ISBN 978-82-326-0112-7 (printed version)
ISBN 978-82-326-0113-4 (electronic version)
ISSN 1503-8181

Øyvind Netland

Doctoral Thesis

NTNU – Trondheim
Norwegian University of
Science and Technology

Doctoral theses at NTNU, 2014:94

NTNU
Norwegian University of Science and Technology
Thesis for degree of Philosophiae Doctor
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Engineering Cybernetics

NTNU

Doctoral theses at NTNU, 2014:94

Øyvind Netland

Remote Inspection of Offshore Wind Turbines

A Study of the Benefits, Usability and Feasibility

NTNU – Trondheim
Norwegian University of
Science and Technology

Øyvind Netland

Remote Inspection of Offshore Wind Turbines

A Study of the Benefits, Usability and Feasibility

Thesis for the degree of Philosophiae Doctor

Trondheim, March 2014

Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Engineering Cybernetics

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Engineering Cybernetics

© Øyvind Netland

ISBN 978-82-326-0112-7 (printed version)
ISBN 978-82-326-0113-4 (electronic version)
ISSN 1503-8181

ITK Report 2014-1-W

Doctoral theses at NTNU, 2014:94

Printed by Skipnes Kommunikasjon as

Summary

Offshore wind turbines are large, unmanned machines that are deliberately located in areas with strong wind. Access to the turbines is expensive and time consuming, and in periods of harsh weather, the turbines can be inaccessible. This is challenging for operation and maintenance of the turbines, and the same maintenance strategies that have been used successfully on land will not necessarily be viable offshore.

This thesis suggests the use of a remotely controlled robot that can be used to inspect a turbine without accessing it. The robot is intended to be installed inside the wind turbine nacelle and move on a rail. Since access to the turbine is not needed, inspections can be done inexpensively, with less planning and regardless of the weather. This concept has been evaluated with the NOWIcob tool and the simulation results indicated that it would have a positive effect on the availability and cost of energy for offshore wind turbines.

A remote inspection robot prototype has been developed, and used for evaluation of the concept in four experiments. These were usability tests, where participants did both remote inspections using the prototype and manned inspections. Two of the experiments were pilot experiments with 4 participants each. The two larger experiments had 21 and 31 participants. Observations and participants' suggestions from each of the experiments were used to improve the prototype, following a user-centered development method. This is a commonly used method within human-computer interaction to improve the usability of a system. The results from the last experiment were the most accurate due to an improved experiment procedure and the largest number of participants. They showed that remote inspection was almost as effective as manned, and that remote inspections would likely perform better if more time had been available.

To simplify the development of the prototype's control system, a method for using code generated from MathWorks Simulink Coder, called *Software Module Real-time Target* (SMRT), was developed. It is intended for embedded developers that would like to include code generated by Simulink Coder into a larger software project. It has been designed to be used with only a minimal knowledge of Simulink Coder. As SMRT by itself can be of interest for other developers, it has been described in its own chapter. The code is also available as open source online.

The experiments presented in this thesis mark the end of the work with the current prototype and the laboratory testing. The experiments have demonstrated the plausibility of using a telerobot for remote inspections, with almost the same effectiveness as manned inspection. The experience gained from these experiments will be brought into the next phase, which will be the development of a new prototype for installation in a real wind turbine. The laboratory evaluation has been an important and necessary step in the development of such a system.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of philosophiae doctor (PhD) at the Norwegian University of Science and Technology. The work for this thesis has been carried out at the Department of Engineering Cybernetics in the period from April 2010 to March 2014. It has been funded by Norwegian Research Centre for Offshore Wind Technology (NOWITECH) and the Research Council of Norway.

This PhD started when I sent my former master thesis supervisor and future PhD advisor Amund Skavhaug an email, and asked if he thought it was I good idea for me to pursue a PhD. He obviously did, and I was happy to start working with an interesting topic within wind energy, which I see as an important industry for the future. I think Amund and me make a good team. He is a constant source of ideas, and I think I do a good job at sorting them and put the best ones into use. As a supervisor, Amund is always available when you need assistance. I know this is appreciated by all his students, both master and PhDs, but sometimes I worry that he is too available for his own good.

I would also like to thank Dag Sjong for his contributions to establishing the PhD position. Norsk Automatisering AS, which consists of Dag and Amund, has also contributed during my work, and I look forward to continue the co-operation with them, to take the remote inspection concept from the laboratory and into wind turbines.

My experiments would not be possible without the students that participated, the master student that helped me during one of the experiment Hilde Marie Schade, and the mechanical workshop at the department that put the laboratory together. Amund also put me in contact with Gunnar Jenssen from Sintef, who gave me valuable advice before the experiments and co-authored two of the articles.

I have enjoyed working together with the other PhDs and post docs in our department. We always managed to get into interesting and strange conversations during lunch and coffee breaks. I am glad the Wednesday meeting tradition is alive and well, and hope to sneak by every once in a while to get some delicious cake. I also had the opportunity to work with several master students, especially Viktor Fidje and Tor Karlsen.

As part of NOWITECH, I have also had colleagues outside our department, both the NOWITECH PhD students and the members of work package 5. It has been interesting to talk with people with very different backgrounds within the multidisciplinary field of wind energy. It has often been difficult to understand the work of the other PhDs, but I have learned a lot about wind energy from trying.

I have my parents, Jan and Berit, to thank for a good upbringing and support. I grew up 200 meters from the student housing area of the Norwegian University of Life Sciences, and I knew I wanted to be a student a long time before I had any idea what that meant. That I would continue all the way to a PhD was not necessarily given, but I suspect that it was my father's secret plan all along. I would also like to thank my sister Malena, even if she tried to sabotage my academic career by sitting on my homework to get some attention. Fortunately, she has stopped doing that, and I wish her luck on her master thesis in food science.

Finally, I would like to thank my fiancee and the love of my life, Marie. I might not have been the best at showing it, but I have appreciated and depended on your encouragement and complete support during these years.

