 4. First Inspection Robot 11
Figure 5. Last Inspection Prototype 12
Figure 6. Client Remote Inspection Interface 13
Figure 7. Example of Industrial Joysticks 16
Figure 8. Put'n Stay Joystick GT 17
Figure 9. Marine Controller BS 130 GT 17
Figure 10. Control Panel for Inspection 18
Figure 11. Augmented Reality System for 3D mapping 20
Figure 12. Augmented Reality System for a moving vehicle 21
Figure 13. Game pad used as the new Controller Device 22
Figure 14. Control Scheme Joypads 22
Figure 15. Architecture of the JXInput package 23
Figure 16. Joystick Test Interface 24
Figure 17. Top values for detection in the Java application 24
Figure 18. Screen Capture of Eclipse (Java classes to update) 25
Figure 19. Example of an Offshore Wind Turbine 28
Figure 20. Main parts of a Nacelle 29
Figure 21. Typical arrangement of yaw bearing and yaw drive 31
Figure 22. Anemometer and wind vane 31
Figure 23. Diagram of an Advantech APAX Control System 32
Figure 24. Taiwan Power Company logo 43
Figure 25. Energo Engineering logo 44
Figure 26. Statkraft logo 44
Figure 27. Components and items to use during the inspection 45
Figure 28. Remote inspection device inside the nacelle of wind turbine 46
Figure 29. Updated path for inspection testing 47
Figure 30. Nacelle lighting and power systems 47
Figure 31. Wear and tear on cables remotely inspected 50
Figure 32. Wear and tear on inspected cables (being there) 51
Figure 33. Oil leakage 52
Figure 34. Control Panel 53
Figure 35. Motor connection 54
Figure 36. Bolts connections and grease stains 54

List of Tables

TABLE 1. WIND POWER PLANTS IN NORWAY AT THE END OF 2011 4
TABLE 2. STATUS OF THE OFFSHORE WIND TURBINES IN JANUARY 2011 5
TABLE 3. RECOMMENDED INSPECTION CYCLES (YEARS) 41

List of Source Code

SOURCE CODE 1. THE JOYSTICKLISTENER CLASS 67
SOURCE CODE 2. THE JOYSTICKNOTIFIER CLASS 70
SOURCE CODE 3. THE JOYSTICK CLASS 78
SOURCE CODE 4. THE JOYSTICKTEST CLASS 82
SOURCE CODE 5. THE CAMCONTROLLER CLASS 87
SOURCE CODE 6. THE VEHICLECONTROLLER CLASS 90
SOURCE CODE 7. THE CAMMODEL CLASS 92
SOURCE CODE 8. THE VEHICLEMODEL CLASS 93

1

Introduction

1.1 WIND ENERGY 1
1.1.1 OFFSHORE WIND POWER 2
1.2 WIND GENERATION IN EUROPE 3
1.2.1 WIND IN NORWAY 4
1.3 OFFSHORE INSPECTION AND REMOTE PRESENCE 6
1.4 THE OBJECTIVE: HOW TO STUDY THE EFFICIENCY 6
1.4.1 CONTRIBUTION 6

1.1 Wind Energy

Wind energy is energy that is transformed from the renewable kinetic power in wind into electrical energy using a wind turbine. The wind moves the blades that through the rotor drive a generator inside the nacelle. From the generator the electric power is transferred through wires and networks to the consumer [1].

Like most other energy sources, wind energy stems basically from the solar energy that is transferred to the earth. Wind is an air stream that seeks to equalize differences of pressures in the atmosphere, which, among other things, arise since the sun heats the air masses at different latitudes of the globe. In Norway, we find the best conditions for wind power production along the coast and also in mountainous areas near the coast. Wind conditions vary widely, and to assess an area's potential for wind power extensive wind measurement must be carried out. Along the coast, the average wind speed reaches between 6 and 10 m/s. When the wind is stronger than 3-4 m/s the nacelle turns the turbine so that the rotor is always standing against the wind and power production starts.

Globally, the long-term technical potential of wind power is believed to be five times total current global energy production, or 40 times current electricity demand. This could require wind turbines to be installed over large areas, particularly in areas of higher wind resources. Offshore resources could contribute substantially more energy than that of land [2].

1.1.1 Offshore Wind Power

The major benefits of offshore wind turbines are: better and more stable wind resources, large available areas and likely lower interest conflicts than on land. The problem with current solutions is that investment, maintenance and operating costs are significantly higher than on land.

There are two main types of technology for Offshore Wind Power according to the depth where the turbines are installed. On the one hand, the turbines that are fixed at depths up to 100 m (real foundation), and on the other hand, the floating turbines in deeper waters.

The main advantage of floating wind turbines is that they can be located in deep water regardless of the seabed conditions. The world's first floating wind turbine is the group 2.3 MW HyWind Karmøy which started production in 2009, Figure 1. Norway has in addition, companies such as SWAY and Wind Sea working with Floating Wind Power.

Figure 1. Hywind in Åmøy Fjord, the first full scale Offshore Floating Windmill

As of September 2011, Japan plans to build a pilot floating wind farm, with six 2-megawatt turbines, off the Fukushima coast of northeast Japan [3].

As of November 2011, Statoil plans to build a multi-turbine project in Scottish waters utilizing the Hywind design [4].

The exploitation of open sea (deep-water) wind energy resources can give access to a large number of sites with good wind conditions and fewer restrictions (noise, visual acceptance) than on-shore. Wind turbines used here may differ from conventional designs due to different operational conditions and the absence of some restrictions [5].

1.2 Wind Generation in Europe

Wind power has an important role in the EU's future energy system. EWEA publishes information annually about the installed wind power in Europe.

Around 5.5% of EU electricity consumption was covered by wind power in 2010. The percentage should increase to around 15% in 2020 and to around 30% in 2030 to reach the EU goals. The figure below shows the new installed capacity for the different forms of energy in the EU from 1995 to 2010. Wind power represents a significant share of new power installed [6].

Since 2009, the average size of offshore wind farms has been increasing steadily. In 2011, the average size of the projects, once fully completed, is just under 200 MW, 45 MW (+29%) more than in 2010 [7].

As the technology matures it is expected that wind farms will continue to grow in size.

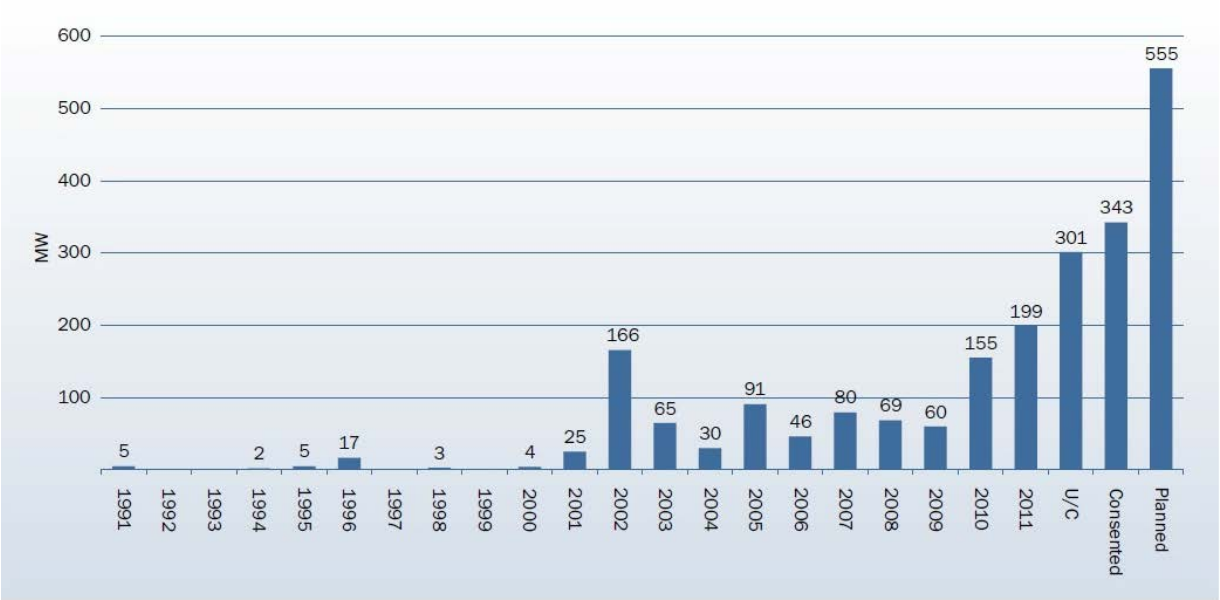

Figure 2. Average wind farm size in MW

The average capacity of offshore wind turbines was 2 MW at the end 2011. Average size of turbines grid connected during 2011 was 3.6 MW up from 3 MW in 2010

Over the course of the next decade the way in which Europe's demand for electricity is met will change fundamentally. Driven by a political will to meet ambitious targets for reductions in emissions of carbon dioxide and supported by generous subsidy schemes, electricity generation from renewable sources is expected to grow explosively between now and 2020 [8].

1.2.1 Wind in Norway

Norway will have 17 wind power plants of 1.2 MW, at the end of 2011, which together have an installed capacity of almost 541 MW.

Table 1. Wind power plants in Norway at the end of 2011

NAME	COMPANY	MUNICIPALITY	COUNTY	INSTALLED CAPACITY
Bessakerfjellet	TrønderEnergi Kraft AS	Roan	Sør-Trøndelag	57,50 MW
Fjeldskår	Agder Energi Produksjon AS	Lindesnes	Vest-Agder	3,75 MW
Harøy	Sandøy Vindkraft AS	Sandøy	Møre og Romsdal	3,75 MW
Havøygavlen	Artic Wind AS	Måsøy	Finnmark	40,00 MW
Hitra	Statkraft SF	Hitra	Sør-Trøndelag	55,20 MW
Hundhammerfjellet	NTE	Nærøy	Nord-Trøndelag	49,65 MW
Hywind	StatoilHydro ASA	Karmøy	Rogaland	2,30 MW
Kjøllefjord	Statkraft SF	Lebesby	Finnmark	39,10 MW
Mehuken	Kvalheim Kraft AS	Vågsøy	Sogn og Fjordane	4,25 MW
Mehuken II	Kvalheim Kraft AS	Vågsøy	Sogn og Fjordane	18,40 MW
Nygårdsfjellet trinn I	Nordkraft Vind AS	Narvik	Nordland	6,90 MW
Smøla trinn 1	Statkraft SF	Smøla	Møre og Romsdal	40,00 MW
Smøla trinn 2	Statkraft SF	Smøla	Møre og Romsdal	110,00 MW
Utsira	Norsk Hydro Produksjon AS	Utsira	Rogaland	1,20 MW
Valsneset	TrønderEnergi Kraft AS	Bjugn	Sør-Trøndelag	9,00 MW
Høg-Jæren trinn 1	Jæren Energi AS	Time/Hå	Rogaland	59,80 MW
Nygårdsfjellet trinn II	Nordkraft Vind AS	Narvik	Nordland	40,00 MW
Total	17			**540,80 MW**

Norwegian Wind power could become Europe's battery. Norway ought to have access to up to 40 terawatt hours of renewable energy in 2020-2025, of which about half would come from offshore wind power [9].

As is shown in table 2 there are future projects in the Offshore Wind Industry.

Norway has an excellent wind power potential. Typical sites at the coast have annual mean winds in the range of 8 to 10 m/s. This is considerably better than the typical wind conditions in Denmark or northern Germany. Wind farms located along the Norwegian coastline are expected to operate very efficiently [10].

Table 2. Status of the Offshore Wind Turbines in January 2011

COMPANY	COUNTY	NAME	MW	FOUNDATION	STATUS
StatoilHydro	Rogaland	HyWind	3	Floating	Licence - Operating
Havgul	Møre og Romsdal	Havsul I	350	Fixed base	Licence
SWAY	Rogaland	SWAY	10	Floating	Licence
Vestavind Kraft	Sogn og Fjordane	Testområde Stadt	10	Floating	Licence
Lyse Produksjon	Rogaland	Pilot Karmøy	10	Fixed base	Licence - Appealed
Lyse Produksjon	Rogaland	Pilot Rennesøy	10	Fixed base	Licence - Appealed
Lyse Produksjon	Rogaland	Pilot Kvitsøy	10	Fixed base	Licence - Appealed
Siragrunnen	Rogaland/Vest-Agder	Siragrunnen	200	Fixed base	Processing - Pending
TrønderEnergi Kraft	Møre og Romsdal	Mørevind	1200	Fixed base	Notification
Offshore Vindenergi	Møre og Romsdal	Steinshamn	105	Fixed base	Notification
Lofotkraft Vind	Nordland	Gimsøy Offshorepark	250	Unknown	Notification
Lofotkraft Vind	Nordland	Lofoten havkraftverk	750	Unknown	Notification
Nord-Norsk Vindkraft	Nordland	Selvær	450	Unknown	Notification
Offshore Vindenergi	Sør-Trøndelag	Fosen Offshore	600	Fixed base	Notification
Fred. Olsen Renewables	Nord-Trøndelag	Aegir	1200	Unknown	Notification
OceanWind	Vest-Agder	Ægir	1000	Fixed base	Notification
Lyse Produksjon	Rogaland	Utsira	300	Floating	Notification
Lyse Produksjon	Rogaland	Utsira pilot	25	Floating	Notification
Vestavind Kraft	Sogn og Fjordane	Stadtvind	1080	Unknown	Notification
Fred. Olsen Renewables	Vest-Agder	Idunn	1100	Unknown	Notification
Lyse Produksjon	Vest-Agder	Sørlige Nordsjøen	1000	Unknown	Notification
Troms Kraft Produksjon	Troms	Vannøya	775	Unknown	Notification

Without any doubt, wind energy will be one of the major forms of energy to take into account in a few years time.

Maps of Offshore Wind Farms in Europe and specifically of Scandinavia are shown in Appendix A

1.3 Offshore Inspection and Remote Presence

As an introduction, the subject matter is Inspection in an Offshore Wind Turbine. The main theme of the thesis is the demonstration of the efficiency in this kind of wind farm.

The importance of offshore projects has been shown in the previous section. Due to this importance, the development of cheap and reliable inspection systems is required. In fact, regarding the cost of energy from offshore wind, a general view is emerging that it is a better idea to invest in reliability to avoid maintenance than in equipment to facilitate it [11]. Taking this into account, this master thesis aims to enhance the alternative of Remote Inspection.

1.4 The Objective: How to study the Efficiency

Generally, the word “efficiency” describes the extent to which time or effort is well used for the intended task or purpose. It can be said that “efficiency” means no waste, or at least the minimum quantity of waste, expense or unnecessary effort.

Again, in general, “Efficiency” is a measurable concept. In our field of study the “Efficiency” of Remote Inspection is closely related to the “effectiveness”. The reason is that at some point of the inspection, achieving the goal will be the important factor.

1.4.1 Contribution

The contribution from this thesis is summarized with the following points:

- A huge investigation about components inside an Offshore Nacelle has been done.
- Operation, Inspection and Maintenance Strategies have been described in detail.
- Based on the first prototype, an alternative Inspection Procedure has been proposed.
- About the client controller and the interface, a study of industrial devices (Joysticks) in the market has been made.
- A Gamepad has been installed instead of a mouse (input device).
- The Java Code that was available has been updated in order to use the new device as the current controller.
- Experiments based on the efficiency of the current system have been made.

Part I.

Development of the Thesis

2

Previous Work

2.1 BACKGROUND 9
2.1.1 MOTIVATION 9
2.1.2 MADE DESIGN 10
2.2 PROTOTYPES 11
2.2.1 FIRST OPTION 11
2.2.2 SECOND OPTION 12
2.3 THE CHOSEN ROBOT 12
2.4 USER INTERFACE 13

2.1 Background

2.1.1 Motivation

Now it is time to remember the main reasons of why Remote Presence inside the nacelle of an Offshore Wind Farm.

Offshore inspection is expensive and must be performed on time. The possibility to have an Operator based in his office being able to inspect wind farms was very interesting. This would save money to Energy Companies due to:

- No matter the weather conditions on the sea during inspection plans.
- No need to hire transport service.

This idea was started by Viktor Fidje in 2010 and continued by Tor Karlsen in 2011. Below, what they were working with is explained and used.

The main goal of this project is to continue investigating and to demonstrate the efficiency of having a remote presence inside the nacelle.

2.1.2 Made Design

In this section the system that has been developed previously is explained. In the last two years two prototypes of Robot Inspection have been made. Further on, these will be explained in detail.

In order to avoid getting lost in theoretical work, on February 21st, 2011 there was a visit to GE Hundhammerfjellet to see one of the Wind Turbines installed in the park. In this thesis, it is helpful to have information on the visit mentioned. For this reason, from chapter 4 onwards, many references to that visit will be found in [12].

The current system consists of two Inspection Robots that are driven through a rail which serves as a guidance. In addition, the on-site communication has also a Supervisor Controller. Its main aim is to forward the messages sent from the Client (Operator). This subsystem supports the USART interface.

The photo below shows the shape of the track, which was built that way to cover as much space as possible inside the nacelle.

Figure 3. The whole system with the track

These rails are two tubes made by aluminium. The process of bending the tubes is not expensive, nor is the material.

2.2 Prototypes

As already mentioned, two prototypes were built. Each was design and built by a student during his Master Thesis. References are mentioned below.

2.2.1 First Option

It consists of two metal pieces joined together in such a way that only movement in the horizontal plane is allowed. Therefore we can deduce that this prototype is not suitable for ascend or descend the rail showed before.

The Robot is able to be controlled from all over the world, as long as the user is connected to the internet using the Remote Inspection software (Java Application). It has an IP Camera attached to the single wagon.

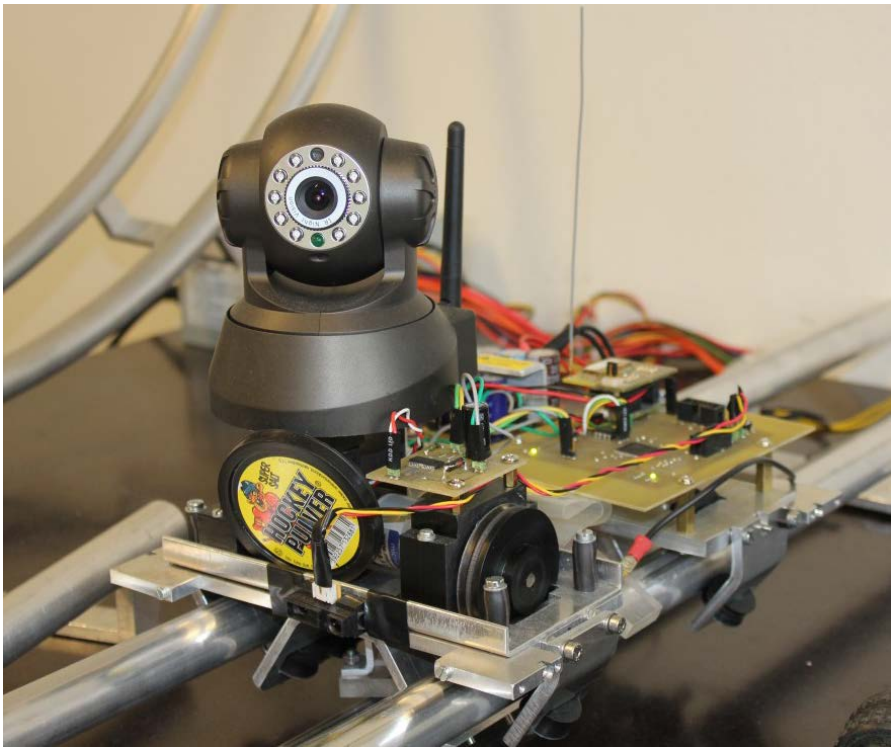

Figure 4. First Inspection Robot

The mechanics of this prototype is not too complicated. The wheel next to the camera is motorized. The other four little wheels down the cart keep it in the right position.

This was not enough for inspecting the inside of a real nacelle. Nevertheless, this was the first fully functional prototype made in the Department.

2.2.2 Second Option

It consists of three parts. The biggest part is placed in the middle and keeps the motor and an encoder, which will be really useful for some applications. The sections are connected with a joint that enables the mechanism to turn around freely in all directions.

To allow the robot to move vertically a gear was added to the motor. This should have been a good idea. The problem was that adding a chain around the guidance required much time and effort.

The Second Prototype is supposed to be an improved version of the first one. However, it is not completely done mechanically and, although, it has some advantages, it has also some drawbacks.

The main problems are that it is not finished, it is heavy and has no camera.