Contents

Summary **i**

Preface **iii**

Contents **v**

List of Figures . . . vii

List of Tables . . . ix

Nomenclature . . . xi

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Main Contributions . . . 6

1.3 Thesis Outline . . . 7

1.4 List of Publications . . . 8

2 Operation and Maintenance of Offshore Wind Farms **11**

2.1 Maintenance Strategies . . . 11

2.2 Remote Maintenance . . . 14

2.3 Cost-Benefit Evaluation of Remote Inspection . . . 18

3 Prototyping and Evaluation of Remote Inspection for Offshore Wind **21**

3.1 Related Work with Usability Testing of Telerobots . . . 21

3.2 The Laboratory . . . 29

3.3 The Remote Inspection Prototype . . . 33

3.4 Results . . . 42

3.5 Discussion . . . 49

4 Simulink Coder Generated Code as a Module within a Software Project **53**

4.1 Simulink Coder for Embedded Linux . . . 53

4.2 Software Module Real-Time Target . . . 57

4.3 Discussion . . . 65

5 Concluding Remarks **67**
5.1 Conclusions 67
5.2 Future Work 69

6 Original Publications **71**
Paper A Adaption of MathWorks Real-Time Workshop for an Unsupported Embedded Platform 73
Paper B Prototyping and Evaluation of a Telerobot for Remote Inspection of Offshore Wind Farms 81
Paper C Two Pilot Experiments on the Feasibility of Telerobotic Inspection of Offshore Wind Turbines 89
Paper D Software Module Real-Time Target: Improving Development of Embedded Control System by Including Simulink Generated Code into Existing Code 95
Paper E An Experiment on the Effectiveness of Remote, Robotic Inspection Compared to Manned 101
Paper F Evaluation of Remote Inspection of Offshore Wind Turbines with a Tablet Controlled Telerobot 109
Paper G A Review of Experiments Evaluating the Usability of Mobile Telerobots 123

Bibliography **137**

List of Figures

1.1 Two wind turbines at Smøla wind farm. 1
1.2 Comparison between a 2.3MW turbine blade from Smøla wind farm and a bus. 2
1.3 View from the top of a wind turbine at Hundhammerfjellet wind farm. . . . 3
1.4 Frequency and downtime caused by different types of failures. The data is from 600 Swedish wind turbines on land over a 5-year period (Ribrant et al. 2007). 5

2.1 Concept illustration of a remote inspection robot inside a simplified and generic nacelle. The nacelle consists of main bearings, gearbox, generator, transformer and cabinets for electronics. The inspection robot is indicated with an arrow. An example rail configuration is also shown. 15
2.2 The improvements in availability and cost of energy compared to the base case. 19

3.1 Examples of different types of displays from Nielsen et al. (2005). 25
3.2 A section of the lab for evaluation of remote inspection 29
3.3 The prototype remote inspection robot used in the experiments. 33
3.4 A description complete control system as intended when installed in a wind turbine. 35
3.5 The Simulink Model used as part of the robot control system during the second experiment. 37
3.6 The evolution of the GUI. 38
3.7 Different views for the tablet controller. Controls for manual adjustments are shown in the lower left corners. 41
3.8 Results from the pilot experiment. Error bars are not shown due to the low number of participants. 42
3.9 Average detection rates for manned (μ_M), teleoperated (μ_T) and assisted inspections (μ_A). 43
3.10 Average NASA-TLX results for manned, teleoperated and assisted inspections. 44
3.11 The detection rates grouped on inspection method. 46

3.12 Average NASA-TLX results for manned and remote inspections. 46
3.13 The detection rates grouped on the most used tablet view. 47
3.14 Average NASA-TLX results for map, orginal and difference views. 48
3.15 Van De Laan results for the remote inspection prototype. 48

4.1 The Beaglebone development board. 57
4.2 The relationship between generated and custom code in SMRT compared to a typical use of GRT 59
4.3 Simulink model used in the test (colors indicates period of tasks). 61
4.4 Graphical representation of communication between generated and custom code in SMRT 63

List of Tables

3.1 Descriptions of usability test variants for evaluation of remote inspection. . . 32
3.2 Comparison between movement on rail and full mobility. 34
3.3 The evolution of the prototype versions. 36

4.1 Where configurations are stored in GRT and SMRT using Eclipse. 60
4.2 Execution time and periods of tasks used in scheduling test. 61
4.3 Results from simulations running for 3 hours. 62

Nomenclature

CBM	Condition-based maintenance
CM	Condition monitoring
CMS	Condition monitoring system
CTV	Crew Transfer Vessels
ERT	Embedded Real-Time Target
GRT	Generic Real-Time Target
NOWIcob	Norwegian Offshore Wind Cost and Benefit Model
O&M	Operation and maintenance
PTZ	Pan/Tilt/Zoom
RMS	Rate Monotonic Scheduling
RTDM	Xenomai real-time driver model
RTW	Real-Time Workshop, the former name for Simulink Coder
TLC	Target Language Compiler

Chapter 1

Introduction

1.1 Motivation

Figure 1.1: Two wind turbines at Smøla wind farm.

The world's energy production is today strongly dominated by fossil fuels, which is causing a growing concern for anthropogenic climate change. Even if this is a controversial topic in the general public, it has a strong scientific consensus. According to the Intergovernmental Panel on Climate Change (Alexander et al. 2013): "Continued emissions of greenhouse gases will cause further warming and changes in all components of the climate system. Limiting climate change will require substantial and sustained reductions of greenhouse gas emissions”. It is not realistic that all of the reduction in emissions can come from decreased global energy consumption, especially due to the rapid economic growth in areas with large populations. An increase in the production of energy from other sources than fossil fuels would also be necessary.

The earliest reliable historical accounts of windmills were from the 7th century in the Persian-Afghan border region of Seistan (Hau et al. 2000). More famous, at least from a European perspective, are the European windmills from the 12th century in England, Flanders and northern France. These were used in Europe until their peak in the 19th century when steam powered milling took over (Musgrove 2010).

Experiments with wind turbines for electricity generation started in the late 1800s, and over the years, a large number of different concepts were tested. An example was the first megawatt (1.25MW) turbine called “Grandpa’s Knob”. It was built in the United States in 1941, but it was not commercially viable, and the plans for building several similar turbines were discarded. After the oil crisis in 1973, there was a renewed interest for wind energy. While many countries experimented with large wind turbines, the Danish started building smaller and less complex machines, which often were sold to farmers. The look of modern wind turbines, with tubular towers and upwind rotors with three blades came from these. During the 1980s, the Danish producers were able to mass-produce these machines, and contributed to about half of the installed capacity during the Californian wind boom. In the 1990s, European countries started to build on a large scale. Today, wind energy contributes significantly to the energy production in Denmark, with 26% of the total electricity generation (EWEA 2012), Spain, Germany etc.