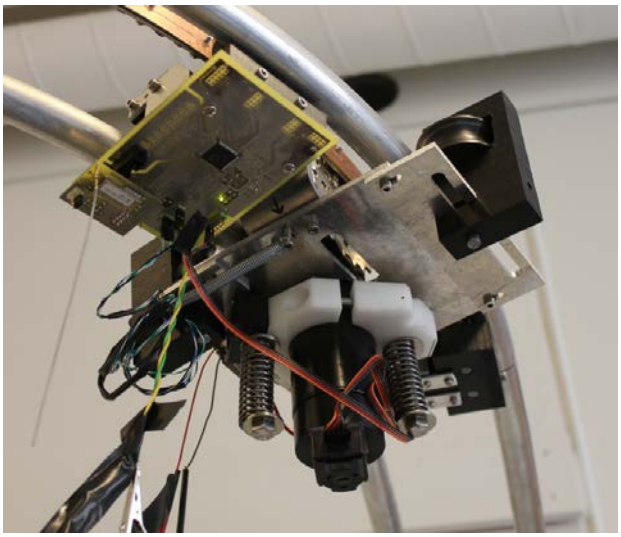

Figure 5. Last Inspection Prototype

2.3 The Chosen Robot

As has already been said, the IP Camera is only attached to the First Robot. This one is simpler and lighter. Therefore, to demonstrate the efficiency of the Inspection System we will use the First Prototype although it is not the most current working version.

The problem using the first prototype is that the path is very short (around 1 meter long). An extension rail could be a solution for future tests.

Another solution would be to use a model train, what makes much more versatile the inspection plan. However, there have been some problems that have delayed the assembly. In future work, a model train can be an interesting option.

In chapters 6 and 7 the use of this robot to demonstrate the efficiency of the Remote Presence is explained.

2.4 User Interface

The figure below shows the client interface. This Java application connects with the camera and the robot (prototype 1). As shown, the big white square is the Real Time Video from the camera and at the bottom of the picture there is the Control Panel. This panel is divided into two different panels. On the left, the Camera Control and on the right, the Vehicle Control.

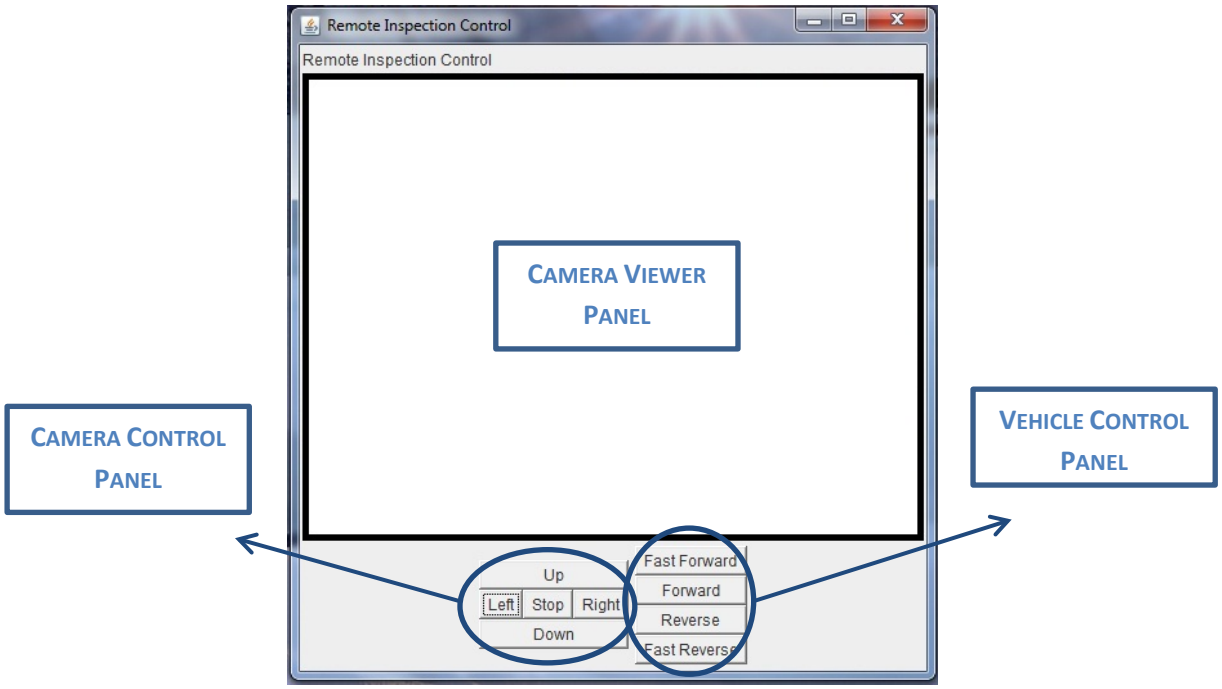

Figure 6. Client Remote Inspection Interface

The client (Operator) can control the robot by clicking with the mouse in each box. This is a really easy way to control the robot, but it can be improved.

This version has limitations on the final performance. The first limitation is that the Operator cannot move the camera and the vehicle at the same time. This is something that you can only do using another input device instead of a mouse.

It can also involve maintenance tasks. For example, adding another panel with a Robot arm Control, or even adding more views to provide a better concept of the space.

3

New Device: Joystick

3.1 CONTRIBUTION 15
3.1.1 TECHNICAL DETAILS 16
3.1.2 INDUSTRIAL APPLICATIONS 16
3.1.3 DEMANDED FEATURES 17
3.2 LIST OF POSSIBLE DEVICES 18
3.2.1 FINAL DEVICE 18
3.3 ASPECTS TO CONSIDER IN MAKING THE INTERFACE 19
3.3.1 DELAY 19
3.3.2 COLLISION 19
3.3.3 VIBRATION 19
3.3.4 PLACING & AUGMENTED REALITY 19
3.4 FINAL IMPLEMENTATION 22
3.4.1 DEVICE USED 22
3.4.2 JAVA PROGRAMING 23
3.4.2.1 JX Input – Input Devices for Java 23
3.4.2.2 Client-Inspection 25

3.1 Contribution

It could be interesting to have a different input device instead of a mouse. One of the most common motion controllers is the Joystick. It consists of a stick that pivots on a base and reports its angle or direction to the device it is controlling. It often has supplementary switches on it to control other secondary issues.

Joysticks are normally used to control video games. A variation of the joystick used on modern application is the analog stick. This variation is shown in the current gamepads. These devices can have a number of action buttons combined with one or more omnidirectional control sticks or buttons.

A control device such as a joystick is more comfortable to use and has unique features compared with an ordinary mouse. In the following section several types of devices and their functions will be exposed.

3.1.1 Technical details

Technologically, a joystick can be differentiated between Intrinsically Conductive Polymer (ICP) and Hall Effect technology. This refers to the material of their Resistance Elements. The advantages of the Hall Effect over the ICP are [13]:

- Immunity to dust, dirt, mud and water.
- This technology makes the Joystick a contactless controller.
- Simplicity in the measuring circuit, no additional resistance (shunt) needs to be inserted in the primary circuit.
- The voltage on the line to be sensed is not transmitted to the sensor, which enhances the safety of the measuring equipment.

On the other hand, the Hall voltage is often on the order of millivolts. This means that the output from the sensor must be amplified by a transistor-based circuit, which increases the price of this type of technology.

Another important issue to be distinguished is how to get the signal from the user. It can be an analog or a digital device. The difference is perfectly understandable. Depending on the applications you want to assign the controller, the use of an analogue or digital input will be recommended. The final implementation will show the differences according to the specific application.

Finally, another classification can be according to the Joystick's axis. This has to do with the virtual dimensions that can be represented. This will be explained in more detail below, in relation to the current application.

3.1.2 Industrial applications

Recently, the use of Joysticks and Pads has become commonplace in many industrial applications. It has virtually replaced the traditional mechanical control lever in almost all modern hydraulic control systems. Additionally, and concerning the aim of the thesis, most Remotely Operated Vehicles (ROVs) require at least one joystick to control the vehicle, the on board camera, sensors and manipulators [14].

Figure 7. Example of Industrial Joysticks

In Figure 7, an example of the wide variety of Joysticks on the market can be observed.

3.1.3 Demanded features

The Inspection Robot should have at this moment two different parts to be controlled. On the one hand, there is the wagon or cart and on the other hand, the IP Camera. Taking this into account is very important for the implementation of the controller.

First, the **cart** must move forwards and backwards. This kind of movement can be implemented with an analog joystick or even a digital one because of its obvious simplicity. Also, the wagon can be controlled by two buttons, if the controller is a gamepad instead of a joystick.

Possible improvements and important aspects [15]:

- If the wagon has variable speed, it is necessary to use an analog device.
- If possible, it would be very interesting to have a second device to perform the same action (forwards – backwards) but in a different way. This is explained below.

Thinking of every aspect that involves the wagon during an inspection, one important thing is the distance between the cart (camera) and the inspection area. Depending on that distance the device controller must be one or another. Here comes the idea of Auto centering button (short distances) vs. Put and stay (long distances).

To get a clear idea of these concepts, self centering has to do with *press* the button (paddle) to move and *release* to stop. In contrast, put and stay allows you to move forwards without holding the handle pressed. This is how boats work. Below, two examples of this last technology are shown.

Figure 8. Put'n Stay Joystick GT

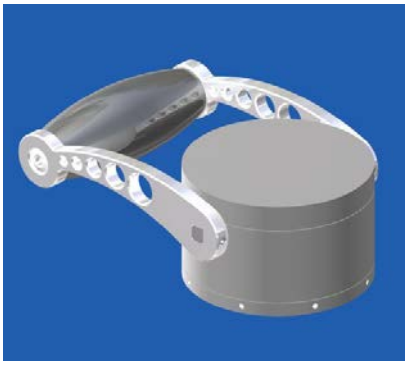

Figure 9. Marine Controller BS 130 GT

Finally, the **IP Camera** can move in four directions, as is shown in Figure 6. Again, this can be implemented with an analog joystick or even a digital one due to the inability to vary the rotation speed.

A significant improvement in the control of the IP Camera can be made by moving in two directions at once, taking advantage of the diagonals (corners) of the joystick.

3.2 List of possible devices

A List of Joysticks from Europe (Genge & Thoma) [13] and United States (OEM Controls) [16] can be found in Appendix B.

Below in the figure it has been used some examples of joysticks from these companies.

3.2.1 Final device

The final device must have implemented all the components previously mentioned in a single control block. An example of one possible configuration is shown below in Figure 10.

There are two blocks joined into one. From left to right there is the Vehicle Control Panel and the Camera Control Panel. This layout has been designed for a right-handed person, the control of the camera being the most important factor.

As has already been explained, there are two joysticks in the first panel. The first one is a Put'n Stay Joystick, allowing a comfortable use over long distances. The other is optimal for precision positioning. It has an ergonomic design and self-centering mid position.

The next device can be a 2 or 3-axis joystick. It should give the operator a comfortable way to move the camera.

Finally, in this way there would not be any problem in adding more panels side by side. So, in the future, when inspection and maintenance are feasible at the same time, it will be possible to make an "update" to the Control Panel.

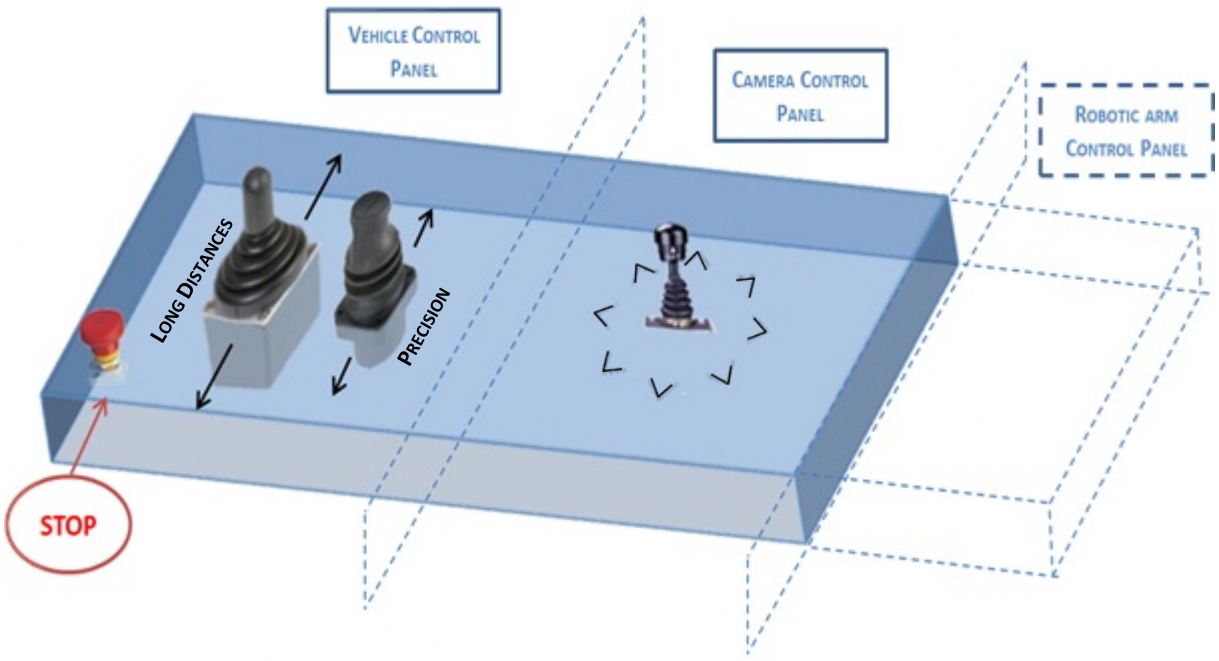

Figure 10. Control Panel for Inspection

3.3 Aspects to consider in making the interface

At this section, the main problems that appear in the interface while using remote presence and some possible solutions to be incorporated into the system are described.

3.3.1 Delay

Having delays or latencies is a fact that we should bear in mind when designing a new interface. Even though delays can be greatly reduced, these will never be eliminated completely. Why can this be a problem to our interface? In reference [17] some causes and consequences of this phenomenon are described. Delays are the main cause of inefficiency on the system.

Imagine pressing the "*forward button*" and having to wait 2 seconds in order to see the effect. It is obvious that this fact is first of all very annoying for the operator, who cannot perform his tasks correctly.

Two main causes of this latency are: the varying traffic on the Internet and the complexity of the video management. The traffic on the Internet is something random that you will never be able to predict. Only working with a private local network would let you act on this problem.

To conclude this issue, there are inherent latencies in the system itself during the video capture that should be reconsidered. Compression and decompression of the image frames represent the bottleneck of the system.

3.3.2 Collision

The aspect of collision is not very important during the inspection task. The camera is driven through a lane that restricts its trajectory. Nevertheless, while talking about maintenance tasks it becomes really critical. This importance is shown in chapter 4 in [17]. It was Eivind Berntsen who first studied causes and solutions in this field.

3.3.3 Vibration

To speed up the inspection, research on how to improve the Dynamic Quality of the Video could be interesting. In [18] an analysis of the dynamic image quality of a camera is made. In it, there is a study of the influence of the random disturbance on the video.

Moreover, in [19] Nikon Corporation made a camera system which is capable of reducing the influence of the vibrations produced by a known disturbance.

3.3.4 Placing & Augmented reality

Other thing that can be baffling is the problem of the inspector unknowing the position of the robot at any given time. This can or cannot be solved easily, by adding or changing something on the interface.

One idea can be to have a top or plan view map in a corner of the interface showing the position of the robot. This means having sensors installed along the rail, which is not an expensive idea.

Another option would be the possibility of using augmented reality. This kind of representation is widespread for last generation games. Professor Brian Blake said: “Augmented reality is changing the way we view the world”. This is for sure another way you could lessen the collision problem.

This recent technology blurs the line between what is real and what is computer-generated by enhancing what we see, hear, feel and smell.

On the spectrum between virtual reality, which creates immersive, computer-generated environments and the real world, augmented reality is definitely closer to the real world.

In the next two pictures, two examples of using the augmented reality technology are shown. These applications are not exactly suitable for our inspection interface, but each one has something interesting to contribute.

In addition, augmented reality companies are growing greatly in recent years. Companies such as, Total Immersion or AR & Co are good example of this growth.

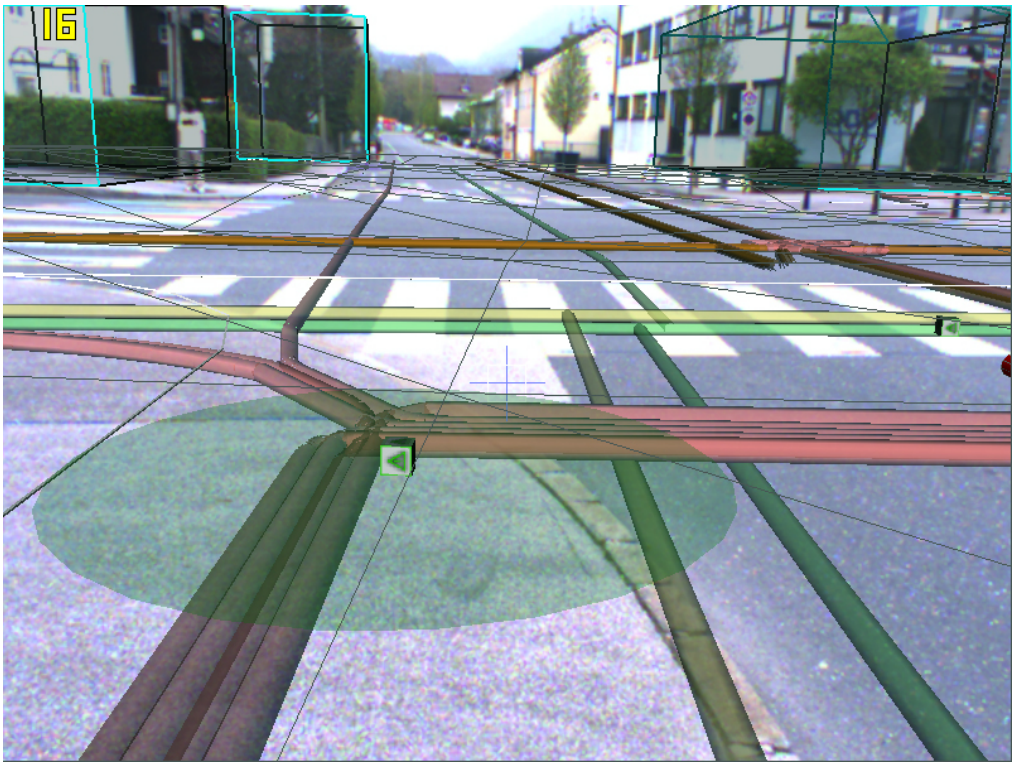

Figure 11. Augmented Reality System for 3D mapping

The picture above from [20] shows the map of a street pipes. This same proposal can be used inside a nacelle on an Offshore Wind Turbine. This gives us better information about the surroundings.

On the picture below, found in [21], another application for augmented reality is described. This technology makes possible to detect objects and to interact with them.

What is more, this technology has unlimited possibilities to create an optimal inspection & maintenance interface.

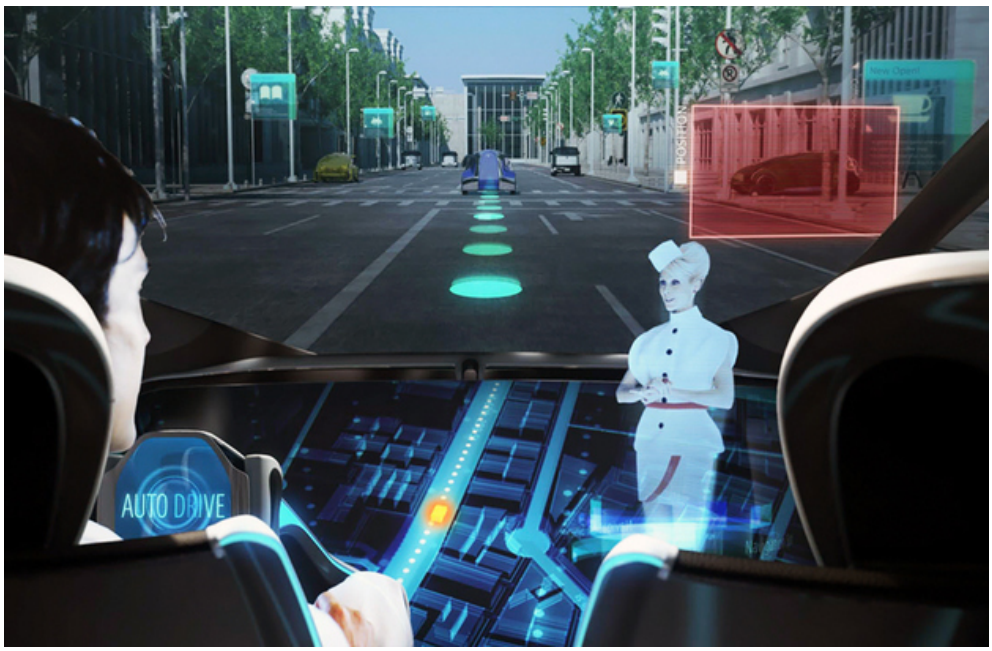

Figure 12. Augmented Reality System for a moving vehicle

It might be easier instead of using an augmented reality system, just to use a set of properly placed cameras inside the nacelle. However, in the near future the use of the augmented reality will be a necessity.

3.4 Final Implementation

At this point, all that remains is to expose what has actually been implemented in the practical project.

3.4.1 Device used

The theoretical design has been explained before; however, it has to be added that the device used in the inspection tests is a Game pad from Saitek. It is shown below.

Figure 13. Game pad used as the new Controller Device

The client (Operator) can control the Vehicle by moving the left “joypad” forwards or backwards. At first, we wanted that the vehicle speed was directly related to the pressure exerted. Then, we decided to keep just two options: normal and fast speed.

The Camera is controlled by the right “joypad”. The control is quite intuitive: UP, DOWN, LEFT and RIGHT in the direction of the arrows.

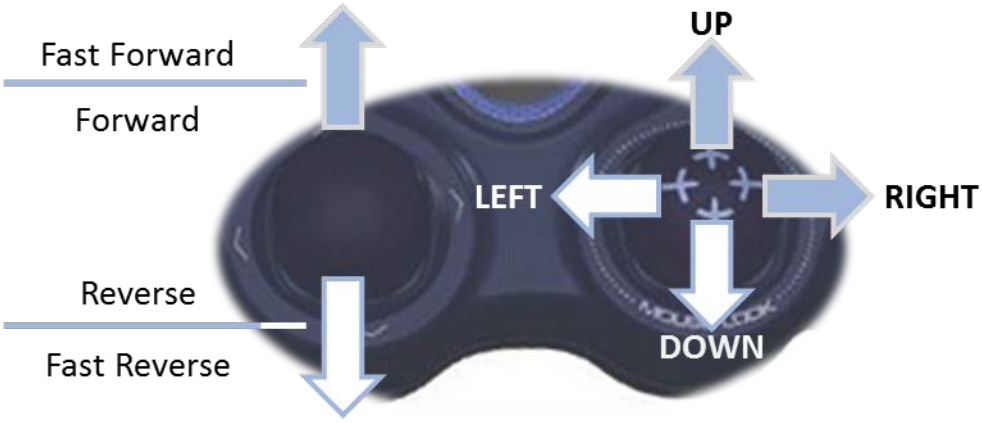

Figure 14. Control Scheme Joypads

In this version, the limitation which makes the Operator unable move the camera and the vehicle at the same time has been removed.

3.4.2 Java Programing

The first Java application made by Viktor Fidje in 2010, has been the beginning of the last version, updated in this Thesis.

3.4.2.1 JX Input – Input Devices for Java

JXInput gives access to any number of Direct Input gaming devices. The source code has been got from [22].

JXInput application allows to ask for the features available with a given device and later to query their respective values. This is the way most games work, polling the state of the input device within a loop.

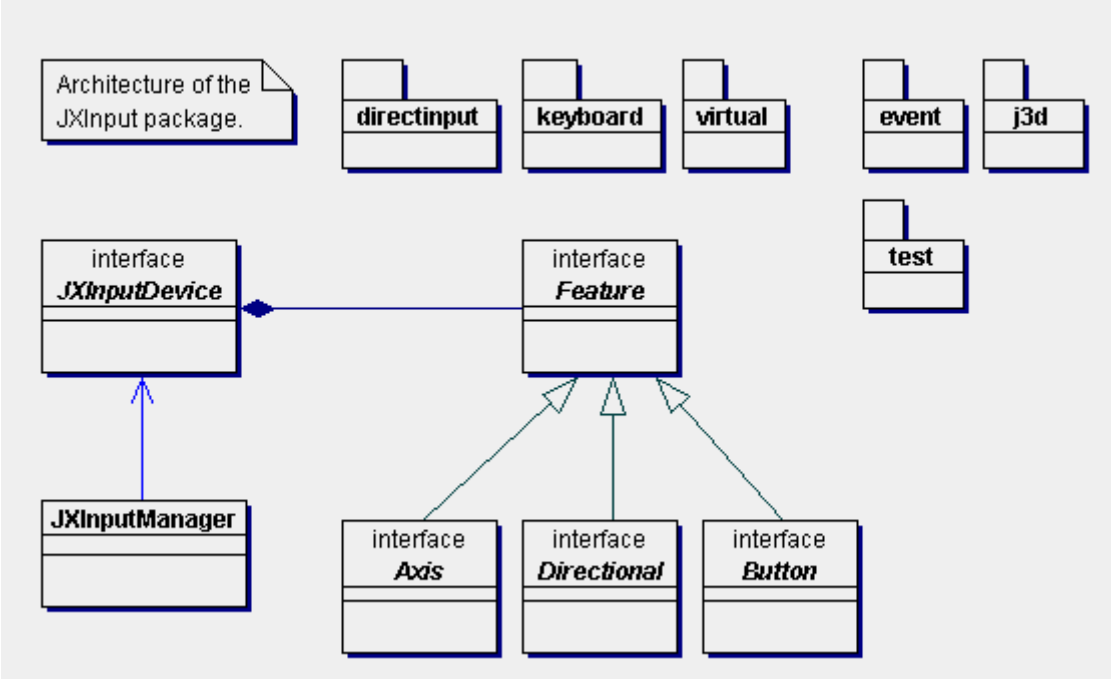

Figure 15. Architecture of the JXInput package

For event driven applications, JXInput allows to register "listener" objects with the features. This way, the application is called back by JXInput in case the input device changes its state. This is the mode operation has been implemented.

This application can be launched in Mac OS or Linux. This fact makes the application to be as capable as it is required.

The complete JXInput Source Code can be found in Appendix C. It is divided into 4 *classes*:

- JoystickListener.java: the listener interface for receiving joystick events.
- JoystickNotifier.java: notifies the joystick listeners of joysticks events.
- Joystick.java: *main class*. Device driver to a Windows joystick.

- The last class in the Appendix C shows a Joystick Test application. This has been modified in order to adapt it to our System.

In the picture below, the Joystick Test Interface is shown.

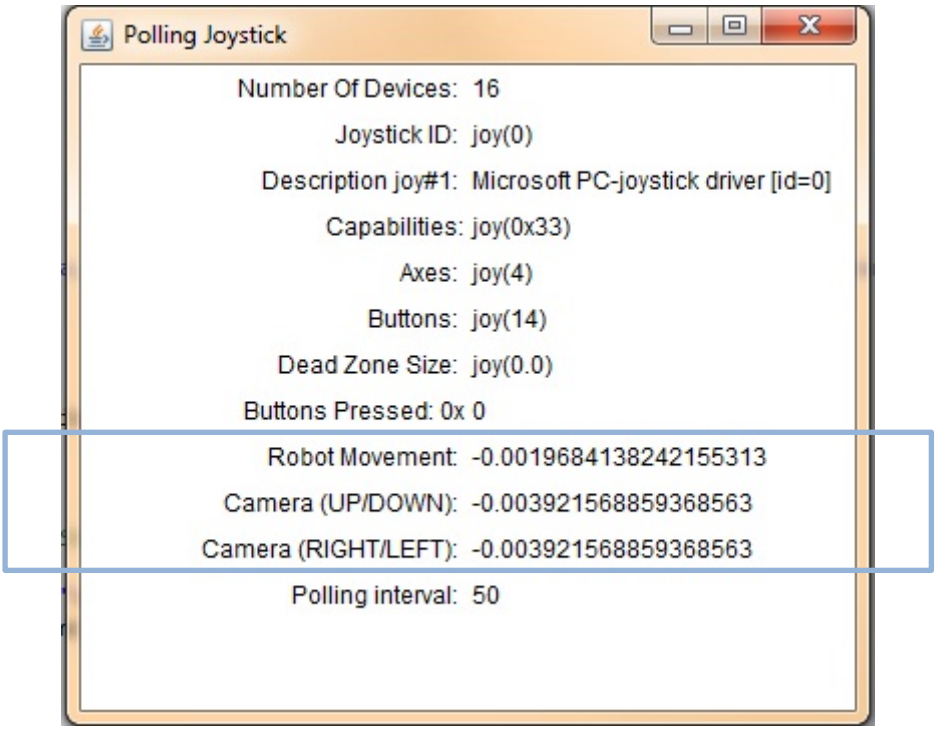

Figure 16. Joystick Test Interface

The top values from Robot and Camera movements are described in next figure. These values are important for the new source code.

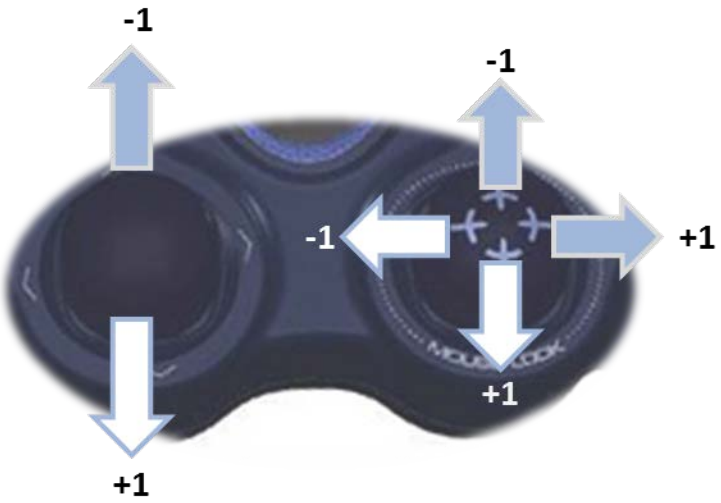

Figure 17. Top values for detection in the Java application

3.4.2.2 Client-Inspection

Having the java code specially made for a mouse control, updating this version was not going to be a big deal.

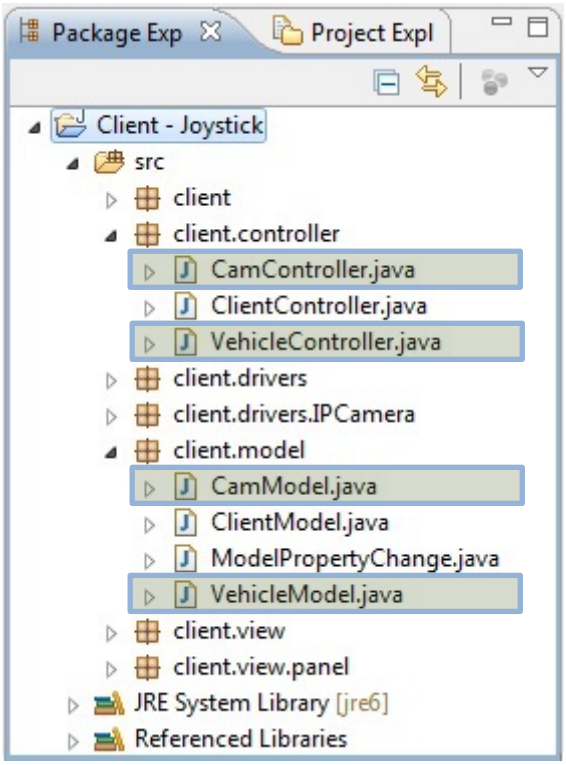

Figure 18. Screen Capture of Eclipse (Java classes to update)

The first step was to find which *java classes* might be modified. In the figure above, four *java classes* are highlighted. The modifications of these classes are found in the Appendix D.

- CamController.java: The camera controller class. One *createInstance* function is used in order to create a Joystick. Two new methods are created: *model.joylistener* (camera movement) and *model.keylistener*.
- VehicleController.java: The vehicle controller class. A *createInstance* function is used. The method *model.joylistener* (vehicle movement) is created.
- CamModel.java: The camera model class. Two new public variables called *joylistener* and *keylistener* are created.
- VehicleModel.java: The vehicle model class. The public variable called *joylistener* is added.

4

Inside the Nacelle

4.1 ANATOMY OF A WIND TURBINE 27
4.1.1 BLADES 28
4.1.2 NACELLE 28
4.1.3 TOWER 28
4.2 MAIN PARTS OF THE NACELLE 29
4.2.1 ROTOR HUB 29
4.2.1.1 Main Shaft 29
4.2.1.2 Small Shaft 29
4.2.2 GEARBOX 30
4.2.3 GENERATOR 30
4.2.3.1 Cooling System 30
4.2.4 MECHANICAL BRAKE 30
4.2.5 YAW DRIVE 30
4.2.5.1 Yaw motor 31
4.2.5.2 Yaw bearing 31
4.2.6 ANEMOMETER AND WIND VANE 31
4.2.7 CONTROLLER 32

4.1 Anatomy of a Wind Turbine

In this chapter, the layout of a typical Wind Turbine Generator, as found on wind farms across the world will be explained. Now exact designs do vary, but almost all the turbines that are used today are horizontal axes machines, which have a three-bladed rotor spinning in a vertical plane attached to the front of the box which is called the Nacelle. In this nacelle we have the generator, often a gearbox and occasionally a high voltage transformer. The whole thing sits on top of the tower which nowadays is usually about 80- 100 meters tall. Now, we will talk about the different parts of a Wind Turbine [23].

4.1.1 Blades

As mentioned previously, the rotor for a typical utility-scale wind turbine includes three high-tech blades, a hub, and a spinner. The blades are one of the most critical aspects for wind turbine and are considered a strategic component by wind turbine OEMs. Most manufacturers create multiple blade types for a single wind turbine in order to enhance performance in different wind conditions. The blades range in size from about 34 to 55 meters and are made of laminated material, such as composites, balsa wood, carbon fibre, and fibreglass. These have high strength-to-weight ratios. These materials are molded into airfoils to generate lift which causes the rotor to turn. The blades also often include material to protect them against lightning. They are bolted onto the hub, with a pitch mechanism interposed to allow the blade to rotate about its axis to take advantage of varying win speeds.

Figure 19. Example of an Offshore Wind Turbine

4.1.2 Nacelle

The nacelle of a wind turbine is the box-like component that sits atop the tower and is connected to the rotor. The nacelle contains the majority of the approximately 8,000 components of the wind turbine such as the gearbox, generators, main frame, etc. The nacelle housing is made of fiberglass and protects the internal components from the environment. The nacelle cover is fastened to the main frame, which also supports all the other components inside the nacelle. The main frames are large metal structures that must be able to withstand large fatigue loads.

4.1.3 Tower

The nacelle and generator are mounted on top of a high tower to allow the blades to take advantage of the best winds. The power available to a wind turbine is proportional to the cube of the wind speed. Therefore, a 10% increase in wind speed would result in a 33% increase in available power. Towers are typically made of three or four tubular steel sections coated with paints and sealants and joined by flanges and bolts. Most towers come with load lifting systems with a load-bearing capacity of more than 180 kilos (400 pounds).

4.2 Main parts of the Nacelle

It is important to know and understand the function of each part of the nacelle. In this section it is explained, in reasonable detail, how its components work [24].

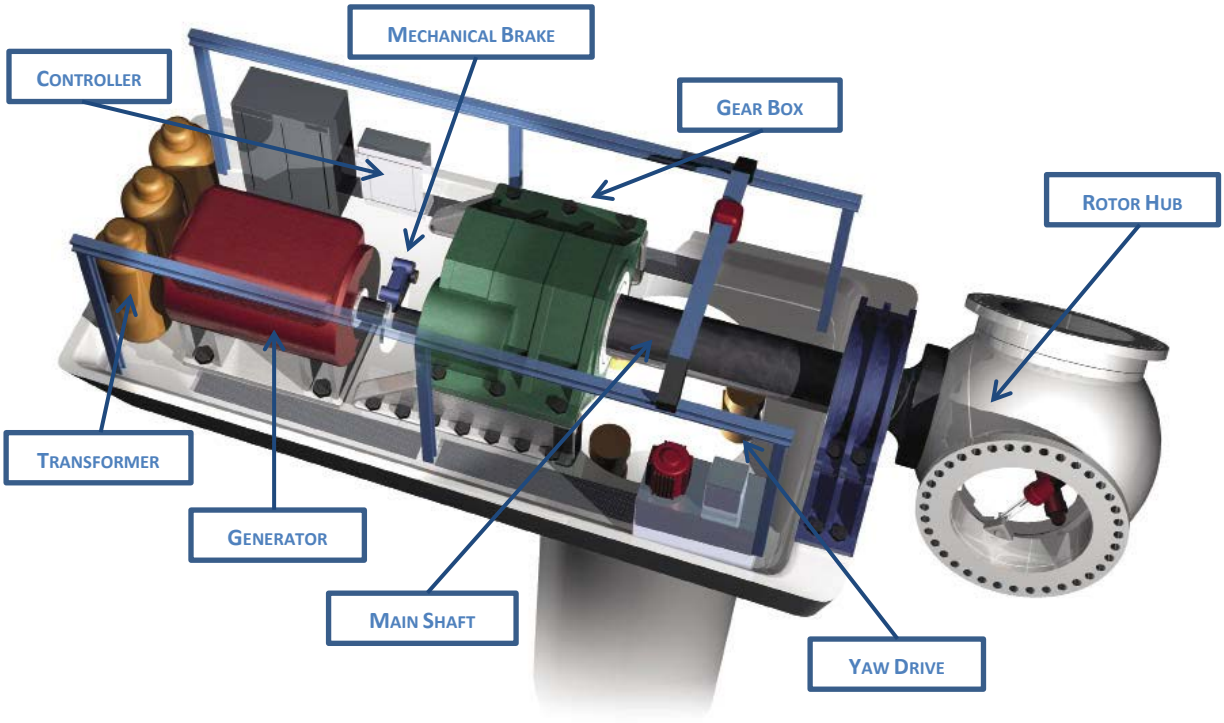

Figure 20. Main parts of a Nacelle

4.2.1 Rotor hub

The rigid rotor hub is a nodular cast iron structure mounted onto the main bearing. Blades are connected to the hub using a four point pre-stressed ball bearing. The hub houses the blade pitch control system and a slip-ring unit for the power supply. The hub is sufficiently large to provide a comfortable working environment for two service technicians during maintenance of the pitch, pitch bearings and blade root from inside the structure [25].

The rotation of the rotor is driven through the main shaft. Consequently, this rotation is transmitted to the generator through the small shaft (faster).

4.2.1.1 Main Shaft

The rotor is bolted to a very strong disc on the main shaft of the wind turbine. It is important that the rotor is firmly secured by a lot of bolts. The gear box is placed at the end of the main shaft.

4.2.1.2 Small Shaft

The small shaft connects the generator to the gearbox. This shaft does not have to transfer as much turning force as the main shaft does. That is why it is a lot thinner. On the other hand, it revolves very quickly: 1500 revolutions per minute.

4.2.2 Gearbox

The function of the gearbox is to step up the speed of rotor rotation to a value suitable for standard induction generators, which, in the case of fixed speed machines or two speed machines operating at the higher speed, is usually 1500 rpm plus the requisite slip [24].

The gearbox then changes the turning force, so instead of revolving slowly with a lot of force in every revolution, it now has to go faster with less force in every revolution.

Gearbox efficiency can vary between about 95% and 98%, depending on the relative number of epicyclic and parallel shaft stages and on the type of lubrication.

4.2.3 Generator

The induction generators commonly used in fixed-speed wind turbines are very similar to conventional industrial induction motors.

4.2.3.1 Cooling System

When the generator is running it gets hot. If it gets too hot it can break down. Therefore, it is necessary to cool down the generator before it becomes so hot that it stops working. The generator can be cooled in two ways, either by air or water.

When cooling with water, cold water is led into some pipes hidden in the shell of the generator. The water cools down the generator and thereby heats itself up. The radiator uses the surrounding air to cool down the water again. So the water can permanently circulate while cooling the generator.

4.2.4 Mechanical brake

A wind turbine has two different types of brakes. One is the blade tip brake. The other is a mechanical brake, which is placed on the small fast shaft between the gearbox and the generator. It is only used as an emergency brake, if the blade tip brake fails.

The brake is also used when the wind turbine is being repaired to eliminate any risk of turning on the turbine.

4.2.5 Yaw Drive

This mechanism is used to rotate the nacelle with respect to the tower on its slewing bearing, in order to keep the turbine facing into the wind so the wind turbine will generate as much electricity as possible; and to unwind the power and other cables when they become excessively twisted [24].

Figure below shows an example of an arrangement of yaw drive.

4.2.5.1 Yaw motor

The yaw motor turns the nacelle so that the rotor faces the wind. Underneath the yaw motor there is a small wheel that engages the wheel of the yaw bearing.

4.2.5.2 Yaw bearing

The yaw motor has a small wheel that engages a huge wheel. The large wheel is called yaw bearing. On some yaw bearings the teeth point outwards, while on others they are turned inwards. It all depends on the position of the yaw motor.