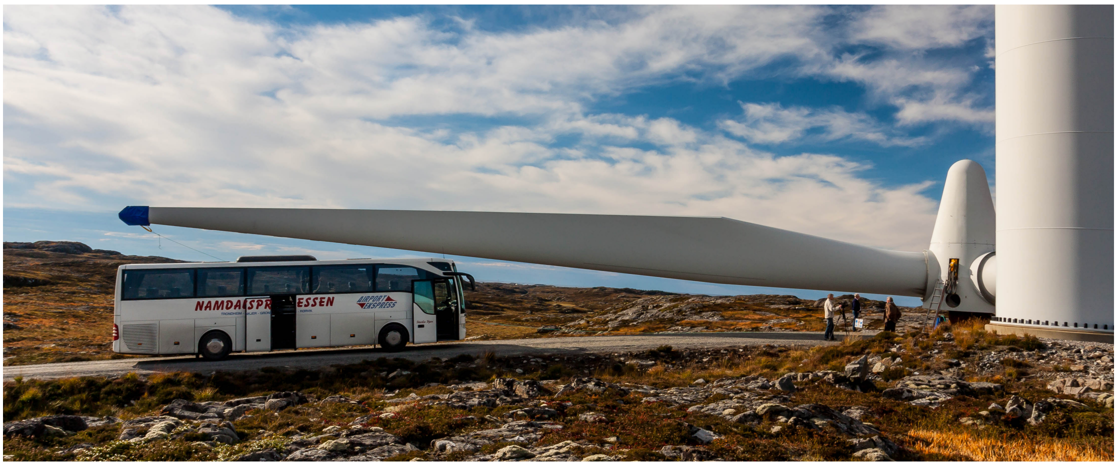


Figure 1.2: Comparison between a 2.3MW turbine blade from Smøla wind farm and a bus.

The size of wind turbines have increased steadily over the years. Since the wind energy increase with the square of the robot diameter, larger turbines are more effective and have a lower impact on the environment (Caduff et al. 2012). A turbine blade and a bus are shown together in figure 1.2, as an illustration of the size of modern wind turbines. The turbine in the figure is 2.3MW, thus it not considered a large turbine by today's standard. According to RenewableEnergyWorld.com, the world's largest offshore wind turbine in October 2013 was the Samsung S7.0-171[1], which is 196 meter tall, with a rotor diameter of 171 meters and a rated power of 7 MW.

Offshore wind energy is one of the more promising renewable sources of energy for the future. The capacity factor of a turbine, i.e. the percentage of the theoretical maximum energy output, can be increased from about 25% on land to about 40% offshore (Junginger et al. 2003) due to favorable wind conditions. Offshore wind turbines have on average 3000 full load hours per year, while wind turbines on land typically have 2000-2300 (Morthors et al. 2009).

Although people are generally positive to wind energy, most people do not want to have turbines close by, the so called “not in my backyard” problem. This is less of a problem for offshore wind turbines. Unless they are very close to the shore, the problem with audible noise is likely to be negligible. The visual impact depends on the viewing conditions, but according to Bishop (2002), any effect from turbines more than 20 km from the shore would be rare.

Figure 1.3: View from the top of a wind turbine at Hundhammerfjellet wind farm.

[1]http://www.renewableenergyworld.com/rea/news/article/2013/10/securing-the-worlds-largest-wind-turbine

According to EWEA (2012), it was about 5 GW of installed offshore wind energy capacity in Europe in 2012. It is estimated that the installed capacity will grow to 40 GW by 2020 and 150 GW by 2030 (EWEA 2011). This would correspond to about 4% of EU electricity consumption in 2020 and about 14% in 2030. Unfortunately, it is expensive to both build and operate turbines offshore, especially since turbine locations are deliberately chosen for their windy conditions. Harsh weather can delay the installation and prevent personnel from accessing the turbines for maintenance. While wind turbines on land have one of the lowest estimated levelized cost for power plants built in 2018, offshore turbines have one of the highest (EIA 2013). This means that these ambitious plans only will be possible if technology is improved and the cost of both installation and operation is reduced.

Offshore wind farms are located in areas with strong and stable wind for optimal energy production. Unfortunately, these are about the worst locations for operating the turbines. Waves can make it difficult or impossible to access the turbines by boat, and wind cause problems for lifting operations and access by helicopter. Often there are both strong wind and high waves. In the North Sea, where many offshore wind projects are planned, the turbines can be inaccessible for long periods due to the weather, especially in the winter months. Even if the weather conditions are favorable, there is still a high cost associated with transporting personnel to and from the turbines. Where a car driven by the maintenance personnel themselves is sufficient to access a wind turbine on land, a large boat with its own crew fitted with special equipment for turbine access is needed offshore. The same for heavy lifting operations, where a mobile crane can be used on land, a jack-up rig or another specialized vessel is needed offshore.

There are relatively few offshore wind farms today, and they have all been built recently. This means that there are little available information about the actual cost of *operation and maintenance* (O&M) for offshore wind turbines. However, estimates suggest that between 20% and 25% of the total energy cost from offshore wind turbines will be due to O&M (Lu et al. 2010; Musial et al. 2006; Snyder et al. 2009; Wiggelinkhuizen et al. 2007), compared to 10% to 15% on land. Blanco (2009) estimated the O&M contribution to be as much as 30%. In the development of a tool for estimating the cost of wind energy, an O&M cost of 0.007 USD/kWh was used for onshore turbines and 0.020 USD/kWh for offshore (Fingersh et al. 2006). It seems as if the cost of O&M for offshore wind is expected to be approximately twice what it would be on land, but estimates of 3 and 5 times the cost has also been used (McMillan et al. 2007).