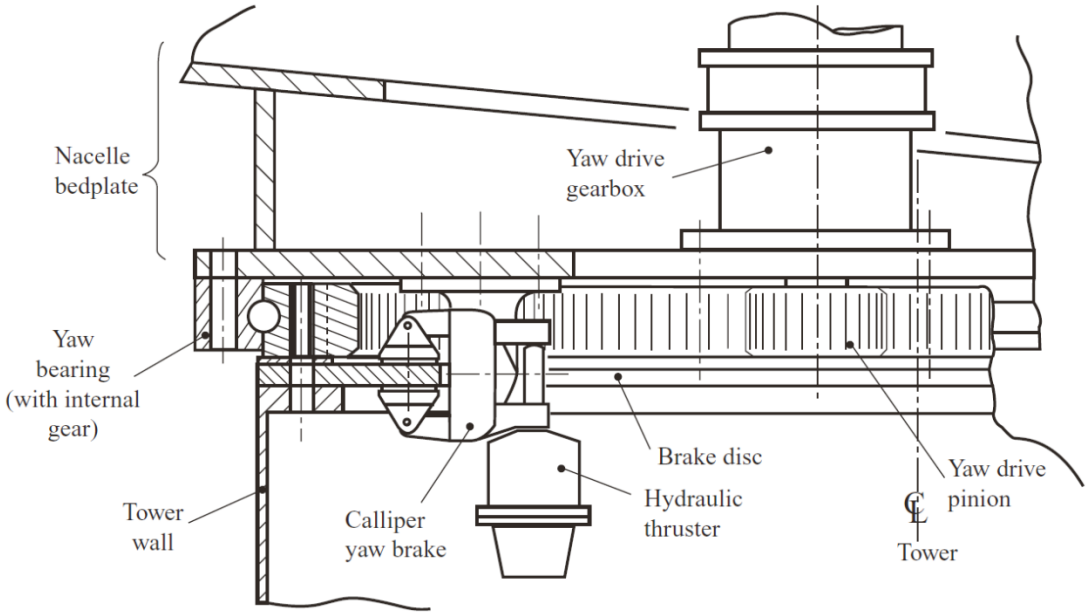

Figure 21. Typical arrangement of yaw bearing and yaw drive

4.2.6 Anemometer and Wind Vane

The anemometer measures the wind speed and notifies the wind controller the right time to turn on the turbine. It must be very windy to use power to make the wind turbine turn (yaw) into the wind and start running. It is important to know how much wind there is. If the wind is too strong the wind turbine can break. This is why the turbine is brought to a stop when the wind exceeds 25 metres per second. When the wind drops, the anemometer tells the controller that it is OK to start the turbine again.

Figure 22. Anemometer and wind vane

A wind vane always positions itself according to the wind direction. There is a small sensor at the foot of the wind vane that notifies the wind turbine controller of the wind direction. The controller tells the yaw motor to yaw (turn) the nacelle so that the rotor faces the wind.

4.2.7 Controller

The wind turbine is controlled by several computers that keep an eye on many different things. Together, these computers are called the wind turbine control system. The main computer is called the controller. In Figure below, there is an example of how the Control System works [26].

Conventional wind turbine controllers have only limited resources, and can only offer restricted monitoring and diagnostic functions [27]. System developers expect controllers to not only monitor environmental conditions or temperatures, but also allow remote management (network).

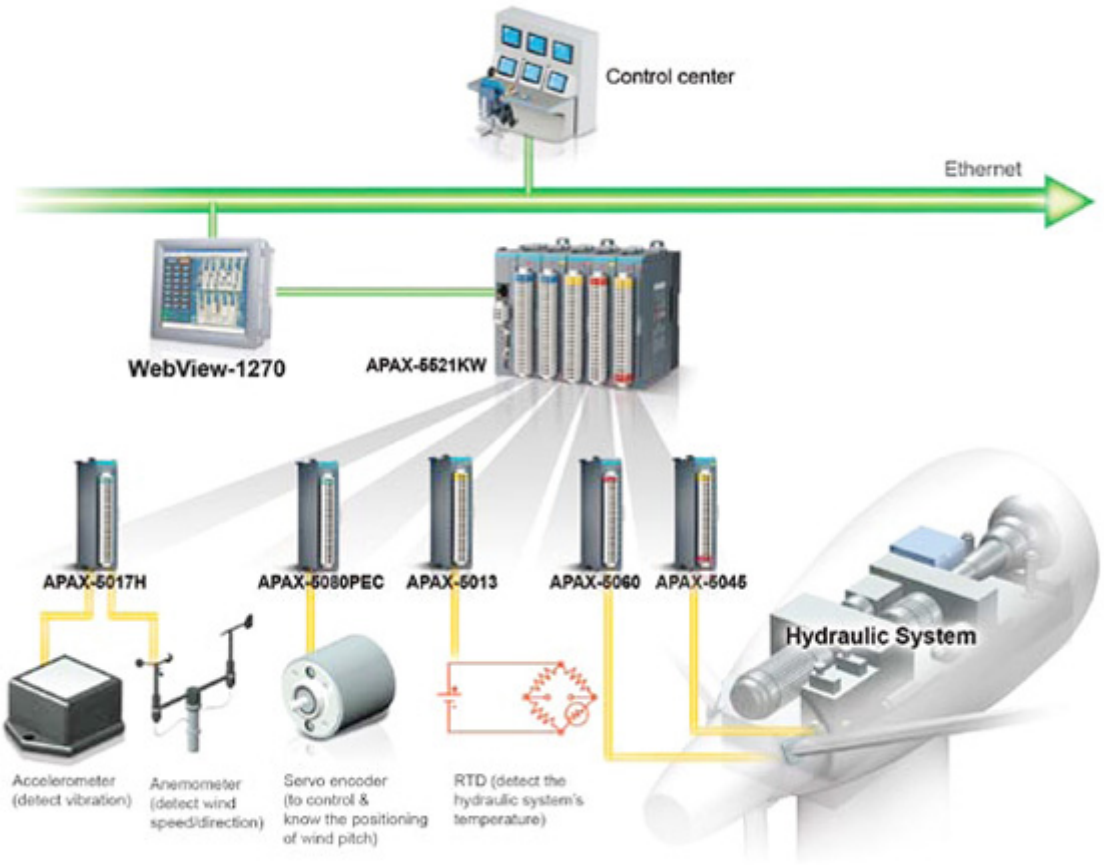

Figure 23. Diagram of an Advantech APAX Control System

A Control System is fitted with a number of sensors to read the speed and direction of the wind, the levels of electrical power generation, the rotor speed, the blades' pitch angle, vibration levels, the temperature of the lubricants and other variables. A computer processes the inputs to carry out the normal operation of the turbine, with a safety system which can override the controller in an emergency. The control system protects the turbine from operating in dangerous conditions and ensures that the power generated has the proper frequency, voltage, and current levels to be supplied to the grid.

When a change has to be made on the adjustment of the turbine, the controller takes care of it. It always keeps an eye on whether or not everything in the wind turbine works as it should.

5

Operation and Maintenance

5.1 INTRODUCTION 33
5.1.1 LAND BASED COMPARATIVE DATA 34
5.2 MAINTENANCE STRATEGIES 34
5.3 O&M OFFSHORE EXPERIENCE 35
5.3.1 AVAILABILITY 35
5.3.2 OPERATIONAL EXPENDITURE 35
5.3.3 SERVICEABILITY 35
5.3.4 ACCESS FOR MAINTENANCE 36
5.4 DESIGNS FOR REDUCED MAINTENANCE 36
5.4.1 COMPONENT RELIABILITY 36
5.4.1.1 Gearbox 37
5.4.1.2 Generator 37
5.4.1.3 Direct Drive System 37
5.4.1.4 Electrical and Electronic Control System 37
5.4.1.5 Hydraulic System 37
5.4.2 CORROSION PROTECTION 38
5.4.3 CONTROL AND CONDITION MONITORING 38
5.4.4 BACK-UP POWER 38
5.5 O&M CONCLUSIONS 39

5.1 Introduction

Operation and Maintenance of offshore wind farms is more difficult and expensive than equivalent onshore wind farms. Offshore conditions cause more onerous erection and commissioning operations and accessibility for routine servicing and maintenance is a major concern. During harsh winter conditions, a complete wind farm may be inaccessible for a number of days due to sea, wind and visibility conditions.

The severe weather conditions experienced by an OWECS (Offshore Wind Energy Conversion Systems) dictate the requirement for high reliability components coupled with adequate environmental protection for virtually all components exposed to sea conditions.

Consequently, the requirement for remote monitoring and visual inspection becomes more important to maintain appropriate turbine availability levels.

5.1.1 Land Based Comparative Data

Operational information for onshore wind turbines has been compiled for a number of years which is directly relevant for operation and maintenance issues.

Operation and maintenance data for onshore wind turbines are readily available as detailed above. However, the environmental conditions associated with offshore installations render this current machine data inadequate.

5.2 Maintenance Strategies

The availability of a wind turbine depends on the O&M strategy adopted by the operators of a wind farm. Given the limited amount of offshore O&M data, strategic planning is in its infancy; however a number of options were developed in the Opti-OWECS study [28]:

1. **No maintenance**: Neither preventive nor corrective maintenance are executed, and major overhauls are performed every five years or so. One of the few alternatives is exchanging a whole turbine if availability drops below a predefined minimum or after a certain amount of operational hours. Given the current level of turbine failure rates, this option is not presently viable.
2. **Corrective maintenance only**: Repair carried out soon after a turbine is down, or, alternatively, wait until a certain number of turbines are down. No permanent maintenance crew is needed.
3. **Opportunity maintenance**: Executing corrective maintenance on demand and taking the opportunity to perform preventive maintenance at the same time. No permanent maintenance crew is needed.
4. **Periodic maintenance**: Scheduled visits performing preventive maintenance, and corrective actions performed as necessary by a permanent dedicated maintenance crew.

In conclusion, given current reliability and failure modes of commercial offshore wind turbines, which have been adapted from onshore models, a reduced level of preventive and corrective maintenance is not a viable option at this stage in the development of the offshore wind energy industry.

Whenever remote monitoring techniques become sufficiently mature they can be used to adopt a maintenance strategy that is guided by the status of the components. Maintenance visits can be paid just in time, in order to prevent failure of the wind turbine and of the other monitored components of the wind farm.

5.3 O&M Offshore Experience

The following four sections explain different aspects of O&M of offshore wind turbines.

5.3.1 Availability

The availability of a wind farm, defined as the percentage of time it is able to produce electricity, is a function of the reliability, maintainability and serviceability of hardware and software used in the whole system. For an offshore wind farm however, the accessibility of the site for O&M hardware equipment as well as the adopted maintenance strategy are of an equal importance for the achieved availability level.

5.3.2 Operational expenditure

As stated above, operating expenditure for offshore wind farms is considerably higher than the equivalent onshore facility. Offshore operations are in the region of between five and ten times more expensive than work on land, and these costs are exacerbated by inflated prices prevalent within the offshore oil and gas industry.

Also, onshore equipment can be sourced and mobilised within a short period of time, usually within hours, and available on site within a day. Offshore lifting cranes are uncommon, and will generally have to travel a considerable distance to an offshore wind farm site, hence the requirement for careful scheduling of such vessel movements. The economics of a large wind farm may justify the purchase of a dedicated purpose-built lifting vessel which would be available during installation and for maintenance throughout the wind farm's lifetime. However, it is commercially expedient to dispense with the need for expensive lifting vessels after installation and hire lifting equipment during scheduled major overhaul. Given relatively calm sea conditions, it is possible to use a floating barge to transport and operate a land crane offshore. The floating barge need only be a crude construction incurring minimal expenditure, hence be procured and stored at a dedicated wind farm [29].

General maintenance tasks are carried out using less specialised equipment which is generally purchased for the designed life of the wind farm.

5.3.3 Serviceability

The service demand of the present generation of offshore wind turbines in terms of man-hour is in the order of 40 to 80 hours [30]. Service visits are paid regularly, (except in the more demanding first year) about every six months. A more major overhaul will be undertaken every five years, and will take around 100 man-hours to complete.

Experience from Tuno Knob shows that the total number of visits have been about 35 to 70 visits per year, an average of approximately 5 visits per turbine per annum. The number of cancelled visits (last moment cancellations due to the weather) makes up about 15% relative to the number of service visits realised [31].

5.3.4 Access for maintenance

Access to an Offshore Wind Turbine for routine servicing and maintenance is difficult or impossible in hard weather conditions due to wave heights, wind speeds and poor visibility. The traditional method for transporting personnel and equipment is by boat, which is in many cases limited to relatively benign sea states.

Since the beginning of offshore wind farm development, suggested methods for gaining safe access have included, such as [32]:

- Helicopter.
- Underwater tunnels.
- Wheeled platforms for turbines in close proximity to the shoreline.
- Amphibious vehicles where caterpillar tracks transport a platform over a firm and stale seabed.
- Small hovercraft or ice roads for frozen seas.

The advantages and disadvantages of the boat compared to the helicopter are now discussed.

On the one hand, boat access has some advantages such as, well proven method of inshore transportation and relatively cheap equipment expenditure. The main problem is that this means of transport is highly dependent on maritime conditions. It is impractical for wave heights greater than 1m and it is very difficult to transfer personnel equipment in rough conditions.

On the other hand, there is a possibility of using helicopter access. In this case, sea state is not a major issue and the transportation of personnel and equipment from land to turbines is much faster. The main drawbacks of helicopter transport are the cost and terms of visibility and wind.

In conclusion, there are a number of current projects addressing the issue of improved access to offshore wind turbine installations. Most focus on maintaining existing boat access methods with emphasis on addressing the issue of motion compensation or complete removal of the vessels from the water at the turbine location. Talking about large wind farms, a possibility is to use small purpose built jack-up vessels. However, access using small purpose-built landing craft continues to present the most pragmatic and economic solution [33].

5.4 Designs for Reduced Maintenance

The issue of accessibility can also be addressed by improvements in offshore wind turbine reliability. Both planned and, more importantly, unplanned maintenance levels can be reduced by increasing the reliability and hence availability of the turbine. Particular emphasis is being placed on reliability issues from component level through to overall design improvements.

5.4.1 Component reliability

Although all the different components of the wind turbine could be discussed we will focus only on the most interesting parts inside the nacelle.

5.4.1.1 Gearbox

Current offshore turbines manufactured by leading manufacturers favour geared drive transmissions. Being widely recognised as the number one item for mechanical failure and servicing supervision, it would appear a progressive step to move to direct drive systems.

5.4.1.2 Generator

In general, induction generators require less maintenance than synchronous generators. They do not require a DC source and being inherently simpler and rugged are the most common generators in onshore wind turbines.

To protect the generator from marine environments, it is totally enclosed to protect the internals from salt and high levels of moisture.

For offshore application it is not recommended to use air cooling generators as in onshore applications. A better solution is to work with closed system water cooling or even air to air heat exchange. These alternatives prevent the risk of corrosion from marine cooling air.

5.4.1.3 Direct Drive System

This type of system dispenses with the historically problematic gearbox, where the drive train, generator and rotor rotate at the same speed of around 20 rpm for a 2 MW turbine.

Having direct drive generators has obvious advantages: no gearbox with associated high speed rotating parts, no gearbox oil contamination and leakage, and less routine servicing. However, this type of generator is very heavy and bulky for megawatt turbines. Therefore, the large diameter required changes the visual appearance of the nacelle. It increases tower stress and hence tower dimensions.

5.4.1.4 Electrical and Electronic Control System

The failures in this system account for the highest percentage of failures. For the year 2000, failures of electrical and controls systems accounted for exactly 50% of the need for wind turbines repairs. Typically, failures of this nature occur due to the number of components, poor electrical connections, corrosion, lightning, etc.

The main solution for offshore conditions is to reduce the number of components and the use of electronic printed circuit boards.

5.4.1.5 Hydraulic System

Wherever possible, elimination of problematic hydraulic systems employed in yaw damping, blade pitching and breaking systems should be realised. It is preferable to use electrical actuation instead of hydraulic. The main reason is that it eliminates the possibility of oil leakage.

5.4.2 Corrosion protection

The main methods of marine corrosion protection for offshore installations, recently developed within the offshore oil and gas industry, are the selection of corrosion resistant, two-pack epoxy coating, cathodic protection, and the creation of controlled environments for sensitive equipment.

More work is needed in developing support structures which can withstand stress caused by wind and wave loading, together with reductions in materials fatigue strength caused by corrosion.

5.4.3 Control and condition monitoring

Surveys of machine outages reveal that around half the unplanned shutdowns on onshore turbines are caused by faults in the electrical and electronic control systems. To reduce the number of unplanned visits to offshore turbines, automatic re-set and remote re-set facilities are now becoming common in all new turbines. Increasing numbers of sensors and monitoring equipment are being used, and the signals categorised to register: data, minor faults requiring notification only or major faults which shut the turbine down automatically.

Using SCADA systems, monitored signals and alarms are transmitted between the turbine and the onshore control station. Control personnel can interact with the monitoring system to over-ride the turbine controller if necessary.

Internet connection, webcams and sophisticated vibration monitoring, for example, can now be utilised to detect a limited number of pending failures prior to their occurrence.

5.4.4 Back-up power

The grid system is responsible for the power for the turbine controller, electrical actuator, and monitoring and communications systems.

This option is interesting in the case of emergency, in the event of loss of turbine power generation or lost electrical grid connection. It should be useful for maintenance work or to keep turbine systems running.

5.5 O&M Conclusions

It can be highlighted that the future wind turbine development for Offshore Wind Farms will adapt existing onshore designs to cope with harsh maritime environments [34].

Below is a list of requirements to reduce the lifetime O&M costs of offshore wind turbines.

- Improvement of access methods.
- Reduction of time required for offshore working.
- Designs for reduced maintenance by:
  - Reduction of overall number of components and simplicity of design.
  - Modular design of wind turbines which facilitates the interchange of faulty modules.
  - Use of high reliability components.
  - Re-siting of electrical units into an environmentally controlled section of the turbine.
  - Implementation of offshore corrosion protection technology.
  - Development of effective condition monitoring and remote control systems.
- Development of appropriate maintenance strategies for service and repair actions.
- Development of an integral design philosophy of a large scale offshore wind turbine, where the design of the individual wind turbines is governed by the overall system targets, and not by a sequential adaptation and up scaling of onshore designs.

6

What to Inspect

6.1 GENERAL OUTLINES ABOUT THE INSPECTION 41
6.2 INSPECTING THE NACELLE 42
6.2.1 STRUCTURAL SYSTEM 42
6.2.2 ELECTRICAL AND MECHANICAL SYSTEM 43
6.3 COMPANIES GENERAL PROCEDURE 43

6.1 General outlines about the Inspection

The main aim of this thesis, as has been already said, is to demonstrate the Efficiency of a Remote Presence inside the Nacelle. As shown in chapter 5, improving the inspection and maintenance of the Offshore Wind Turbines is required.

Again, as already mentioned, Inspection and Maintenance of offshore wind farms is more difficult and expensive than the equivalent activities in onshore wind farms. It would be much easier and cheaper for the operator to control all the facts from land. It is from this moment when a Remote Inspection is proposed.

In the following Table (3) there is an overview of the current inspection cycles for the different parts of the offshore turbine facilities [35].

Table 3. Recommended Inspection Cycles (years)

Facility Area	Annual	Intermediate	Extended	Post-Event
Subsea Structure	1	3-5	6-10	As needed
Subsea Equipment	n/a	3-5	n/a	As needed
Above Water Structure and Systems	1	3-5	n/a	As needed
Blades	1	3-5	n/a	As needed

Although the proposed remote inspection is only for the inside of the nacelle, it is interesting to know that there will be an important part outside to take into account.

The above water structure consists of the tower structure on the subsea and the nacelle, where we find the control systems and turbine with all other components.

The annual inspection targets mainly above water activities. The above water structures are examined to evaluate the general conditions noting features such as coating breakdowns, corrosion, and any physical damage present. So, the way to carry out the inspection will be essentially by visual control.

In this chapter how to perform a proper inspection of the inside of the Nacelle is highlighted and specified.

6.2 Inspecting the Nacelle

The nacelle structure integrity should receive primary attention when developing an inspection program due to its importance in the overall system behaviour.

While inspecting the nacelle two different systems are distinguished: the Structural system and the Electrical and Mechanical system.

6.2.1 Structural System

As shown in Table (3), the inspection cycles are divided into: Annual, Intermediate and Additional Inspections.

The most important for this project is the first one. The reason is that it is based basically on a General Visual inspection (GVI). This level includes a visual inspection performed by qualified personnel and documented with a written report including video and photographs. The inspection can be carried out with binoculars or other equipment that provides sufficient detail to identify the most frequent anomalies. These irregularities are described below [36].

1. Corrosion or coating breakdown
2. Damaged or missing members
3. Cracks or indications at welded joints
4. Damaged risers/cables attached to structure
5. Loose bolts
6. Evidence of lateral deflection or lean

6.2.2 Electrical and Mechanical System

The maintenance of turbines, electrical cabling, junction boxes, panels, transformers and generators, hydraulic systems and control systems should be done in accordance with manufacturer recommendations to ensure efficient and safe operations.

The use of remote monitoring systems is strongly recommended to provide regular feedback on equipment function as well as to identify anomalous conditions without the need to have maintenance crews on the facility.

6.3 Companies General Procedure

We have been in touch with three different companies and a global view of the Inspection Procedure has been acquired. Each Company has been useful for a different aspect related to inspecting the nacelle of an Offshore Wind Farm. The companies are described below:

- Taiwan Power Company (Wind Park TaiChung Power Plant).

  http://www.amccentre.nl/harakosanDemo/

The "Taiwan Power Company", was organized on May 1, 1947. The company's major business items are power development, power supply, investment in power and energy technology services, etc.

Figure 24. Taiwan Power Company logo

In particular, this company owns the Wind Park TaiChung Power plant. We had access to information about this wind farm. This information included technical documentation on the components of the nacelle and also other parts of the wind turbine. Probably, the most valuable information has been the maintenance list of the elements inside the nacelle.

This document describes interesting details that are essential for Chapters 7 and 8. Some components dealt with are: First aid box, fire-extinguisher, cable work, bolt connections lubrication of the yaw bearing and motor, lighting and emergency lighting, lifting winch, nacelle control box, glass fibre cables, automatics, wind speed sensor, hydraulic unit, PLC, acceleration sensor, etc.

- Energo Engineering (A Speciality Structural Engineering Company).

  http://www.energoeng.com/index.html

"Energo Engineering" is a speciality engineering firm that solves difficult structural engineering problems for the oil, gas and wind industry. "Energo" recently became a member of the Offshore Operators Committee (OOC).

Figure 25. Energo Engineering logo

From this company, inspection methodologies for Offshore Wind Turbine Facilities have been got. Essentially, the most valuable information has been dealing with the above water structural and access systems and the electrical and mechanical systems.

In addition, we have an Inspection Sample Checklist and Data Reporting. These two documents are in Appendixes E and F.

- Statkraft (A European leader in renewable energy).

  http://www.statkraft.com/

"Statkraft" is Europe's leader in renewable energy. The Group develops and produces hydropower, wind power, gas power and district heating, and is a major player on the European energy exchanges.

Figure 26. Statkraft logo

Øyvind Netland, PhD at Department of Engineering Cybernetics requested some information about visual inspection in their wind turbine nacelles. The result was successful. The got a confidential report with photos during the inspection of a nacelle. Undoubtedly, the photos provide needed information to perform the simulation of a real nacelle.

7

Different possible solutions

7.1 INSPECTION PLAN 45
7.1.1 THE COMPONENT LIST 46
7.2 LAYOUT 46
7.2.1 LIGHTING CONDITIONS 47
7.2.2 ANALYSED SCENARIOS 48
7.3 EXPERIMENTAL CONTROL SYSTEM 48

7.1 Inspection Plan

In this Chapter the components that make up the system are presented. As the inspection will be done with the first prototype, the layout will also change in order to be consequent with the structure of a real nacelle.

Figure 27. Components and items to use during the inspection

7.1.1 The Component List

A list of components has been made to represent some of the most typical scenarios during a real inspection.

The list is shown below:

- Control Panel
- Oil & Grease
- Paper board
- Air tubes
- Bolts & washers
- Electric motor
- Industrial cables

Many of these items are not exactly what you find inside a real nacelle. In spite of this, the elements are adequate enough to the aim of the thesis.

7.2 Layout

Before talking about layout, it is interesting to remember how we wanted to do the inspection.

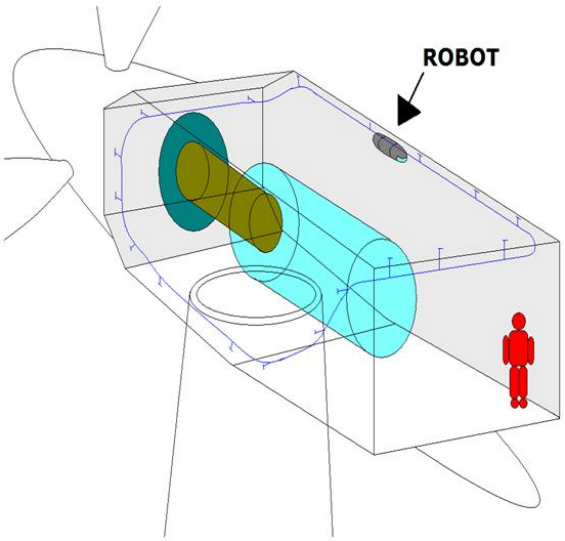

Figure 28. Remote inspection device inside the nacelle of wind turbine

In the picture above, from [37], the first idea about how to do the inspection is shown.

On the other hand, taking into account the decision of using the first prototype instead of the model train, a change on the path of the robot is required.

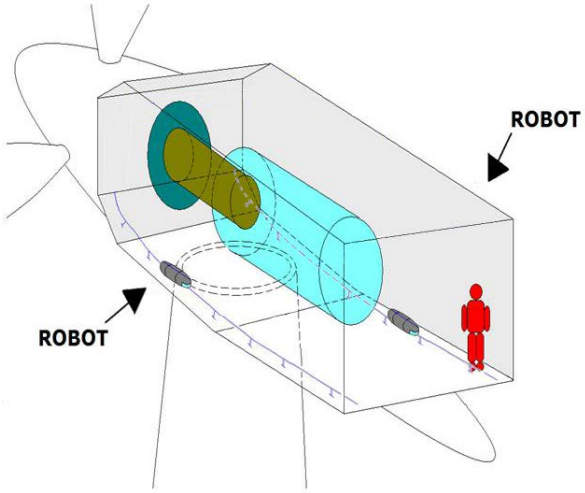

Figure 29. Updated path for inspection testing

In the figure above, the new philosophy for inspection testing is shown. The reason of this change is mainly because of the turning limitation that the first prototype shows.

In our assumption, the rail will be on one side or the other of the main shaft. It is worth recalling again that we are not talking about the layout during operation but during testing on our assembly.

7.2.1 Lighting conditions

The Nacelle Lighting and Power Systems have been found in [38].

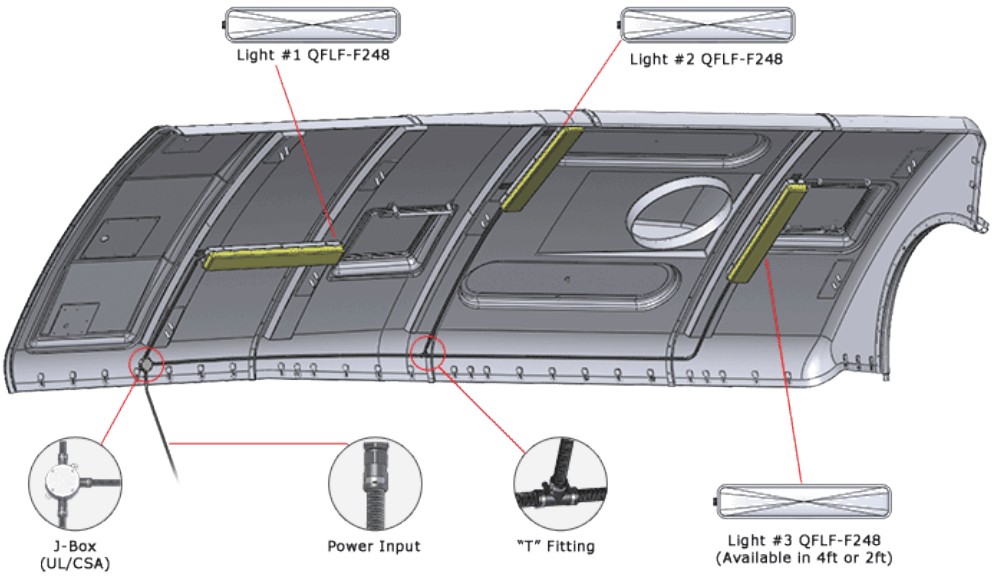

Figure 30. Nacelle lighting and power systems

7.2.2 Analysed scenarios

The nacelle and its components were described in Chapter 4. Now, it is time to implement a simulated scenario using those components shown on the list before.

There are two main possibilities to make the layout. The first idea was to create the whole system at once. This meant that we were trying to reproduce a real nacelle. Although this seems the best idea, because we would be reflecting the reality, this is much more difficult than creating different scenarios.

The aim of this thesis is to demonstrate the capacity of a remote presence to detect problems inside a Nacelle. Creating different scenarios the aim can be reached with the same probability and less effort. This solution, after all, is even more efficient.

The chosen scenes or situations to be represented are shown below:

- Inspecting the bolt connections.
- State of the industrial cables.
- Oil leakages.
- State of the Control Panel (controller).
- Inspecting the electric motor connection.

This set of situations will help us to draw conclusions about the efficiency of the system.

7.3 Experimental Control System

The idea we had at the beginning was to build a big stage, simulating a real nacelle, and to test the Inspection System by another Control System that would control the stage.

This intention lost interest when it was decided not to implement the model train.

In the future, if a new more flexible prototype and with a greater path was built, an Experimental Control System would be a good idea to show Wind Farm Companies how the Remote Presence works.

8

Tests

8.1 INTRODUCTION 49
8.1.1 MODUS OPERANDI 49
8.2 TEST 1. GENERAL SEARCH 50
8.2.1 VISIBLE CABLES 50
8.2.2 OIL LEAKAGE 52
8.3 TEST 2. COMPARISON SEARCH 53
8.3.1 CONTROL PANEL 53
8.3.2 MOTOR CONNECTION 54
8.3.3 BOLTS CONNECTIONS 54

8.1 Introduction

In this Chapter, the experimental part is presented. The robot is going to be tested against different situations that can appear inside a real nacelle.

Before making this demonstration, how a real inspection is carried on has been taken into account. This is the point at which we are trying to demonstrate the efficiency of the remote inspection vs. inspection *in situ*.

8.1.1 Modus operandi

There two ways of making inspection:

On one way, a general search must be performed firstly. This approach can be easily explained as when you do not really know what you are looking for. During this kind of inspection the most important elements and the most striking failures are watched. Elements such as: first aid box, fire-extinguisher, easily visible cables, lighting and emergency lighting, significant oil leakages and grease stains, etc. can be checked.

On the other way, and secondly, a more thorough search is usually done. This inspection is made by comparison in time. This is effective for searching details such as: bolt connections, motor connection, state of the sensors, etc.

8.2 Test 1. General Search

In this point, the inspection is running and it starts with a global search. The images shown below are discussed in detail in the following Chapter 9.

8.2.1 Visible cables

Just glancing around the “nacelle” faulty cable can be detected easily.

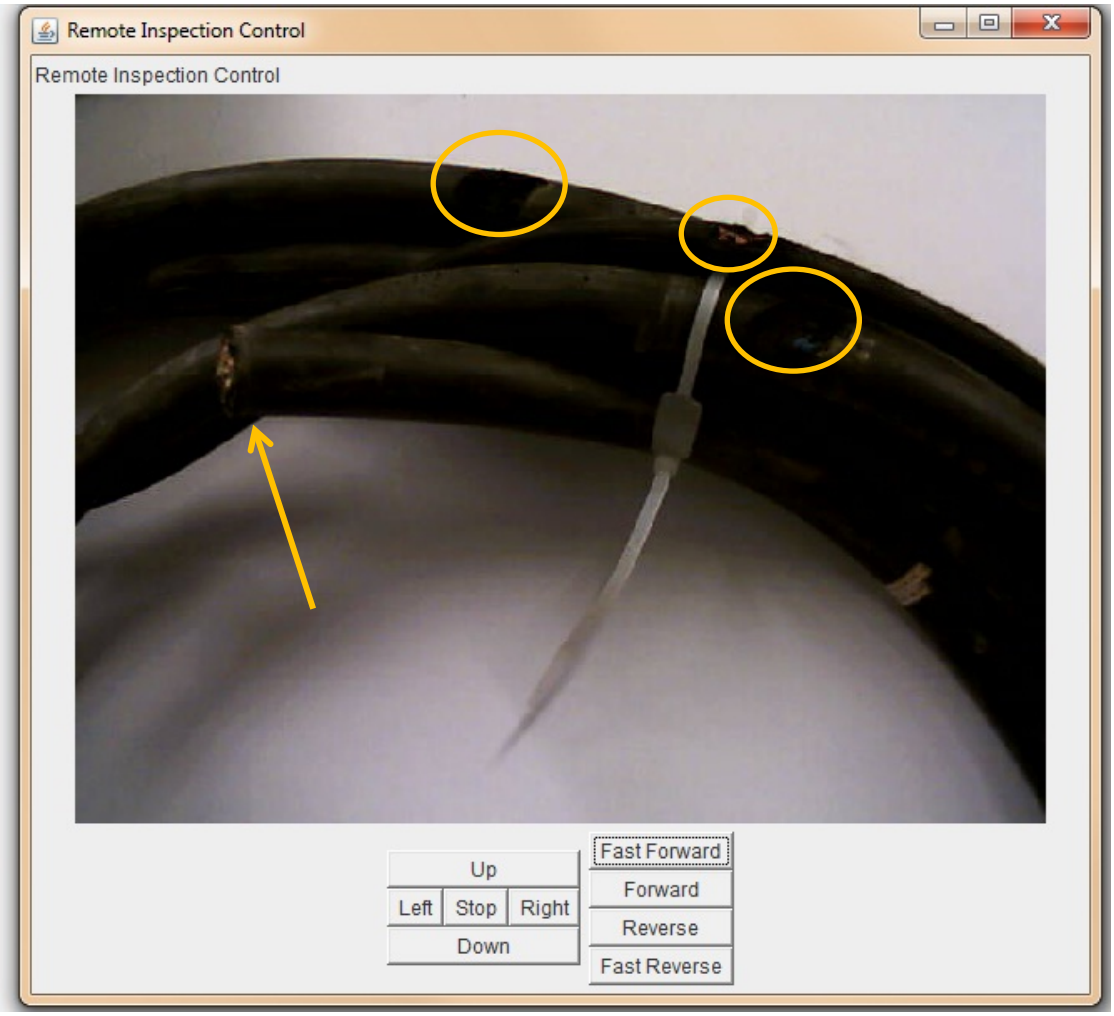

Figure 31. Wear and tear on cables remotely inspected

Wear and tear on cables can be seen in picture above. The yellow arrow is pointing a cut located in the cable.

The following scheme shows the previous situation, but this time the photo has been taken with another camera and from a different angle.

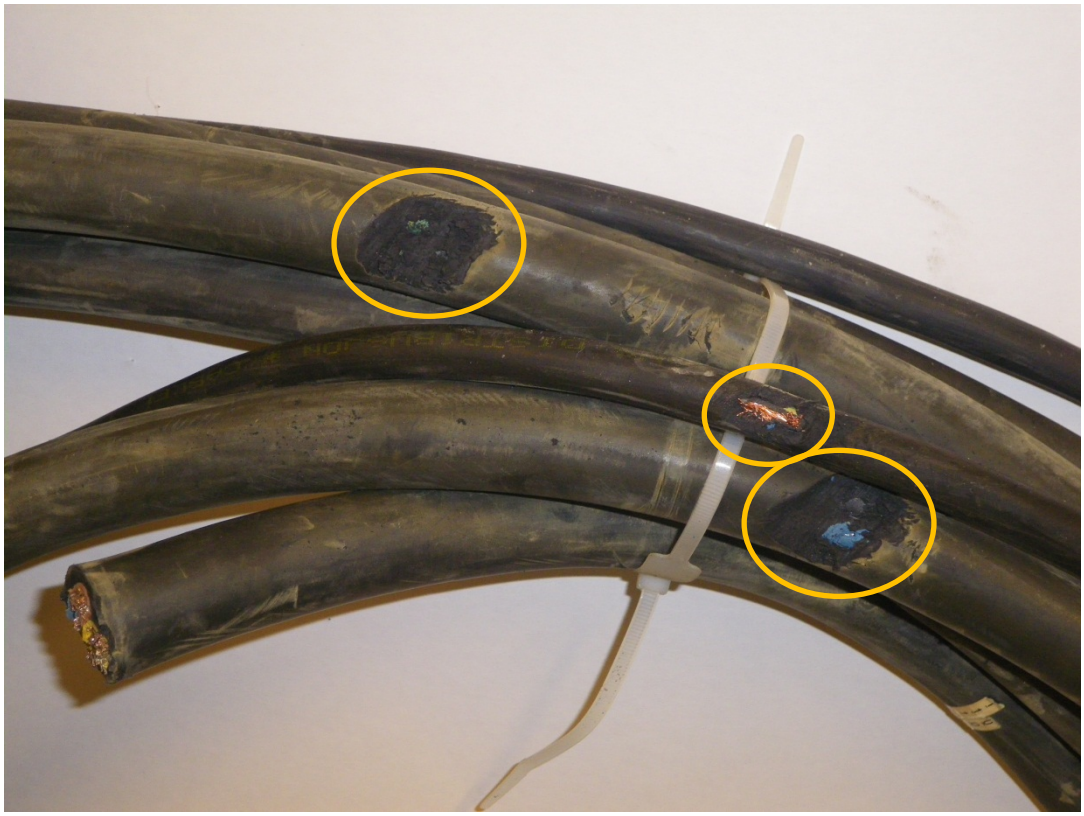

Figure 32. Wear and tear on inspected cables (being there)

Cables defects in this picture are much more evident than in the previous photo. Here, an advantage that the operator has over our remote presence appears.

Looking carefully, we can even see the cooper that stands out from the thinnest wire.

8.2.2 Oil leakage

A very typical accident in the industrial word is an oil leakage.

Figure 33. Oil leakage

The picture above shows one of the numerous oil leakages found during the inspection.

It is very difficult to know where this leakage comes from. It would be easier if we could interact with the surroundings.

8.3 Test 2. Comparison Search

The philosophy of this method is simple: first you take a photo of the component you want to inspect. If after a while any change is observed in the picture, there will be a FAILURE sign. But, if no change at all occurred, it will mean that everything is FINE.

8.3.1 Control panel

In this case we are not talking about defects or mechanical failures. During inspection might be useful to be able to see LED changes, for example in a Control panel.

Figure 34. Control Panel

We will return on this photo in Chapter 9 to explain some ideas to improve the behaviour of our Inspection System.

8.3.2 Motor connection

Here is another example of comparison search. The figure below shows a cable poorly connected to motor.

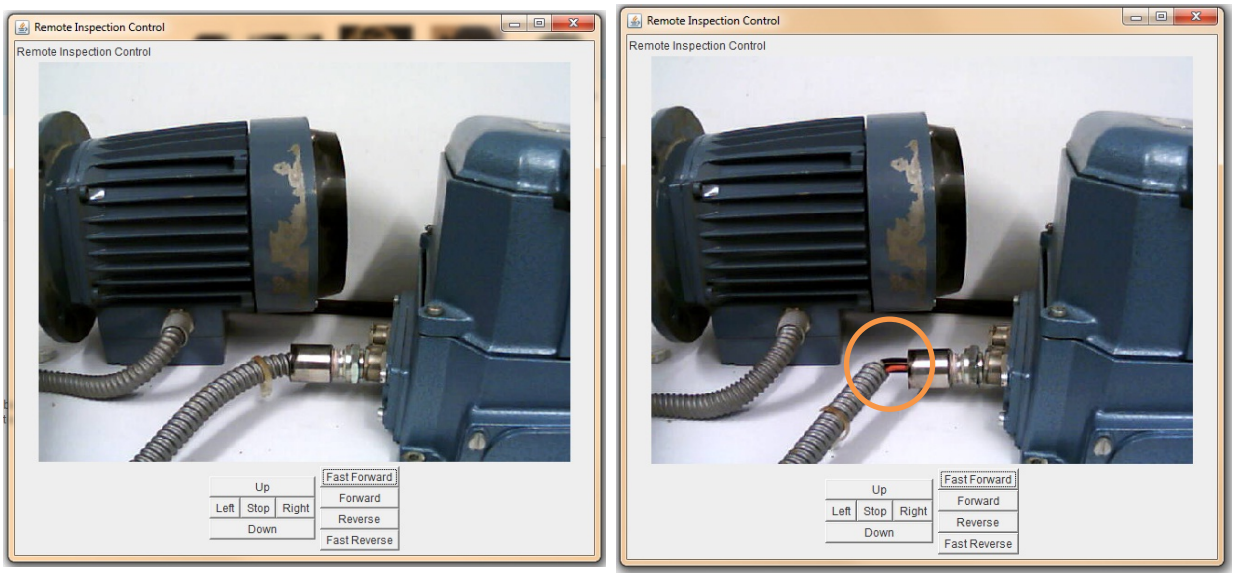

Figure 35. Motor connection

8.3.3 Bolts connections

This is a very clear example of what happens in these systems when time passes by.

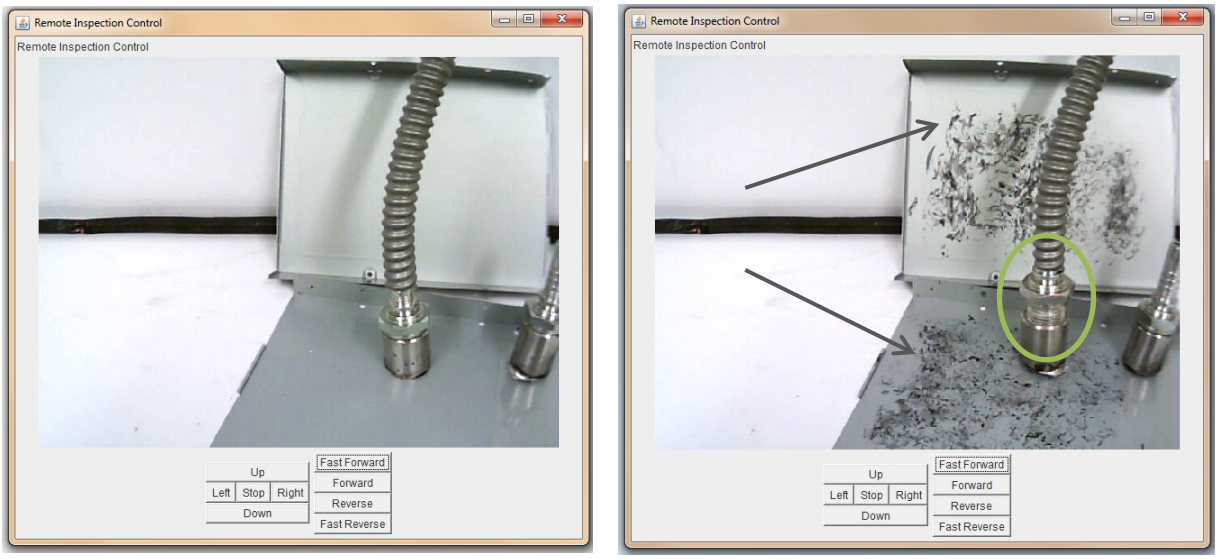

Figure 36. Bolts connections and grease stains

As you can see in the picture on the right the screw is not fastened and the metal sheets are covered in grease.