More data is available for wind turbines on land. Figure 1.4 show the relationship between frequency and downtime for different types of failures in 600 Swedish onshore turbines based on data from Ribrant et al. (2007). It shows that that most of the frequent failures cause relatively short downtimes. These frequent, minor failures are expected to become more time consuming to repair offshore (McMillan et al. 2007), as waiting time for suitable weather and transportation time must be added to the downtime. A turbine can be down for days or weeks due to a minor failure if it happens at a period when the

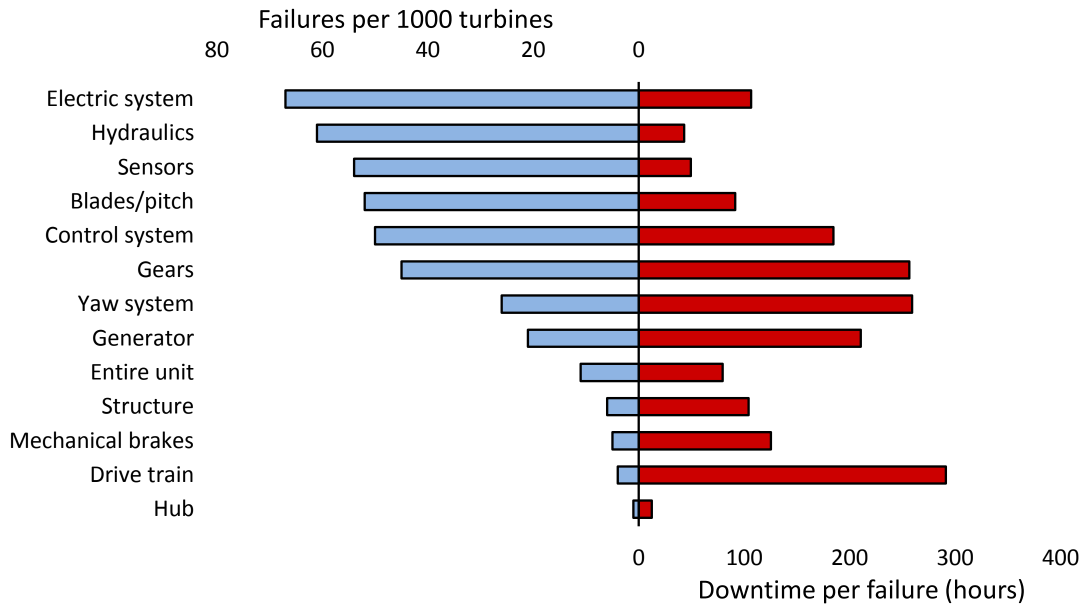


Figure 1.4: Frequency and downtime caused by different types of failures. The data is from 600 Swedish wind turbines on land over a 5-year period (Ribrant et al. 2007).

turbine is inaccessible due to strong wind. This is likely to be a period when the turbine could have operated at maximum capacity, making this downtime especially costly.

For onshore turbines, it has been a low tolerance for increased investment costs to increase the efficiency of O&M (Musial et al. 2006). Although the same pressure to keep the investment cost low exists for offshore turbines, it is more accepted that an increase in the investment might be necessary in order to reduce the need for time consuming and expensive offshore maintenance operations.

The concept of performing maintenance tasks remotely, using a telerobot system, has been evaluated in this thesis. If some maintenance tasks can be performed without needing to access the turbines, there will be an direct economic benefit from reduced cost of transportation to the turbines, and indirect benefits as increased availability because it is possible to do tasks at a shorter notice and regardless of the weather. Not all maintenance tasks are feasible to do remotely, as it would require a too complicated and expensive system. Inspections have been identified as the task that is easiest to perform remotely, and how *remote inspections* perform compared to the alternative of manned inspections has been evaluated in this thesis.

1.2 Main Contributions

In this thesis, the remote inspection concept has been evaluated to determine whether it is a viable alternative to manned inspections for the use in offshore wind turbines. The main contributions from this work are:

- The evaluation of the potential use and advantage of remote inspection and remote maintenance of offshore wind turbines, including simulations with the NOWIcob tool.
- A review of experiments described in the literature that have evaluated the usability of telerobots in other fields.
- A laboratory for testing inspections has been built with visually similar equipment as one might find in an industrial system. The laboratory has several errors that can be visible or hidden for each inspection.
- A series of usability tests that compared the use of remote and manned inspections in the laboratory. No other experiments with a direct comparison between performing a task with a robot or in person have been found in the literature.
- The development of a flexible method for using code generated from Simulink Coder that are easy to combine with manually written code.
- Contributed to the work of Norsk Automatisering AS to get further funding for commercialization and to install and test a remote inspection prototype in a wind turbine.

1.3 Thesis Outline

The reminder of this thesis is organized as follows:

Chapter 2: Operation and Maintenance of Offshore Wind Farms is an introduction to the maintenance of offshore wind farms, and how remote inspection can be used together with the tools and methods used today.

Chapter 3: Prototyping and Evaluation of Remote Inspection for Offshore Wind describes the user-centered development of a remote inspection prototype, including a series of usability tests. This chapter is based on articles G, B, C, E and F.

Chapter 4: Simulink Coder Generated Code as a Module within a Software Project describes a method for using MathWorks Simulink Coder for generating code as a part of a larger software project. This was used in the development of the remote inspection prototype, and is based on articles A and D.

Chapter 5: Concluding Remarks summarizes the work, concludes the thesis and describe the future plans for the project.

Chapter 6: Original Publications contains five published peer-reviewed conference papers as well as two submitted journal paper manuscript.

1.4 List of Publications

The work underlying this thesis has produced the following publications (ordered by publication type, and chronologically numbered):

Journal Papers

Paper F: Ø. Netland, G. Jenssen, A. Skavhaug, "Evaluation of Remote Inspection of Offshore Wind Turbines with a Tablet Controlled Telerobot" submitted to the *IEEE Transactions on Human-Machine Systems*.

Paper G: Ø. Netland, A. Skavhaug, "A Review of Experiments Evaluating the Usability of Mobile Telerobots" submitted to the *IEEE Transactions on Human-Machine Systems*.

Peer-reviewed Conference Papers

Paper A: Ø. Netland, A. Skavhaug, "Adaption of MathWorks Real-Time Workshop for an Unsupported Embedded Platform" published in the *Conference Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications* (SEAA 2010). DOI: 10.1109/SEAA.2010.53.

Paper B: Ø. Netland, A. Skavhaug, "Prototyping and Evaluation of a Telerobot for Remote Inspection of Offshore Wind Farms" published in the *Conference Proceedings of the 2nd International Conference on Applied Robotics for the Power Industry* (CARPI 2012). DOI: 10.1109/CARPI.2012.6473351.