9 Discussion

9.1 THE INSPECTION CONTROLLER 55
9.2 AUGMENTED REALITY 56
9.3 INSPECTION TESTS 56
9.4 FUTURE WORK 57

Problems, solutions, and other aspects of the project are shown below. The main topics of this thesis are classified in different sections.

9.1 The Inspection Controller

An important change has been made to the device controller. This change has provided the control system two additional features:

- Greater speed of inspection: the fact that the camera moves together with the cart means an improvement on the efficiency of the whole System.
- Extra comfort in controlling: by adding a gamepad an apparently none important aspect has been achieved. At this moment the system can be controlled just looking at the video screen. This way is much more comfortable than the previous one (pressing buttons with the mouse).

Some delays have been detected while moving the camera and have not yet been solve.

The camera is not able to move in diagonal, preventing the movement in two directions simultaneously. Moreover, the IP camera has a limitation (dead angle) of 90 degrees, which can be an important drawback in many cases.

9.2 Augmented Reality

That Augmented Reality is the future of technology is known. The question is if this technology can provide a huge number of benefits for the current system.

Nowadays the prototype is not ready for such a powerful technology. There are limitations, such as those already mentioned and others that will be discussed in the next section, which influences the evolution of the system.

9.3 Inspection TESTS

For testing the inspection system, the first made decision was to choose the prototype that was going to be used. The best solution would have probably been to use a model train, what should have made the inspection plan much more versatile. However, there have been some problems that have delayed the assembly of the train path. So, the second prototype has been used to make the "experiments".

Now TESTS done in Chapter 8 will be revised and commented.

- Condition of cables: Figure 31 shows wear and tear cables. The quality of the photo is not really good comparing to Figure 32. This tries to be an example of the difference between the remote presence and *in situ* presence. An extendable arm to reach higher places is required. Also would be a good idea to have some tools, attached to the robot, to interact with the components inside the nacelle. If this is not implemented, only the visible side of the cables will be able to be inspected rather than the hidden part.
- Oil leakage: this frequent accident is shown in Figure 33. With our prototype is possible to detect this kind of failure. Oil leakages are normally on the floor, walls or also near the bearings.
- Control panel sensor: in Figure 34 a LED is turned ON and it is detected by the camera. There is no difference between what the robot watches than what an inspector would be able to watch being there.
- Motor connection: here is another example of change in the time. The Figure 35 shows perfectly the cable poorly connected to the motor.
- Bolts connections: Continuing with comparison of the same two things at different moment, in Figure 36 a not fastened bolt is shown. Moreover, the image also shows that grease stains have appeared around it.

There have been no “experiments” for detecting: sound, vibration, or heat, simply because the robot will not detect these phenomenons. Anyway, having been able to test all the things above, other cases can be easily extrapolated. The more sensors available, the easier will be to detect different type of problems.

The remote inspection will be very useful and necessary. Continuing researching in this field will be always something profitable.

9.4 Future Work

To take more benefit from the Remote Presence several aspects should first of all be improved.

Talking about mechanic systems, anything that increases the robot’s line of vision must be added. This refers to a kind of lift for the camera. This would provide multiple angles of vision.

To have a kind of robotic arm could be another mechanical improvement to be developed. This arm would not be used for maintenance. Its role would be exclusively to interact with the environment for a better inspection.

Regarding to the sensors, a thermal camera and a microphone should be implemented in the system.

Something very important is to have better feedback from Wind Farm Companies. There is a lot of confidential material that cannot be used. Dealing with a few companies and trying to reach some cooperation agreement should be the future priority. As, only in this way, all positive developments in the system would be certified inside real nacelles.

Finally, it would be really interesting to learn more about augmented reality; real application of this technology on our Inspecting System.

10 Conclusion

The first that has been done has been to place a Remote Presence among the Wind Technology current tendency. Due to the importance of wind power in the future and taking into account the lasts designs for reduced maintenance on offshore wind farms, the development of cheap and reliable inspection systems is required.

A new device controller has been proposed. This device provides a great amount of different configurations to the Input System. Future updates or new input devices can be attached to this model. For the moment, a gamepad has been implemented successfully as the current input device. The java code made by Viktor has been changed in some parts, such as: vehicle controller and cam controller.

A huge investigation about components inside an offshore wind turbine nacelle has been done. Also, by comparison with three different companies, and based on the first robot prototype built, an Inspection Procedure has been proposed.

Finally, the most important task, which lies in practical “experiments” on the efficiency of the system have been done. In conclusion, the idea of detecting some specific and tabulated failures has been achieved. The system is good enough to be controlled remotely and to have a look of its surroundings. Moreover, the system allows the operator to observe changes with sufficient accuracy, at least having a distance shorter than two meters between the object and the camera.

So, these “experiments” have proved that Remote Inspection is possible and can be feasible. Although it is an obvious statement, it must be said that the greater number of sensors in a prototype the better knowledge of the surroundings we will get.

Part II.

Appendix

A

Map of Wind Farms in SCANDINAVIA


SCANDINAVIA
Wind Farms and Ocean Energy
Researched, Compiled and Produced by La Tene Maps
SSE Renewables
www.sserenewables.com
Right Technology Right Time
AQUARET
A European Project Providing Free Educational Tools and Materials on Aquatic Renewable Energy Technologies
www.aquaret.com
Scandinavian Companies with Ocean Energy Projects
Other Maps with Wind Farm Coverage
Other Maps with Ocean Energy Coverage
wind power data
www.windpowerdata.com
ASK RÅDGIVNING
Powerful competence within energy and environment
www.askradgivning.no
NORWEGIAN SEA
FINNMARK
TROMS
LAPIN LÄÄNI
NORDLAND
NORRBOTTEN
VÄSTERBOTTEN
OULUN LÄÄNI
SWEDEN
NORD-TRØNDELAG
VÄSTERNORRLAND
GULF OF BOTHNIA
SØR-TRØNDELAG
JÄMTLAND
ITÄ-SUOMEN LÄÄNI
FINLAND
MØRE OG ROMSDAL
LÄNSI-SUOMEN LÄÄNI
GÄVLEBORG
SOGN AND FJORDANE
OPPLAND
NORWAY
HEDMARK
KOPPARBERGS
ETELÄ SUOMEN LÄÄNI
BUSKERUD
NORDALAND
UPPSALA
GULF OF FINLAND
AKERSHUS
VÄRMLAND
VÄSTMANLAND
STOCKHOLM
TELEMARK
ØSTFOLD
ÖREBRO
SÖDERMANLAND
ROGALAND
AUST-AGDER
VEST-AGDER
ÖSTERGÖTALAND
SKAGERRAK
VÄSTRA GÖTALANDS
JÖNKÖPING
KALMAR
GOTLAND
KATTEGAT
HALLAND
BLEKINGE
KRONEBERG
ÖLAND
BALTIC SEA
Small Wind Farms or Single Turbines in Sweden
Legend
Wind.
It means the world to us.
Vestas
Wind Farm Civils

B

Joysticks from Genge & Thoma

C

JXInput – Input Devices for Java

Joystick Listener.java

```
package com.centralnexus.input;

/**
 * The listener interface for receiving joystick events. A Joystick
 * will periodically notify the implementor of this listener of changes
 * to the attached joystick. It is <b>not</b> neccessary to call the
 * poll function on the joystick when this interface is implemented.
 *
 * @see Joystick
 * @author George Rhoten
 * @author Ed Burns <edburns@acm.org>
 * @since July 8, 2001
 */

public interface JoystickListener
{
    /**
     * Implement this function to get periodically notified that
     * a joystick changed one of its axis values.
     *
     * @param j The joystick that was recently polled.
     */
    public void joystickAxisChanged(Joystick j);

    /**
     * Implement this function to get periodically notified that
     * a joystick button changed one of its values.
     *
     * @param j The joystick that was recently polled.
     */
    public void joystickButtonChanged(Joystick j);
}
```

Source Code 1. The JoystickListener class

Joystick Notifier.java

```
package com.centralnexus.input;

/**
 * Notifies the joystick listeners of Joystick events.
 * Only one of these notifiers should be created per joystick.
 *
 * @author George Rhoten
 * @since July 8, 2001
 */
class JoystickNotifier implements Runnable
{
    /** A joystick used for polling. */
    private Joystick jstick;

    /** The thread used to poll one joystick. */
    private Thread pollingThread = null;

    /** polling interval for this joystick */
    private int interval = 50;

    /** Listeners for this joystick */
    private JoystickListener joyListeners[] = new JoystickListener[0];

    /**
     * Joysticks for the listeners. Multiple joysticks may be created,
     * but they all have the same data values. Keeping this array
     * makes it easier to use the operator== in Java.
     */
    private Joystick joysticks[] = new Joystick[0];

    /** axis values for all axis values on all joysticks with the same ID. */
    float axisValues[] = new float[Joystick.AXIS_TOTAL];

    /** Old axis values for all axis values. */
    private float axisValuesOld[] = new float[Joystick.AXIS_TOTAL];

    /** The cache of current button values */
    int buttonValues = 0;

    /** The Old cache of current button values */
    private int buttonValuesOld = 0;

    JoystickNotifier(Joystick js) {
        jstick = js;
    }

    /** Start polling the joystick. */
    public void start() {
        pollingThread.start();
    }

    /**
     * Add a listener to this joystick
     */
    public synchronized void addJoystickListener(Joystick j, JoystickListener l) {
        if (l != null) {
            JoystickListener newListeners[] = new JoystickListener[joyListeners.length +
            1];
            Joystick newJoystickArr[] = new Joystick[joyListeners.length + 1];

            System.arraycopy(joyListeners, 0, newListeners, 0, joyListeners.length);
            System.arraycopy(joysticks, 0, newJoystickArr, 0, joysticks.length);
```

```
            newListeners[joyListeners.length] = l;
            newJoystickArr[joysticks.length] = j;

            joyListeners = newListeners;
            joysticks = newJoystickArr;

//            System.out.println("adding " + j);
            // This new listener needs to now the current state.
            joyListeners[joyListeners.length -
            1].joystickAxisChanged(joysticks[joyListeners.length - 1]);
            joyListeners[joyListeners.length -
            1].joystickButtonChanged(joysticks[joyListeners.length - 1]);

            if (pollingThread == null) {
                pollingThread = new Thread(this);
                pollingThread.start();
//                System.out.println("starting");
            }
        }
    }

    /**
     * Remove a listener to this joystick
     */
    public synchronized void removeJoystickListener(Joystick j, JoystickListener l) {
        if (l != null && joyListeners.length > 0) {
            JoystickListener newListeners[] = new JoystickListener[joyListeners.length -
            1];
            Joystick newJoystickArr[] = new Joystick[joyListeners.length - 1];
            int idx;

            for (idx = 0; idx < newListeners.length
             && joyListeners[idx] != l && joysticks[idx] != j; idx++)
            {
                newListeners[idx] = joyListeners[idx];
                newJoystickArr[idx] = joysticks[idx];
            }
//            System.out.println(idx + " removed");
            if (idx < joyListeners.length) {
                if (0 < newListeners.length) {
                    System.arraycopy(joyListeners, idx + 1, newListeners, idx,
newListeners.length - idx);
                    System.arraycopy(joysticks, idx + 1, newJoystickArr, idx,
newJoystickArr.length - idx);
                }
                joyListeners = newListeners;
                joysticks = newJoystickArr;
            }
        }
    }

    public synchronized final void notifyJoystickListeners() {
        jstick.poll();

        boolean axisChanged = false;
        for (int idx = 0; idx < axisValues.length; idx++) {
            if (axisValues[idx] != axisValuesOld[idx]) {
                axisChanged = true;
                break;
            }
        }

        for (int listenNum = joyListeners.length - 1; listenNum >= 0; listenNum--)
        {
            if (axisChanged) {
```

```
//                System.out.println(jstick + " " + listenNum + " axis changed");
                joyListeners[listenNum].joystickAxisChanged(joysticks[listenNum]);
            }
            if (buttonValues != buttonValuesOld) {
//                System.out.println(jstick + " " + listenNum +" button changed");
                joyListeners[listenNum].joystickButtonChanged(joysticks[listenNum]);
            }
        }
        buttonValuesOld = buttonValues;
        System.arraycopy(axisValues, 0, axisValuesOld, 0, axisValues.length);
    }

    public void setPollInterval(int pollMillis) {
        interval = pollMillis;
    }

    public int getPollInterval() {
        return interval;
    }

    public void run()
    {
        while (joyListeners.length > 0) {
            notifyJoystickListeners();
            //System.out.println(jstick.toString());

            try {
                Thread.sleep(interval);
            }
            catch (InterruptedException e) {
                pollingThread = null;
                return;
            }
        }
        pollingThread = null;
    }
}
```

Source Code 2. The JoystickNotifier class

Joystick.java

```
package com.centralnexus.input;

import java.io.IOException;

/**
 * <p>
 * Device driver to a Windows joystick.
 * This handles at least an x,y motion joystick. A joystick can be plugged
 * in as any id, and the first joystick id may or may not be plugged in.
 * In order to create a Joystick, you must use one of createInstance functions.
 * </p>
 * <p>
 * There are two ways to update the axis and button values. You can either:
 * <ol>
 *   <li>Add a JoystickListener to the joystick with addJoystickListener().
 *   </li>
 *   <li>Use the poll() function from you own thread.
 *   </li>
 * </ol>
 * </p>
 *
 * @see JoystickListener
 * @author George Rhoten
 * @since June 10, 2000
 */
public class Joystick {

// Joystick driver capabilites
// These are equivalent to JOYCAPS in windows.

    /** Does this joystick have a z-axis capability? */
    public static final int HAS_Z       = 0x0001;

    /** Does this joystick have a r-axis capability? */
    public static final int HAS_R       = 0x0002;

    /** Does this joystick have a u-axis capability? */
    public static final int HAS_U       = 0x0004;

    /** Does this joystick have a v-axis capability? */
    public static final int HAS_V       = 0x0008;

    /** Does this joystick have a point-of-view control capability? */
    public static final int HAS_POV     = 0x0010;

    /**
     * Does this joystick point-of-view support discrete values
     * capability (centered, forward, backward, left, and right)?
     */
    public static final int HAS_POV4DIR = 0x0020;

    /**
     * Does this joystick point-of-view support continuous degree bearings
     * capability?
     */
    public static final int HAS_POVCONT = 0x0040;

    /**
     * Point-of-view hat is in the neutral position. The value -1 means the
     * point-of-view hat has no angle to report.
     */
    public static final float POV_CENTERED     = -1.0f/100.0f;
```

```
    /**
     * Point-of-view hat is pressed forward. The value 0 represents an
     * orientation of 0.00 degrees (straight ahead).
     */
    public static final float POV_FORWARD      = 0.0f;

    /**
     * Point-of-view hat is pressed to the right. The value 9,000 represents
     * an orientation of 90.00 degrees (to the right).
     */
    public static final float POV_RIGHT        = 90.00f;

    /**
     * Point-of-view hat is pressed backward. The value 18,000 represents
     * an orientation of 180.00 degrees (to the rear).
     */
    public static final float POV_BACKWARD     = 180.00f;

    /**
     * Point-of-view hat is being pressed to the left. The value 27,000
     * represents an orientation of 270.00 degrees (90.00 degrees to the left).
     */
    public static final float POV_LEFT         = 270.00f;

    /** These are the standard buttons. */
    public static final int BUTTON1 = 0x0001;
    public static final int BUTTON2 = 0x0002;
    public static final int BUTTON3 = 0x0004;
    public static final int BUTTON4 = 0x0008;   // Last button
    /** These are the extended buttons */
    public static final int BUTTON5  = 0x00000010;
    public static final int BUTTON6  = 0x00000020;
    public static final int BUTTON7  = 0x00000040;
    public static final int BUTTON8  = 0x00000080;
    public static final int BUTTON9  = 0x00000100;
    public static final int BUTTON10 = 0x00000200;
    public static final int BUTTON11 = 0x00000400;
    public static final int BUTTON12 = 0x00000800;
    public static final int BUTTON13 = 0x00001000;
    public static final int BUTTON14 = 0x00002000;
    public static final int BUTTON15 = 0x00004000;
    public static final int BUTTON16 = 0x00008000;
    public static final int BUTTON17 = 0x00010000;
    public static final int BUTTON18 = 0x00020000;
    public static final int BUTTON19 = 0x00040000;
    public static final int BUTTON20 = 0x00080000;
    public static final int BUTTON21 = 0x00100000;
    public static final int BUTTON22 = 0x00200000;
    public static final int BUTTON23 = 0x00400000;
    public static final int BUTTON24 = 0x00800000;
    public static final int BUTTON25 = 0x01000000;
    public static final int BUTTON26 = 0x02000000;
    public static final int BUTTON27 = 0x04000000;
    public static final int BUTTON28 = 0x08000000;
    public static final int BUTTON29 = 0x10000000;
    public static final int BUTTON30 = 0x20000000;
    public static final int BUTTON31 = 0x40000000;
    public static final int BUTTON32 = 0x80000000;

/*  //Not known to be used.  Maybe used for a joystick listener.
    public static final int BUTTON1CHG = 0x0100;
    public static final int BUTTON2CHG = 0x0200;
    public static final int BUTTON3CHG = 0x0400;
    public static final int BUTTON4CHG = 0x0800;
*/
```

```
/** Constant for axisValues */
static final int AXIS_TOTAL = 7;
/** Constant for axisValues */
private static final int AXIS_X     = 0;
/** Constant for axisValues */
private static final int AXIS_Y     = 1;
/** Constant for axisValues */
private static final int AXIS_Z     = 2;
/** Constant for axisValues */
private static final int AXIS_R     = 3;
/** Constant for axisValues */
private static final int AXIS_U     = 4;
/** Constant for axisValues */
private static final int AXIS_V     = 5;
/** Constant for axisValues */
private static final int AXIS_POV   = AXIS_TOTAL - 1;

/** Singleton JoystickNotifiers that creates joystick events */
private static JoystickNotifier joyNotifier[];

/** The joystick id. Typically 0 or 1 */
private int id = 0;
/** The joystick id. Typically 0 or 1 */
private float deadZone = 0.0f;
/** The cache of capabilities bits */
private int capabilities = 0;
/** The cache of current axis values */
float axisValues[];
/** This is who notifies me */
JoystickNotifier myJoyNotifier;

static {
    try {
        new jjstick();
        joyNotifier = new JoystickNotifier[getNumDevices()];
    }
    catch (UnsatisfiedLinkError e) {
        e.fillInStackTrace();
        throw e;
    }
}

/**
 * Returns the number of joysticks supported by the joystick driver or zero
 * when no driver is present.
 */
public native static final int getNumDevices();

/**
 * Returns true when the joystick is plugged into the computer, false
 * otherwise.  This function may do some initialization to get the
 * joystick working.
 * @param id The ID of the joystick where 0 <= id < getNumDevs().
 */
public native synchronized static boolean isPluggedIn(int id);

native synchronized static int poll(int id, float axisValues[]);
private native static int getCapabilities(int id);

private native static int getNumButtons(int id);
private native static int getNumAxes(int id);
private native static String toString(int id);

Joystick(int id) throws IOException {
    if (isPluggedIn(id)) {  // Must be called before anything else!
```

```
            this.id = id;
            if (joyNotifier[id] == null)
            {
                joyNotifier[id] = new JoystickNotifier(this);
            }
            myJoyNotifier = joyNotifier[id];
            axisValues = joyNotifier[id].axisValues;
            capabilities = getCapabilities(id);
            myJoyNotifier.buttonValues = poll(id, axisValues);
        }
        else {
            throw new IOException("Invalid joystick ID: " + id);
        }
    }

    /**
     * Start using the first available joystick.
     * @throws IOException Thrown when a joystick
     *          is not plugged into the computer.
     */
    public static Joystick createInstance() throws IOException {
        int maxID = getNumDevices();
        int id = 0;

        while (id < maxID && !isPluggedIn(id)) {
            id++;
        }
        if (id >= maxID) {
            throw new IOException("Joystick not found.");
        }
        return createInstance(id);
    }

    /**
     * Start using a joystick with a specific id. This should be used when
     * you know a specific joystick is plugged into the computer.
     *
     * @param id The joystick id to get joystick information from.
     * @throws IOException Thrown when the joystick for the id
     *          is not plugged into the computer.
     */
    public static Joystick createInstance(int id) throws IOException {
        if (id < 0 || getNumDevices() <= id) {
            throw new IOException("Invalid joystick ID: " + id);
        }
        int capabilities = getCapabilities(id);
        Joystick newJoystick;
        if (capabilities == 0 || capabilities == HAS_Z) {
            newJoystick = new Joystick(id);
        }
        else {
            newJoystick = new ExtendedJoystick(id);
        }
        return newJoystick;
    }

    /**
     * The joystick id for the joystick connected to the computer.
     * The ID numbers have a range of 0 <= id < getNumDevices()
     * @see #getNumDevices()
     */
    public final int getID() {
        return id;
    }
```

```
/**
 * This polls (updates) the joystick for its values. This must be called
 * after the owner is done with the old values.
 */
public void poll() {
    myJoyNotifier.buttonValues = poll(id, axisValues);
}

/**
 * Get the Capability bits. These bits can be ORed together.
 * @see #getCapability(int)
 */
public int getCapabilities() {
    return capabilities;
}

/**
 * Is a certain capability bit turned on?
 * @see #HAS_Z
 * @see #HAS_R
 * @see #HAS_U
 * @see #HAS_V
 * @see #HAS_POV
 * @see #HAS_POV4DIR
 * @see #HAS_POVCONT
 */
public final boolean getCapability(int capability) {
    return (capabilities & capability) == capability;
}

/**
 * Returns the axis value, or 0 if the axis value is inside the deadZone.
 */
final float removeDeadZone(float axis) {
    return (axis <= -deadZone || deadZone <= axis) ? axis : 0.0f;
}

/**
 * The x value of a joystick has a range from -1 to 1.
 */
public synchronized float getX() {
    return removeDeadZone(axisValues[AXIS_X]);
}

/**
 * The y value of a joystick has a range from -1 to 1.
 */
public synchronized float getY() {
    return removeDeadZone(axisValues[AXIS_Y]);
}

/**
 * The z value of a joystick has a range from -1 to 1.
 * @see #getCapabilities()
 */
public synchronized float getZ() {
    if ((capabilities & HAS_Z) == HAS_Z) {
        return removeDeadZone(axisValues[AXIS_Z]);
    }
    return 0.0f;
}

/**
 * The r value of a joystick has a range from -1 to 1.
 * @return the rudder value (4th axis of movement)
```

```
     * @see #getCapabilities()
     */
    public synchronized float getR() {
        if ((capabilities & HAS_R) == HAS_R) {
            return removeDeadZone(axisValues[AXIS_R]);
        }
        return 0.0f;
    }

    /**
     * The u value of a joystick has a range from -1 to 1.
     * @return the u value (5th axis of movement)
     * @see #getCapabilities()
     */
    public synchronized float getU() {
        if ((capabilities & HAS_U) == HAS_U) {
            return removeDeadZone(axisValues[AXIS_U]);
        }
        return 0.0f;
    }

    /**
     * The v value of a joystick has a range from -1 to 1.
     * @return the v value (6th axis of movement)
     * @see #getCapabilities()
     */
    public synchronized float getV() {
        if ((capabilities & HAS_V) == HAS_V) {
            return removeDeadZone(axisValues[AXIS_V]);
        }
        return 0.0f;
    }

    /**
     * Current position of the point-of-view control. Values for this member
     * are in the range 0 through 359.00. These values represent the angle,
     * in degrees.
     * @return the point of view
     * @see #getCapabilities()
     */
    public synchronized float getPOV() {
        if ((capabilities & HAS_POV) == HAS_POV) {
            return axisValues[AXIS_POV];
        }
        return 0;
    }

    /**
     * Current state of joystick buttons. To see which buttons are pressed,
     * "&" the result with one of the BUTTON constants.
     * @return the bits representing each button.
     * @see Joystick#BUTTON1
     * @see Joystick#BUTTON2
     * @see Joystick#BUTTON3
     * @see Joystick#BUTTON4
     */
    public int getButtons() {
        return myJoyNotifier.buttonValues;
    }

    /**
     * Current state of a specific joystick button.
     * @param button can be BUTTON1, BUTTON2, BUTTON3 and so on.
     * @return true if the button is being pressed, false otherwise.
     * @see Joystick#BUTTON1
```

```
     * @see Joystick#BUTTON2
     * @see Joystick#BUTTON3
     * @see Joystick#BUTTON4
     */
    public boolean isButtonDown(int button) {
        return (myJoyNotifier.buttonValues & button) == button;
    }

    /** Number of buttons on the joystick. */
    public int getNumButtons() {
        return getNumButtons(id);
    }

    /** Number of axes currently in use by the joystick. */
    public int getNumAxes() {
        return getNumAxes(id);
    }

    /** Size of the dead zone. The default value is 0.0. */
    public final float getDeadZone() {
        return deadZone;
    }

    /**
     * Size of the dead zone. The dead zone is the range of values of each
     * axis that returns 0. For example, when the deadZone = 0.1 and
     * joystick(x, y, z) = (-0.09, 0.5, 0.1), then
     * joystick(x, y, z) = (0.0, 0.5, 0.0).
     * @throws IllegalArgumentException when deadZone is out of the range
     *          0 <= deadZone <= 1.0.
     */
    public final void setDeadZone(float deadZoneVal) {
        if (0.0f <= deadZoneVal && deadZoneVal <= 1.0f) {
            deadZone = deadZoneVal;
        }
        else {
            throw new IllegalArgumentException("Dead Zone needs to be between 0 and 1: "
             + deadZoneVal);
        }
    }

    /**
     * Synonym for setDeadZone(float)
     */
    public final void setDeadZone(double deadZoneVal) {
        setDeadZone((float)deadZoneVal);
    }

    /**
     * Adds the specified joystick listener to receive joystick events
     * from this joystick. If l is null, no exception is thrown
     * and no action is performed.
     * @param l The joystick listener
     * @param notifyOnChangeOnly Notify the listener of changes only.
     *        Setting this parameter to false will poll the joystick
     *        at a regular interval.
     */
    public void addJoystickListener(JoystickListener l) {
        myJoyNotifier.addJoystickListener(this, l);
    }

    /**
     * Removes the specified joystick listener so that it no longer
     * receives joystick events from this joystick. If l is null,
     * no exception is thrown and no action is performed.
```

```
     * @param l The joystick listener
     */
    public void removeJoystickListener(JoystickListener l) {
        myJoyNotifier.removeJoystickListener(this, l);
    }

    /**
     * Set the time in milliseconds that the JoystickListeners get notified of
     * this joystick events.
     */
    public void setPollInterval(int pollMillis) {
        myJoyNotifier.setPollInterval(pollMillis);
    }

    /**
     * Get the time in milliseconds that the JoystickListeners get notified of
     * this joystick events.
     */
    public int getPollInterval() {
        return myJoyNotifier.getPollInterval();
    }

    /** Text description of this joystick without the axis values */
    public String toString() {
        return toString(id) + " [id=" + id + "]";
    }
}

/**
 * This is used only for JDKs that only allow you to load a library
 * with the same class name.
 */
class jjstick {
    static {
        try {
            System.loadLibrary("jjstick");
        }
        catch (UnsatisfiedLinkError e) {
            e.fillInStackTrace();
            throw e;
        }
    }
}
```