Paper C: Ø. Netland, A. Skavhaug, "Two Pilot Experiments on the Feasibility of Telerobotic Inspection of Offshore Wind Turbines" published in the *Conference Proceedings of the 2nd Mediterranean Conference on Embedded Computing* (MECO 2013). DOI: 10.1109/MECO.2013.6601378.

Paper D: Ø. Netland, A. Skavhaug, "Software Module Real-Time Target: Improving Development of Embedded Control System by Including Simulink Generated Code into Existing Code" published in the *Conference Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications* (SEAA 2013). DOI: 10.1109/SEAA.2013.51.

Paper E: Ø. Netland, G. Jenssen, H.M. Schade, A. Skavhaug, "An Experiment on the Effectiveness of Remote, Robotic Inspection Compared to Manned" published in the *Conference Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics* (IEEE SMC 2013).

NOWITECH Reports

- Ø. Netland, A. Skavhaug, "Condition Monitoring of Offshore Wind Farms - State of the Art Study", 2010.

- Ø. Netland, A. Skavhaug, “Pre-Study on Cost-effective, Remote, Environmental Friendly O&M of Large Scale Offshore Wind Turbine Plants”, 2011.

Other Publications

- Ø. Netland, A. Skavhaug, “Remote Presence: Performing Maintenance of Offshore Wind Farms without Leaving your Office", *29th International Conference on Computer Safety, Reliability and Security SAFECOMP 2010*, Vienna, Austria, Sept 2010.
- Ø. Netland, A. Skavhaug, “Remote Presence, Cost-Effective Robotic Inspection and Maintenance of Offshore Wind Turbines”, *6th EWEA PhD Seminar on Wind Energy in Europe*, Trondheim, Norway, Sept 2010.
- Ø. Netland, A. Skavhaug, “Extending Condition Monitoring of Offshore Wind Farms with Remote Inspection” *24th International Congress on Condition Monitoring and Diagnostics Engineering Management* (COMADEM 2011), Stavanger, Norway, May 2011.
- Ø. Netland, G. Jenssen, A. Skavhaug, “Experimental design of a feasibility study for remote inspection of wind turbines”, *EWEA Offshore 2011*, Amsterdam, The Netherlands, Nov 2011.

Other Presentations

- Ø. Netland, A. Skavhaug, "Remote presence, Operation and Maintenance of Offshore Wind Farms Without Leaving Your Office" poster presentation at the *Deep Sea Offshore Wind R& D Seminar*, Jan 2011.
- Ø. Netland, J. Heggset, "Remote Presence", invited presentation at *NCEI Drift og vedlikehold av offshore vindturbiner - nye muligheter gjennom instrumentering*, Trondheim, Norway, Sept 2011.
- Poster presentations at the *NOWITECH day* in 2011 and 2013. Awarded best poster award in 2013.
- Several presentations at NOWITECH WP5 meetings, both internal and with industry partners.

Chapter 2

Operation and Maintenance of Offshore Wind Farms

This chapter is an introduction to maintenance of offshore wind farms, and a description of how remote inspection compares to other maintenance strategies that are used today.

2.1 Maintenance Strategies

Regular maintenance is important for the safe and reliable operation of industrial systems. There are several strategies for how maintenance is planned and performed. The most commonly used are the following (Rausand et al. 2004):

2.1.1 Corrective Maintenance

Corrective maintenance is repair or replacement after a failure occurs, or "run-to-failure". It is risky and often expensive to rely on corrective maintenance alone. When a component of a turbine fails, this can often cause damage to the rest of the turbine. A typical example would be a failure in a cooling or lubrication system that other systems rely on.

2.1.2 Preventive Maintenance

Preventive maintenance is to repair or replace worn components before they cause failures. This is normally less expensive and time consuming than corrective maintenance. In addition, downtime caused by a failure is avoided. The disadvantage of preventive maintenance is that it could be performed before it is needed, or that a preventable failure can happen before the planned preventive maintenance. Some failures, typically those that are externally caused, e.g. lightning, cannot be prevented with preventive maintenance, as they happen regardless of the condition of the equipment.

Different strategies can be used in order to determine when and which maintenance tasks to perform. The simplest being clock-based or calendar-based, when maintenance is performed at regular time intervals. These time intervals are set based on the expected life of different components. Age-based maintenance is similar to clock-based, but plan maintenance based on how much the system has been used, which often is a better estimate for when maintenance is needed. The clock-based approach, however, is easier to organize, as it is known ahead of time when maintenance is supposed to be performed.

2.1.3 Condition-based Maintenance

Condition-based maintenance (CBM) is a type of preventive maintenance that is performed based on the condition of the turbine. To get information about the condition, a *condition monitoring system* (CMS) is used, which continuously analyze information from sensors in the turbine. The intention is to predict failures early enough that preventive maintenance can be performed before they occur. Maintenance is performed when it is required based on the current observed condition, but not earlier than that.

Jardine et al. (2006) describes three key steps of condition monitoring. A short description of each of them is given here:

Data acquisition obtains data from the system. Data can be acquired *online* or *offline*, where online means that the system is capable of acquiring the data by itself, usually with sensors. Offline means that a person is doing a measurement or analysis and feed the results back into the condition monitoring system. For offshore wind turbines, online acquisition is preferable, as it will not require transportation to the turbines. Many different types of measurements can be used for condition monitoring of wind turbines. The most common measurements are vibration monitoring of the drive train and temperature measurements of bearings and electrical equipment. Lubrication oil analyses provide useful information about the condition of gearboxes and bearings, but it is difficult to do online. Yang et al. (2012) has a comprehensive list of other possible techniques. The different techniques are able to identify different failures, thus a combination of several systems will be necessary.

Data processing analyze the data. The first step will usually be to remove data from faulty sensors or other errors in the data. The further analysis will depend on the type of data. Waveform data, e.g. vibration and acoustic data, can be analyzed both in the time-domain (e.g. comparison with ARMA models) and in the frequency-domain (e.g. fast Fourier transform). Images, from thermographic or normal cameras, can be analyzed with image processing.

Maintenance decision support use the processed data for diagnostics and prognostics. Diagnostics attempt to detect whether faults are present in the system based on the available data. Prognostics attempt to predict the future condition of the system. A commonly used metric for this is the *remaining useful life*. Both diagnostics and prognostics are usually found with statistical approaches, artificial intelligence approaches (e.g. neural networks and fuzzy logic) or model-based approaches. Regardless of the method, the diagnostics and prognostics will not be perfect, thus it will always be a compromise between being able to detect faults early, and the probability that the condition monitoring will create false positive diagnoses or too conservative predictions.

Several research projects, e.g. CONMOV (Wiggelinkhuizen et al. 2007) and Cleverfarm (Giebel et al. 2004), have looked into how CBM can make O&M of wind turbines more effective, especially offshore. Unfortunately, it is not easy to test the effect of CBM. The wind turbines are so reliable that a large number of them running for several years must be observed to gather enough data to see a change in the number of failures. It is also difficult to get data from controlled experiments. Owners of wind turbines are not eager to induce failures in turbines to test how well a CMS is able to detect them.

Besnard et al. (2010) presents results from a life-cycle-cost simulation, which indicates that CBM most likely will have an economical benefit for offshore wind energy, and will reduce the risk of expensive maintenance operations. McMillan et al. (2007) presents a similar simulation, but has focused on the CMS' ability to produce accurate diagnoses. It was estimated that the diagnoses from a CMS should be accurate between 60% and 80% of the time in order to be cost-effective.

2.2 Remote Maintenance

The cost of O&M can be reduced significantly if maintenance could be performed without accessing the turbines, i.e. remote maintenance. This would require a system capable of doing maintenance tasks autonomously or controlled remotely from land. The proposed method in this thesis is to have a remotely controlled robot inside each wind turbine nacelle, as this is where most of the equipment is located. It would also be possible to extend the concept to other areas of the turbine.

The cost of such a system would be dependent on its capabilities. It could be possible to develop a system able to do a large range of maintenance operations, but that would likely be too expensive to be viable for wind turbines. Larger maintenance operations would not be possible to do remotely, regardless of the system, as they would require spare parts that are larger than what could be stored in the turbines.

2.2.1 Remote Inspection

The easiest type of maintenance tasks to do remotely is inspections. An inspection system only need to observe the turbine, not interact with it. A remotely controlled inspection robot that move around the turbine, equipped with cameras and sensors could allow an operator on land to inspect the turbine as if he was there. A concept illustration of such a system moving on a rail inside the nacelle of a wind turbine is shown in figure 2.1. The reason for using a rail for the robot is discussed in 3.3.1.

2.2.1.1 Sensors

Since the operator is not physically present at the wind turbine, he is dependent on the telerobot's sensors for the inspection, and observes the turbine through these. This means that the sensors should be able to provide him with the same information as if he was there. Humans rely on their vision, which also is important during inspections, thus the telerobot should be equipped with one or several cameras. The camera quality is important to get the best view of the equipment and for the feeling of being present in the turbine.

Hearing is also important during inspections, as unusual sounds often are indications of problems. With suitable equipment, a robot system can identify where a sound originates from faster and more accurately than a human could. Sounds can be isolated, both with the use of directional microphones and by using sound processing.

Thermographic cameras are expected to be an important type of sensor for remote inspection. Such cameras are frequently used during manned inspections to search for hot spots in the equipment. Hot spots can be caused by both friction and over-current, which both can be early symptoms of failures. Yang et al. (2012) discuss the possibility of using thermography as part of CM, but because of the high cost of such cameras, it is

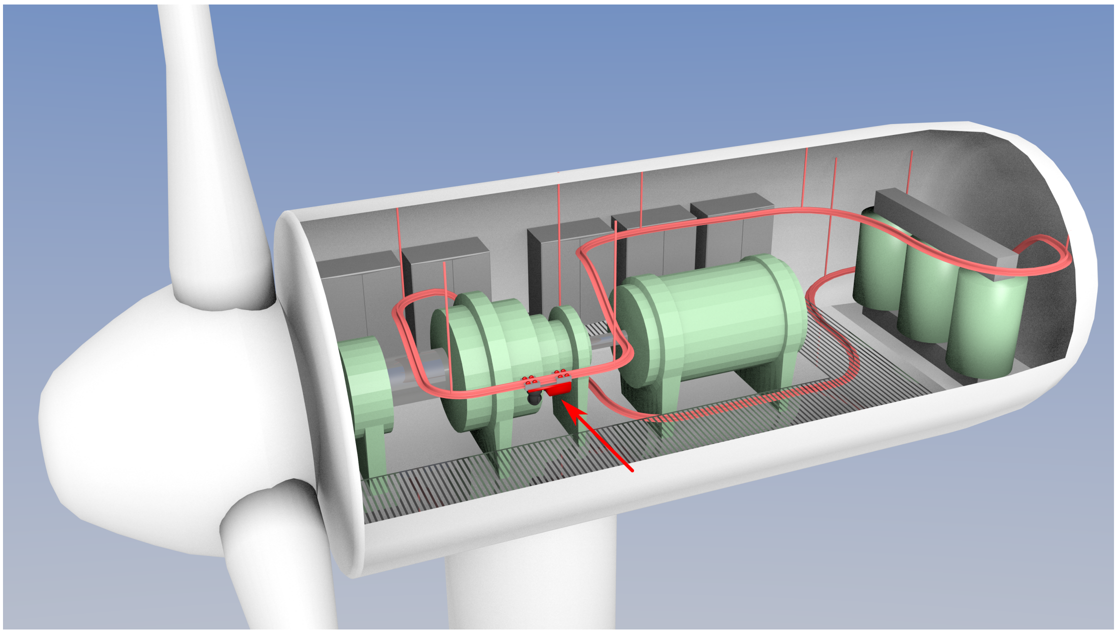


Figure 2.1: Concept illustration of a remote inspection robot inside a simplified and generic nacelle. The nacelle consists of main bearings, gearbox, generator, transformer and cabinets for electronics. The inspection robot is indicated with an arrow. An example rail configuration is also shown.

not considered a viable method. Since an inspection robot can move around, one camera can be used to observe different parts of the turbine, making it more cost effective.