Source Code 3. The Joystick class

Joystick TEST.java

```
package com.centralnexus.test;

import java.awt.*;
import java.awt.event.*;
import java.io.IOException;

import com.centralnexus.input.*;

public class JoystickTest
extends Frame
implements Runnable, JoystickListener, ActionListener

{
   Joystick joy;

   private int interval;

   Thread thread = new Thread(this);

   Label buttonLabel = new Label(),
         button2Label = new Label(),
         deadZoneLabel = new Label(),
         yLabel = new Label(),
         zLabel = new Label(),
         rLabel = new Label();
   Label xyLabel = new Label();
   Label intervalLabel = new Label();

   Button addButton = new Button("Add Listener");
   Button removeButton = new Button("Remove Listener");

   JoystickTest() throws IOException {
       super();

       joy = Joystick.createInstance();
       for (int idx = joy.getID() + 1; idx < Joystick.getNumDevices(); idx++) {
           if (Joystick.isPluggedIn(idx)) {
           }
       }
       doWindowLayout();
   }

   JoystickTest(int joystickID) throws IOException {
       super();

       joy = Joystick.createInstance(joystickID);
       doWindowLayout();
   }

   private void doWindowLayout() {
       addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e) {
               dispose();
               System.exit(0);
           }
       });
       setTitle("Joystick Test");

       setLayout(new GridLayout(14, 2));
       add(new Label("Number Of Devices: ", Label.RIGHT));
       add(new Label(Integer.toString(Joystick.getNumDevices())));
       add(new Label("Joystick ID: ", Label.RIGHT));
```

```
        add(new Label("joy(" + Integer.toString(joy.getID()) + ")"));
        add(new Label("Description joy#1: ", Label.RIGHT));
        add(new Label(joy.toString()));
        add(new Label("Capabilities:", Label.RIGHT));
        add(new Label("joy(0x" + Integer.toHexString(joy.getCapabilities()) + ")"));
        add(new Label("Axes: ", Label.RIGHT));
        add(new Label("joy(" + Integer.toString(joy.getNumAxes()) + ")"));
        add(new Label("Buttons: ", Label.RIGHT));
        add(new Label("joy(" + Integer.toString(joy.getNumButtons()) + ")"));
        add(new Label("Dead Zone Size: ", Label.RIGHT));
        add(deadZoneLabel);
        add(new Label("Buttons Pressed: 0x", Label.RIGHT));
        add(buttonLabel);

        add(new Label("Robot Movement: ", Label.RIGHT));
        add(yLabel);
        add(new Label("Camera (UP/DOWN): ", Label.RIGHT));
        add(zLabel);
        add(new Label("Camera (RIGHT/LEFT): ", Label.RIGHT));
        add(rLabel);

        add(new Label("Polling interval: ", Label.RIGHT));
        add(intervalLabel);
        addButton.addActionListener(this);
        removeButton.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == addButton) {
            joy.addJoystickListener(this);
        }
        else {
            joy.removeJoystickListener(this);
        }
    }

    /**
     * This is used by the internal thread.
     */
    public void run() {
        for (;;) {
            joy.poll();

            updateFieldsEx(joy);
            try {
                Thread.sleep(interval);
            } catch(InterruptedException e) {
                break;
            }
        }
    }

    public void joystickAxisChanged(Joystick j) {
        if (j == joy) {
            updateFieldsEx(j);
        }
        else {
            updateFields(j);
        }
    }

    public void joystickButtonChanged(Joystick j) {
        if (j == joy) {
            updateFieldsEx(j);
        }
```

```
        else {
            updateFields(j);
        }
    }

    public void setPollInterval(int pollMillis) {
        interval = pollMillis;
        joy.setPollInterval(pollMillis);
        intervalLabel.setText(Integer.toString(interval));
    }

    public void updateFields(Joystick joystick) {
        button2Label.setText(Integer.toHexString(joystick.getButtons()));
        xyLabel.setText(joystick.getX() + ", " + joystick.getY());
    }

    public void updateFieldsEx(Joystick joystick) {
        buttonLabel.setText(Integer.toHexString(joystick.getButtons()));
        yLabel.setText(Double.toString(joystick.getY()));
        zLabel.setText(Double.toString(joystick.getZ()));
        rLabel.setText(Double.toString(joystick.getR()));
    }

    public void startPolling() {
        thread.start();
    }

    public void addListeners() {
        add(addButton);
        add(removeButton);
        joy.addJoystickListener(this);
    }

    public void setDeadZone(double deadZone) {
        joy.setDeadZone(deadZone);
        updateDeadZone();
    }


    public void updateDeadZone() {
        deadZoneLabel.setText("joy(" + joy.getDeadZone() + ")");
    }

    private static void help() {
        System.out.println("Help:");
        System.out.println(" -h This help screen info");
        System.out.println(" -v Verbose Joystick debug information");
        System.out.println(" -j:n Set the Joystick ID to test (n is an integer)");
        System.out.println(" -j2:n Set the second joystick ID to test (n is an integer)");
        System.out.println(" -d:n Set the dead zone size of the Joystick (n is a real
                           number)");
        System.out.println(" -d2:n Set the dead zone size of the second Joystick (n is a
                           real number)");
    }

    public static void main(String args[]) {
        // This first and last one are never there, but this is for internal testing.
        // They should ALWAYS be false.
        try {
           JoystickTest mainFrame;
            int joystickNum = -1;
            int joystickNumEx = -1;
            double deadZone = -1.0;
            int interval = 50;
```

```
            for (int idx = 0; idx < args.length; idx++) {
                if (args[idx].startsWith("-d:")) {
                    deadZone =
                        Double.valueOf(args[idx].substring(3, args[idx].length()))
                        .doubleValue();
                }
                else if (args[idx].startsWith("-i:")) {
                    interval =
                        Integer.valueOf(args[idx].substring(3, args[idx].length()))
                        .intValue();
                }
                else if (args[idx].startsWith("-j:")) {
                    joystickNum =
                        Integer.valueOf(args[idx].substring(3, args[idx].length()))
                        .intValue();
                }
                else if (args[idx].startsWith("-j2:")) {
                    joystickNumEx =
                        Integer.valueOf(args[idx].substring(4, args[idx].length()))
                        .intValue();
                }
                else if (args[idx].startsWith("-v")) {
                    for (int id = -1; id <= Joystick.getNumDevices(); id++) {
                        System.out.println("Joystick " + id + ": " +
                       Joystick.isPluggedIn(id));
                    }
                }
                else if (args[idx].startsWith("-h")) {
                    help();
                }
                else {
                    System.out.println("Unknown option: " + args[idx]);
                    help();
                }
            }
            if (joystickNum >= 0) {
                if (joystickNumEx < 0) {
                    joystickNumEx = joystickNum;
                }
                mainFrame = new JoystickTest(joystickNum);
            }
            else {
                mainFrame = new JoystickTest();
            }
            if (deadZone >= 0.0) {
                mainFrame.setDeadZone(deadZone);
            }
            mainFrame.setPollInterval(interval);
            mainFrame.updateDeadZone();
            mainFrame.pack();
            mainFrame.setTitle("Polling Joystick");
            //mainFrame.show();
            mainFrame.setVisible(true);
            mainFrame.startPolling();

            Point pt = mainFrame.getLocation();
            //listenerFrame.show();
        } catch (IOException e) {
            System.err.println("");
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Source Code 4. The JoystickTEST class

D

Client – Inspection Application

CLIENT.CONTROLLER CamController.java

```
package client.controller;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.IOException;

import client.model.CamModel;
import client.view.panel.CamControlPanel;

import com.centralnexus.input.*;

public class CamController {

	private CamModel.CameraDirection cam_state = CamModel.CameraDirection.STOP_ALL;

	private Joystick joy;

	public CamController(){
		//Register in the service registry in order to avoid multiple instances?

		try {
			joy = Joystick.createInstance();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	//Add listeners
	public void addListeners(final CamControlPanel viewer, final CamModel model){
		//Initializing buttonListeners
		model.upMouseListener = new MouseListener() {

			@Override
			public void mouseReleased(MouseEvent e) { moveCam(model,
				CamModel.CameraDirection.UP, false); }
			@Override
			public void mousePressed(MouseEvent e) { moveCam(model,
				CamModel.CameraDirection.UP, true); }
```

```
        @Override
        public void mouseExited(MouseEvent e) {/*  moveCam(model,
               CamModel.CameraDirection.UP, false); */}
        @Override
        public void mouseEntered(MouseEvent e) {}
        @Override
        public void mouseClicked(MouseEvent e) {}

};

model.downMouseListener = new MouseListener() {

        @Override
        public void mouseReleased(MouseEvent e) { moveCam(model,
               CamModel.CameraDirection.DOWN, false); }
        @Override
        public void mousePressed(MouseEvent e) { moveCam(model,
               CamModel.CameraDirection.DOWN, true);}
        @Override
        public void mouseExited(MouseEvent e) {/*  moveCam(model,
               CamModel.CameraDirection.DOWN, false); */}
        @Override
        public void mouseEntered(MouseEvent e) {}
        @Override
        public void mouseClicked(MouseEvent e) {}

};

model.leftMouseListener = new MouseListener() {

        @Override
        public void mouseReleased(MouseEvent e) { moveCam(model,
               CamModel.CameraDirection.LEFT, false); }
        @Override
        public void mousePressed(MouseEvent e) { moveCam(model,
               CamModel.CameraDirection.LEFT, true); }
        @Override
        public void mouseExited(MouseEvent e) {/*  moveCam(model,
               CamModel.CameraDirection.LEFT, false); */}
        @Override
        public void mouseEntered(MouseEvent e) {}
        @Override
        public void mouseClicked(MouseEvent e) {}

};

model.rightMouseListener = new MouseListener() {

        @Override
        public void mouseReleased(MouseEvent e) { moveCam(model,
               CamModel.CameraDirection.RIGHT, false); }
        @Override
        public void mousePressed(MouseEvent e) {moveCam(model,
               CamModel.CameraDirection.RIGHT, true); }
        @Override
        public void mouseExited(MouseEvent e) {/*  moveCam(model,
               CamModel.CameraDirection.RIGHT, false); */}

        @Override
        public void mouseEntered(MouseEvent e) {}
        @Override
        public void mouseClicked(MouseEvent e) {}

};
```

```
model.stopAllMouseListener = new MouseListener() {

	@Override
	public void mouseReleased(MouseEvent e) {moveCam(model,
		CamModel.CameraDirection.STOP_ALL, false); }
	@Override
	public void mousePressed(MouseEvent e) { moveCam(model,
		CamModel.CameraDirection.STOP_ALL, true); }
	@Override
	public void mouseExited(MouseEvent e) {/*  moveCam(model,
		CamModel.CameraDirection.STOP_ALL, false); */}

	@Override
	public void mouseEntered(MouseEvent e) {}
	@Override
	public void mouseClicked(MouseEvent e) {}

};
```

```
model.joyListener = new JoystickListener() {

	@Override
	public void joystickAxisChanged(Joystick arg0) {
		// TODO Auto-generated method stub
		double z = arg0.getZ();
		double r = arg0.getR();

		switch(cam_state){
		case STOP_ALL:
			if(z > 0.3) {
				moveCam(model, CamModel.CameraDirection.DOWN,
				true);
				cam_state = CamModel.CameraDirection.DOWN;
			}
			if(z < -0.3) {
				moveCam(model, CamModel.CameraDirection.UP,
				true);
				cam_state = CamModel.CameraDirection.UP;
			}
			if(r > 0.3) {
				moveCam(model, CamModel.CameraDirection.RIGHT,
				true);
				cam_state = CamModel.CameraDirection.RIGHT;
			}
			if(r < -0.3) {
				moveCam(model, CamModel.CameraDirection.LEFT,
				true);
				cam_state = CamModel.CameraDirection.LEFT;
			}
			break;
		case RIGHT:
			if(r < 0.3) {
				moveCam(model, CamModel.CameraDirection.RIGHT,
				false);
				cam_state = CamModel.CameraDirection.STOP_ALL;
			}
			break;
		case LEFT:
			if(r > -0.3) {
				moveCam(model, CamModel.CameraDirection.LEFT,
				false);
				cam_state = CamModel.CameraDirection.STOP_ALL;
			}
			break;
```

```
                case UP:
                        if(z > -0.3) {
                                moveCam(model, CamModel.CameraDirection.UP,
                                false);
                                cam_state = CamModel.CameraDirection.STOP_ALL;
                        }
                        break;
                case DOWN:
                        if(z < 0.3) {
                                moveCam(model, CamModel.CameraDirection.DOWN,
                                false);
                                cam_state = CamModel.CameraDirection.STOP_ALL;
                        }
                        break;
                }
        }

        @Override
        public void joystickButtonChanged(Joystick arg0) {
                // TODO Auto-generated method stub

        }

};
```

```
model.keyListener = new KeyListener() {

        @Override
        public void keyPressed(KeyEvent arg0) {
                // TODO Auto-generated method stub
                System.out.println("PRESSED:" + arg0.toString());

        }

        @Override
        public void keyReleased(KeyEvent arg0) {
                // TODO Auto-generated method stub
                System.out.println("RELEASED:" + arg0.toString());
        }

        @Override
        public void keyTyped(KeyEvent arg0) {
                // TODO Auto-generated method stub

        }
};
```

```
joy.addJoystickListener(model.joyListener);

viewer.addKeyListener(model.keyListener);
```

```
//Adding buttonListeners
viewer.getUpBtn().addMouseListener(model.getUpMouseListener());
viewer.getDownBtn().addMouseListener(model.getDownMouseListener());
viewer.getLeftBtn().addMouseListener(model.getLeftMouseListener());
viewer.getRightBtn().addMouseListener(model.getRightMouseListener());
viewer.getStopAllBtn().addMouseListener(model.getStopAllMouseListener());

}
```

```
	//Remove listeners
	public void removeListeners(final CamControlPanel viewer, final CamModel model){
		//Remove all listeners (free memory)
		viewer.getUpBtn().removeMouseListener(model.getUpMouseListener());
		viewer.getDownBtn().removeMouseListener(model.getDownMouseListener());
		viewer.getLeftBtn().removeMouseListener(model.getLeftMouseListener());
		viewer.getRightBtn().removeMouseListener(model.getRightMouseListener());
		viewer.getStopAllBtn().removeMouseListener(model.getStopAllMouseListener());
	}

	//Controller functions

	public void moveCam(CamModel model, CamModel.CameraDirection command, boolean
	enable){
		//Checking if new state: Not a pushed button ,
//FIXME		if((!enable && command==CamModel.STOP_ALL) && !model.isLastCommandEnable() &&
			command==CamModel.STOP_ALL) return;

		//Updating model
		model.setLastCommand(command, enable);

		//Telling driver what to do
		switch (command) {
		case UP:
			if(enable) model.getCameraDriver().goUp();
			else model.getCameraDriver().stopUp();
			break;

		case DOWN:
			if(enable) model.getCameraDriver().goDown();
			else model.getCameraDriver().stopDown();
			break;

		case LEFT:
			if(enable) model.getCameraDriver().goLeft();
			else model.getCameraDriver().stopLeft();
			break;

		case RIGHT:
			if(enable) model.getCameraDriver().goRight();
			else model.getCameraDriver().stopRight();
			break;

		case STOP_ALL:
			model.getCameraDriver().stopAll();
			break;

		default:
			model.getCameraDriver().stopAll();
			break;
		}
	}

}
```

Source Code 5. The CamController class

CLIENT.CONTROLLER VehicleController.java

```
package client.controller;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.IOException;
import client.drivers.VehicleDriver.VehicleCommand;
import client.model.VehicleModel;
import client.view.panel.VehicleControlPanel;
import com.centralnexus.input.*;



public class VehicleController {

	private Joystick joy;

	public VehicleController(){
		//Register in the service registry in order to avoid multiple instances?

		try {
			joy = Joystick.createInstance();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}

	//Add listeners
	public void addListeners(final VehicleControlPanel viewer, final VehicleModel
	model){
		//Initializing buttonListeners
		model.fastFwdMouseListener = new MouseListener() {
			@Override
			public void mouseReleased(MouseEvent arg0) { moveVehicle(model,
				VehicleCommand.STOP); }
			@Override
			public void mousePressed(MouseEvent arg0) { moveVehicle(model,
				VehicleCommand.FFWD); }
			@Override
			public void mouseExited(MouseEvent arg0) {/* moveVehicle(model,
				VehicleCommand.STOP); */}
			@Override
			public void mouseEntered(MouseEvent arg0) {}
			@Override
			public void mouseClicked(MouseEvent arg0) {}
		};

		model.fwdMouseListener = new MouseListener() {
			@Override
			public void mouseReleased(MouseEvent arg0) { moveVehicle(model,
				VehicleCommand.STOP); }
			@Override
			public void mousePressed(MouseEvent arg0) { moveVehicle(model,
				VehicleCommand.FWD); }
			@Override
			public void mouseExited(MouseEvent arg0) {/* moveVehicle(model,
				VehicleCommand.STOP); */}
```

```
        @Override
        public void mouseEntered(MouseEvent arg0) {}
        @Override
        public void mouseClicked(MouseEvent arg0) {}
};

model.revMouseListener = new MouseListener() {
        @Override
        public void mouseReleased(MouseEvent arg0) { moveVehicle(model,
                VehicleCommand.STOP); }
        @Override
        public void mousePressed(MouseEvent arg0) { moveVehicle(model,
                VehicleCommand.REV); }
        @Override
        public void mouseExited(MouseEvent arg0) {/* moveVehicle(model,
                VehicleCommand.STOP); */}
        @Override
        public void mouseEntered(MouseEvent arg0) {}
        @Override
        public void mouseClicked(MouseEvent arg0) {}
};

model.fastRevMouseListener = new MouseListener() {
        @Override
        public void mouseReleased(MouseEvent arg0) { moveVehicle(model,
                VehicleCommand.STOP); }
        @Override
        public void mousePressed(MouseEvent arg0) { moveVehicle(model,
                VehicleCommand.RREV); }
        @Override
        public void mouseExited(MouseEvent arg0) {/* moveVehicle(model,
                VehicleCommand.STOP); */}
        @Override
        public void mouseEntered(MouseEvent arg0) {}
        @Override
        public void mouseClicked(MouseEvent arg0) {}
};
```

```
model.joyListener = new JoystickListener() {

        @Override
        public void joystickAxisChanged(Joystick arg0) {
                // TODO Auto-generated method stub
                /*System.out.println(arg0.toString());
         System.out.println(Double.toString(arg0.getY()));
         System.out.println(Double.toString(arg0.getZ()));
         System.out.println(Double.toString(arg0.getR()));*/
                double y = arg0.getY();

                if(y < -0.6) moveVehicle(model, VehicleCommand.FFWD);
                if(y < -0.3 && y >= -0.6) moveVehicle(model,
                VehicleCommand.FWD);
                if(y < 0.3 && y >= -0.3) moveVehicle(model,
                VehicleCommand.STOP);
                if(y < 0.6 && y >= 0.3) moveVehicle(model,
                        VehicleCommand.REV);
                if(y >= 0.6) moveVehicle(model, VehicleCommand.RREV);
        }

        @Override
        public void joystickButtonChanged(Joystick arg0) {
                // TODO Auto-generated method stub
        }

};
```

```
            joy.addJoystickListener(model.joyListener);

            //Adding buttonListeners
            viewer.getFastForwardBtn().addMouseListener(model.getFastFwdMouseListener());
            viewer.getForwardBtn().addMouseListener(model.getFwdMouseListener());
            viewer.getReverseBtn().addMouseListener(model.getRevMouseListener());
            viewer.getFastReverseBtn().addMouseListener(model.getFastRevMouseListener());
        }

        //Remove listeners
        public void removeListeners(final VehicleControlPanel viewer, final VehicleModel
        model){
            //Remove all listeners (free memory)
            viewer.getForwardBtn().removeMouseListener(model.getFwdMouseListener());
            viewer.getReverseBtn().removeMouseListener(model.getRevMouseListener());
        }

        //Controller functions
        private void moveVehicle(VehicleModel model, VehicleCommand command){
            model.getVehicleDriver().sendVehicleCommand(command);
        }

}
```

Source Code 6. The VehicleController class

CLIENT.MODEL CamModel.java

```
package client.model;

import java.awt.event.KeyListener;
import java.awt.event.MouseListener;

import client.ClientConstants;
import client.drivers.IPCamera.IPCameraDriver;

import com.centralnexus.input.*;

public class CamModel extends ModelPropertyChange {

	//Commands
	public static enum CameraDirection{ UP, DOWN, LEFT, RIGHT, STOP_ALL, UNKNOWN}

//	public static final int UP = 1;
//	public static final int DOWN = 2;
//	public static final int LEFT = 3;
//	public static final int RIGHT = 4;
//	public static final int STOP_ALL = 5;

	//Variables
	private CameraDirection lastCommandDirection;	//Used to prevent unnecessary
	network traffic
	private boolean lastCommandEnable;  //Used to prevent unnecessary network traffic
	public MouseListener upMouseListener;
	public MouseListener downMouseListener;
	public MouseListener leftMouseListener;
	public MouseListener rightMouseListener;
	public MouseListener stopAllMouseListener;

	public JoystickListener joyListener;
	public KeyListener keyListener;
	private final IPCameraDriver cameraDriver;

	public CamModel(){
		super();
		lastCommandDirection = CameraDirection.UNKNOWN;
		cameraDriver = new IPCameraDriver(ClientConstants.CAMERA_ADDRESS,
			ClientConstants.CAMERA_USERNAME, ClientConstants.CAMERA_PASSWORD);

	}

	//Getters & Setters



	public void setLastCommand(CameraDirection command, boolean enable) {
		//Setting new status
		CameraDirection lastDir = this.lastCommandDirection;
		boolean lastMode = this.lastCommandEnable;
		this.lastCommandDirection = command;
		this.lastCommandEnable = lastMode;

		//Firing property change

	changeSupport.firePropertyChange(ClientConstants.CAMERA_CONTROLLER_MOVEMENT_SET_DIR,
	lastDir, lastCommandDirection);

	changeSupport.firePropertyChange(ClientConstants.CAMERA_CONTROLLER_MOVEMENT_SET_MOD,
	lastMode, lastCommandEnable);
	}
```

```
	public CameraDirection getLastCommandDirection() {
		return lastCommandDirection;
	}

	public boolean isLastCommandEnable() {
		return lastCommandEnable;
	}

	public MouseListener getUpMouseListener() {
		return upMouseListener;
	}


	public MouseListener getDownMouseListener() {
		return downMouseListener;
	}


	public MouseListener getLeftMouseListener() {
		return leftMouseListener;
	}


	public MouseListener getRightMouseListener() {
		return rightMouseListener;
	}


	public MouseListener getStopAllMouseListener() {
		return stopAllMouseListener;
	}

	public IPCameraDriver getCameraDriver() {
		return cameraDriver;
	}


}
```

Source Code 7. The CamModel class

CLIENT.MODEL VehicleModel.java

```
package client.model;

import com.centralnexus.input.*;

public class VehicleModel extends ModelPropertyChange {

	//Variables
	private int lastCommand;
	public MouseListener fastFwdMouseListener;
	public MouseListener fwdMouseListener;
	public MouseListener revMouseListener;
	public MouseListener fastRevMouseListener;

	public JoystickListener joyListener;

	private final VehicleDriver vehicleDriver;

	public VehicleModel(){
		super();
		lastCommand = -1;

		vehicleDriver = new VehicleDriver();
	}

	//Getters & Setters
	public void setMovement(int command, boolean enable){
		int oldLastCommand = lastCommand;
		lastCommand = command;
		System.out.println("MOVE: "+command+(enable?"on":"off"));
		//FIRE ON NETWORK

	changeSupport.firePropertyChange(ClientConstants.VEHICLE_CONTROLLER_COMMAND_
MODE,		oldLastCommand, lastCommand);
	}

	public VehicleDriver getVehicleDriver() {
		return vehicleDriver;
	}

	public MouseListener getFwdMouseListener() {
		return fwdMouseListener;
	}

	public MouseListener getRevMouseListener() {
		return revMouseListener;
	}

	public MouseListener getFastFwdMouseListener() {
		return fastFwdMouseListener;
	}

	public MouseListener getFastRevMouseListener() {
		return fastRevMouseListener;
	}
}
```

Source Code 8. The VehicleModel class

E

Inspection Summary

Minerals Management Service
Offshore Wind Turbine Facility Inspection Summary

The following information shall be submitted to the MMS as a record of ongoing structural integrity management activities for offshore wind turbine facilities. Indicate which inspections were performed for each facility for the reporting year and whether anomalies were identified.

					Subsea Structure & Equipment				Topsides Structure		Blades		Anomalies		
Oper	FID	RY	YI	WD	SS-A	SS-I	SS-E	SE-I	TS-A	TS-I	BL-A	BL-I	M-AN	S-AN	Remarks
ACME Wind	T01-ACME	2013	2010	30	X	X		X	X		X	X	N	Y	Anomaly Rpt T01-A
ACME Wind	T02-ACME	2013	2010	30	X	X		X	X		X	X	N	N	
ACME Wind	T03-ACME	2013	2010	31	X	X		X	X		X	X	N	N	
ACME Wind	T04-ACME	2013	2010	30	X	X		X	X		X	X	N	N	
ACME Wind	T11-ACME	2013	2011	25	X				X		X		Y	N	Grating damage
ACME Wind	T12-ACME	2013	2011	27	X				X		X		Y	N	Grating damage
ACME Wind	T14-ACME	2013	2011	24	X				X		X		Y	N	Minor corrosion
ACME Wind	T101-ACME	2013	2009	52	X				X		X		N	N	
ACME Wind	T102-ACME	2013	2009	51	X		X		X		X		N	N	
ACME Wind	T103-ACME	2013	2009	55	X				X		X		N	N	
ACME Wind	T104-ACME	2013	2009	50	X		X		X		X		N	N	
ACME Wind	T105-ACME	2013	2009	50	X				X		X		N	N	
ACME Wind	T201-ACME	2013	2010	42	X	X		X	X	X	X		Y	N	Handrail bent
ACME Wind	T202-ACME	2013	2010	42	X	X		X	X	X	X		Y	Y	Anomaly Rpt T202-B
ACME Wind	T203-ACME	2013	2010	43	X	X		X	X	X	X		Y	N	Small nacelle dent
ACME Wind	T204-ACME	2013	2010	43	X	X		X	X	X	X		N	N	

Oper Operator Name
FID Facility ID
RY Report Year
YI Year Installed
WD Water Depth
SS-A Subsea Structure Annual
SS-I Subsea Structure Intermediate
SS-E Subsea Structure Extended
SE-I Subsea Equipment Intermediate
TS-A Topsides Structure Annual
TS-I Topsides Structure Intermediate
BL-A Blades Annual
BL-I Blades Intermediate
M-AN Minor Anomaly (e.g., maintenance items, minor repair/replacement requiring no engineering, etc.)
S-AN Serious Anomaly (e.g., conditions requiring engineered repair, modification to future inspection program, etc.)

F

Inspection Checklist

Minerals Management Service
Offshore Wind Turbine Facility Annual Inspections Checklist

Note: This checklist is not a substitute for an integrity management program and data recording system for offshore wind turbine facilities. It is a data summary requested by the MMS.

Operator: _____ Facility ID: _____

Report Year: _____ Year Installed/Water Depth _____

SAFETY / SIGNAGE / MARKINGS
Indicate Yes or No to each item and whether or not Corrective Action (CA) was taken

No.	Item	Y	N	CA
1.	Are walkways, ladders, handrails, stairs and other access systems in good working condition?			
2.	Are warning/safety/instructional signs visible and legible?			
3.	Are markings showing facility identification, water level markings, etc. visible and legible			
4.	Are there obstructions to egress paths (e.g., equipment stored on stairs)?			
5.	Are fall protection anchorage points in good condition?			
6.	Are navigation and aviation warning lights operational?			
7.	Are fire protection systems operational?			
8.	Are there other anomalies noted?			

SUBSEA CATHODIC PROTECTION
Indicate Yes or No to each item and whether or not Corrective Action (CA) was taken

No.	Item	Y	N	CA
1.	Are CP measurements within acceptable range?			
2.	Are there other anomalies noted?			

TOPSIDES STRUCTURE INSPECTIONS
Indicate Yes or No to each item and whether or not Corrective Action (CA) was taken

No.	Item	Y	N	CA
1.	Are there any signs of coating breakdown and/or corrosion?			
2.	Are there any signs of physical damage including dents, holes or other deformation to structural members or nacelle housing?			
3.	Are there any cracks or visible indications at welded connections?			
4.	At bolted connections are nuts noticeably loose?			
5.	Are cables and risers, and their attachments in good condition?			
6.	Are there other anomalies noted?			

BLADE INSPECTIONS
Indicate Yes or No to each item and whether or not Corrective Action (CA) was taken

No.	Item	Y	N	CA
1.	Is there any sign of blade material degradation (e.g., de-lamination)?			
2.	Are there signs of blade damage or erosion?			
3.	Are there signs of corrosion especially at the blade attachment points?			
4.	Are there other anomalies noted?			

G

The DVD

The DVD includes:

- Appendix A plus a Map of Wind Farms in Europe.
- Data sheets of Joysticks from Genge & Thoma and OEM Inc.
- The whole java code of the Client – Inspection.
- Appendixes E and F, Inspection Summary and Checklist.
- Photos of Inspection TEST.
- A copy of the Master Thesis.

References

[1] P. Jain, Wind energy engineering, New York: McGraw-Hill, 2011.

[2] C. L. Archer and M. Z. Jacobson, “Evaluation of global wind power,” Journal of Geophysical Research - Atmospheres, 2005.

[3] Breakbulk Staff, “Breakbulk,” 16 September 2011. [Online]. Available: http://www.breakbulk.com/wind-renewables/japan-plans-floating-wind-power-plant. [Accessed November 2011].

[4] “Wikipedia,” [Online]. Available: http://en.wikipedia.org/wiki/Floating_wind_turbine#Hywind. [Accessed November 2011].

[5] I. f. E. T. (IFE), “Deep sea offshore wind turbine technology,” February 2011.

[6] “About Wind Power: NORWEA,” [Online]. Available: http://www.nyfornybar.no/. [Accessed 23 October 2011].

[7] European Wind Energy Association, “The European offshore wind industry key 2011 trends and statistics,” EWEA, 2012.

[8] E. Norge, “Blowing in the wind,” vol. Final Report, December 2009.

[9] “Vind i Norgue og Europa,” [Online]. Available: http://www.vindkraft.no/Default.aspx?ID=148. [Accessed 23 October 2011].

[10] I. f. E. T. (IFE), “Wind Energy,” January 2011.

[11] “Wind Turbine Technology for Offshore Locations,” [Online]. Available: http://www.wind-energy-the-facts.org/en/part-i-technology/chapter-5-offshore/wind-turbine-technology-for-offshore-locations/. [Accessed 25 October 2011].

[12] T. M. Karlsen, “Ekskursjon GE Hundhammerfjellet,” Trondheim, 2011.

[13] Heinrich A. Bieler, “Genge & Thoma,” GT Joystiks & Sensors, [Online]. Available: http://www.gengethoma.com/en/Products/201.Joysticks. [Accessed 10 January 2012].

[14] “Wikipedia,” [Online]. Available: http://en.wikipedia.org/wiki/Joystick. [Accessed December 2011].

[15] T. Engdahl, “Epanorama,” 1994-2009. [Online]. Available: http://www.epanorama.net/documents/joystick/intro.html. [Accessed January 2012].

[16] “OEM Controls Inc.,” 2006. [Online]. Available: http://www.oemcontrols.com/halleffect.html. [Accessed January 2012].

[17] E. Berntsen, “Prestudy into the applicability of using a robotic arm in an O&M telepresence system, operating inside a nacelle,” NTNU, Trondheim, 2012.

[18] C. Fan and B. Zhang, “Analysis on the dynamic image quaility of th TDICCD camera,” Optics Photonics and Energy Enginneering, Wuhan, China, 2010.

[19] T. Watanabe, R. Suganuma and T. Kai, “Camera System for reducing the influence of vibration generated by actuators,” Nikon Corporation, 2000.

[20] Student projects, “Suburface mobile augmented reality technology,” Smart Vidente, [Online]. Available: http://studierstube.icg.tugraz.at/outdoor/. [Accessed January 2012].

[21] “Toyota's Car Of the Future,” Designboom, December 2011. [Online]. Available: http://www.buzzfeed.com/keenan/toyotas-car-of-the-future.

[22] G. Rhoten, “JXInput - Input Devices for Java,” 2000. [Online]. Available: http://www.hardcode.de/jxinput/. [Accessed November 2011].

[23] AWEA, “American Wind Energy Association,” [Online]. Available: http://www.awea.org/issues/supply_chain/Anatomy-of-a-Wind-Turbine.cfm. [Accessed November 2011].

[24] T. Burton, N. Jenkins, D. Sharpe and E. Bossanyi, Wind Energy Handbook, UK: John Wiley & Sons, 2011.

[25] H. Stiesdal, “The Wind Turbine. Components and Operation,” Bonus Energy A/S, Brande, 1999.

[26] Advantech Co., “Advantech,” [Online]. Available: http://www.advantech.com/sector/power-generation-distribution/CaseStudies.aspx?doc_id=%7BFC0B4C45-7467-4CE3-B00D-A2AF48AF219. [Accessed November 2011].

[27] Avantech Co., Advantech, [Online]. Available: http://www.advantech.eu/sector/power-energy/CaseStudies.aspx?doc_id=%7BDF8957E3-CB0A-407A-85B5-EBB3BEC23F3. [Accessed December 2011].

[28] C. Schöntag, “Optimisation of Operatio and Maintenace of Offshore Wind Farms,” Institute for Wind Energy, 1996.

[29] T. Petersen, “Offshore wind power - the operational aspects,” Vestas Danish Wind Technology, Lem.

[30] G. v. Bussel, “The Development of an Expert System for the determination of Availability an O&M Costs for Offshre Wind Farms,” European Wind Energy Conference, Nice, 1999.

[31] T. Knob, “Questionnaire response,” 2001.

[32] G. v. Bussel, “Reliability, availability and maintenance aspects of large-scale offshore wind farms, a concept study,” Delft University of Technology, Marec, 2001.

[33] G. v. Bussel and C. Schöntag, “Operation and Maintenance Aspects of Large Offshore Wind Farms,” EWEC, Dublin, 1998.

[34] H. Braam and L. Rademakers, “The influence of weather conditions on the strategies and costs of operation and maintenance,” Offshore Wind Energy Conference, Brussels, 2001.

[35] R. E. Sheppard and F. J. W. C. Puskar, “Inspection Guidance for Offshore Wind Turbine Facilities,” Offhore Technology Conference, Texas, 2010.

[36] Energo Engineering, Inc, “Inspection Methodologies for Offshre Wind Turbine Facilities,” Energo, Houston, 2009.

[37] Ø. Netland, “Pre-Study on Cost-effective, Remote, Environmental Friendly O&M of Large Scale Offshore Wind Turbine Plants,” NTNU, Trondheim, 2011.

[38] “Nacelle lighting and power system,” Moltec Windpower Products, 2009. [Online]. Available: http://www.moltecwind.com/Products/NacelleLighting.html. [Accessed January 2012].