Remote inspections can be performed regardless of weather conditions and while the turbine is running. Both sound and heat are best observed on running machinery. Some sounds will only be present when the machine runs, and a hot spot will slowly dissipate when no new heat is generated. This is an important advantage for remote inspections as turbines normally are stopped for safety reasons when accessed by personnel.

Touching the machinery is to use a human sense that is difficult to replicate with a telerobot. However, it is possible to have a small robotic arm with sensors for detecting temperature, vibrations and possibly a small camera. This can be used similar to a human hand. It will likely be more appropriate to display the measurements on a screen than to induce heat and vibration to the hands of the operator.

2.2.1.2 Remote Inspection and Condition Monitoring

Remote inspection would not be the same as condition monitoring, nor would it replace it. The two technologies have different and complimentary abilities. Condition monitoring operates continuously to produce diagnoses or predictions without human interaction. The intention of remote inspections is instead to replicate how manned inspections are performed, by allowing human eyes, ears and ingenuity to be a part of the system, as if the operator was in the turbine himself.

The diagnoses from condition monitoring are not always accurate, and it is possible that there are false positives. Normally, these would not be detected before a maintenance team investigates the diagnosis on site. With remote inspection, it can be investigated without expensive access to the turbine, thus unnecessary operations due to false positives can be prevented. When a diagnosis is accurate, remote inspections can be used as a tool for preparing and planning maintenance operations.

Remote inspections can also be used to monitor parts of the equipment or look for symptoms that are difficult to find with condition monitoring. In some cases, the sensors on the remote inspection robot can be a replacement for a faulty sensor used by condition monitoring, at least for short durations.

2.2.1.3 Storing of Information

As all information the operator sees is already transferred digitally, it is trivial to store all or parts of this information. In addition, the remote inspection robot can gather information autonomously. Stored information can be useful for future reference, to see how the condition of the equipment has changed over time.

Storing or logging of information during remote inspections can be done without operator intervention. This is in contrast to logging during manned inspections, which will be done manually. This means that the technician will decide what information to register based on subjective observations. Information that was not found important at the time might not be stored.

2.2.1.4 Uses of Remote Inspection

Some of the potential uses of remote inspection, both as a standalone system and together with CMS, are listed below:

- Inspections can be performed at almost no cost, allowing inspections to be performed frequently.
  - Each inspection increase probability that an error is detected.
  - Frequent inspections increase the probability that an inspection is performed after a symptom of an error becomes visible and before it causes a failure.
- Can be used to investigate a failure and plan corrective maintenance. To get correct information early can reduce downtime if spare parts have to be ordered. The personnel can also be better prepared when they have studied the failure beforehand.

- Verify diagnoses from the CMS.
  - False positive diagnoses can be identified as such with remote inspection, before they cause an unnecessary maintenance action.
  - Correct diagnoses can be studied with remote inspection as part of the planning of preventive maintenance.
  - Since the consequence of false positives can be reduced, the CMS can lower its thresholds for giving diagnoses, thus reducing the probability that a failure will go unnoticed.
- A CMS will often have a large number of sensors that both increase the cost of the system and the number of sensors failures.
  - CMS can use information from the sensors on the mobile inspection robot, and possibly reduce the number of sensors installed in the turbine. An example is that a thermographic camera could replace a large number of temperature sensors.
  - Sensors on the inspection robot can be used as an alternative for a failed sensor. Although the sample time and accuracy likely will be lower, it can at least reduce the urgency of replacing the sensor.

2.2.2 Remote Repair and Replacements

It is outside the scope of this thesis to consider how a remote system could perform repair and replacements. However, it could be a possible addition to remote inspection in the future. This would likely be limited to replacing small parts, cleaning, lubrication etc. When considering figure 1.4, it is obvious that the most frequent failures are minor ones that could be possible to do remotely. This means that even the ability to do a limited number of maintenance tasks could significantly reduce the need to travel offshore. It is not feasible to develop a robot that can do large maintenance tasks, at least not at an affordable cost.

2.3 Cost-Benefit Evaluation of Remote Inspection

In NOWITECH work package 5, which covers the topic of operation and maintenance, it has been developed a simulation tool called NOWIcob (Hofmann et al. 2013). It has been developed for analyzing the consequences of different decisions related to the maintenance and logistic strategy of offshore wind farms. It uses a time-sequential event-based Monte Carlo technique to simulate the operational period.

In a co-operation with the developers of NOWIcob, a cost-benefit evaluation of remote inspection for offshore wind farms has been performed. Three simulation cases have been defined. These share the same set of possible failures, their failure rates and what type of maintenance that is required. For larger maintenance tasks, a pre-inspection task is required as part of the planning.

In the base case, there is no condition monitoring nor remote inspection. Preventive maintenance is performed yearly, and corrective maintenance is performed when there has been a failure.

The second case includes a state of the art condition monitoring system that provides warnings about future failures. If condition-based maintenance is performed before the failure occurs, the task will be less expensive and time consuming. However, the condition monitoring system is not perfect, and will not detect all failures. It is also assumed that half of the alarms are false positives and that the sensors of the condition monitoring system can fail and need repair.

The third case has a remote inspection system in addition to the condition monitoring. This means that pre-inspections and investigations of false alarms can be done remotely. However, since remote inspections are considered to take longer, these tasks take twice as long. There are also other potential benefits to remote inspections, e.g. reduction of failures due to inexpensive, frequent inspections. Since these effects are uncertain and difficult to quantify they have not been included in these simulations. A remote inspection system failure has also been added to the list of possible failures.

The investment cost of the turbines has been estimated to be 2,250,000 Euro/MW, with an addition to the cost of 120,000 Euro for a condition monitoring system and 60,000 Euro for a remote inspection system. A wind farm with 100 3MW turbines was used for the simulation. The wind farm was located 40 km from a harbor, a reasonable distance for future wind farms. Each case was simulated with two *crew transfer vessels* (CTV) equipped with advanced systems for accessing the turbines. A jack-up was available and could be leased for periods of 2 weeks when operations that include heavy lifting were required.

For each case, a 20-year simulation was run 20 times. The results are shown in figure 2.2, as the improvement in availability and cost of energy compared to the base case. Relative values have been used to minimize any bias in our assumptions for the simulations. Especially the parameters regarding cost are considered preliminary. The availability are likely a more reliable result than the cost of energy, as it does not rely on

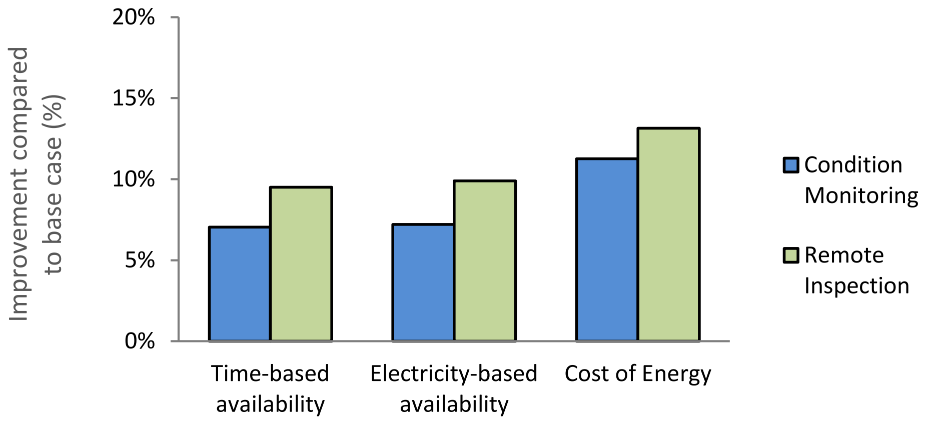


Figure 2.2: The improvements in availability and cost of energy compared to the base case.

any assumptions about the costs.

Both condition monitoring and remote inspection have significant improvements from the base case. This is as expected, as relying on corrective maintenance alone is not considered a viable strategy. It is also clear that the improvements are larger for remote inspection than condition monitoring. When some maintenance tasks are performed remotely, even trivial ones as checking for false alarms and pre inspections, there are more time available to do other tasks and thus reducing the total downtime. The reduced cost of energy is mostly due to less downtime, but there was also a small reduction in the use of both CTVs and chartering of the jack-up, which reduce the O&M cost.

The simulations presented in this paper demonstrate a significant potential economic benefit by using remote inspections. The availability is higher than without remote inspection, and the cost of energy is reduced. The validation of the NOWIcob tool is a continuous ongoing process, but as a tool for comparing different strategies the results should be transferable to the real world.

Chapter 3

Prototyping and Evaluation of Remote Inspection for Offshore Wind

This chapter presents the following work:

- 3.1 presents related work with evaluation of usability of telerobots, and is based on article G.
- 3.2 describes the laboratory used in our experiments, and is based on article B.
- 3.3 describes the user-centered development of the prototype and the different versions used in the experiments from articles C, E and F
- 3.4 presents the results of the performed usability tests and is based on articles C, E and F. Detailed descriptions of each of the experiments can be found in the articles.
- 3.5 discusses the most important results from the experiment. Discussions of the individual experiments can be found in articles C, E and F.

3.1 Related Work with Usability Testing of Telerobots

Traditionally, robotic research has focused on increasing the robots' capabilities, and robotic systems "have been developed by robotics experts for use by robotic experts" (Scholtz 2002). *Human-robot interaction* (HRI) considers the interaction between robots and their users, in addition to the capabilities of the robot. As most users of telerobots are experts in the task the robot is used for, i.e. *domain experts*, they often have less experience from controlling robots. This makes the human-robot interaction especially important.

Human-robot interaction can be considered a subset of *human-computer interaction* (Yanco et al. 2002). The concept of usability is often applied when evaluating computer systems, especially when a system is intended for novice users. Similarly, it can be used when evaluating robot systems where the intended users are not robot experts. Usability is defined in ISO-9241 (ISO 1998) as the "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". *Usability tests* (Dumas 1999) are experiments where potential end users of a system use it for relevant tasks, and the results are used to evaluate its usability.

3.1.1 Usability Goals

Usability goals describe different aspects of the usability of a system. Different definitions of usability goals exist. In the ISO-9241 standard, (ISO 1998) the three usability goals of effectiveness, efficiency and satisfaction are used. Other often-used goals are learnability and memorability (Rogers et al. 2011; Schneiderman et al. 2010), which consider how easy it is to learn and remember how to use a system.

For this thesis, we define and use the same five usability goals as in article G. They are also suitable usability goals for telerobots in general. In addition to the three goals defined in the ISO standard, *workload* (Adams 2002) and *situation awareness* (Endsley 1988) have been added, because of their relevance for telerobots.

3.1.1.1 Effectiveness

The effectiveness is how accurately and completely a telerobot is able to perform its intended task. It is measured differently depending on the task the robot is used for.

- The size of an explored area.
- The ratio of targets found when searching.
- The score in a game like Robotflag.

The participants can also be asked to rate how successful they believed they were. A potential problem is that people cannot be expected to evaluate their own success objectively and accurately.

3.1.1.2 Efficiency

The efficiency is the amount of time or resources spent when using the robot. It is usually measured as the time it takes to complete a task. Efficiency is often linked to effectiveness, as an efficient system allow the user more time for the main task, thus improving its effectiveness.

3.1.1.3 Satisfaction

The satisfaction is the users' subjective assessment of how pleased they were with the telerobot system. It is usually evaluated with a questionnaire or similar.

3.1.1.4 Workload

The workload relates to how demanding, mentally and physically, it is to use the system. It is demanding to perceive and understand a large amount of information from the robot and decide appropriate actions. If the workload becomes more than the user can handle, the performance will decrease while the chance of doing incorrect decisions will increase.

Workload is often evaluated subjectively with the NASA-TLX survey (Hart et al. 1988). Although not specifically targeted at telerobots, this technique is commonly used. Workload can also be estimated with measurements that are expected to correlate with it.

- The amount of commands given by the participants (joystick use, number of keys pressed etc.) is an indicator of the effort and the workload of using the system.
- A secondary task can be given (answering math questions, simple computer game etc.), and the performance of the secondary task indicates the workload from the primary task. High performance means low workload.

3.1.1.5 Situation Awareness

Situation awareness (SA) refers to the robot user's ability to be aware of, and understand the situation of the robot. A definition of SA is "the perception of the robots' location, surroundings, and status; the comprehension of their meaning; and the projection of how the robot will behave in the near future" (Yanco et al. 2004a). The user is not physically present at the robot's location, and cannot observe it directly, thus it becomes difficult to maintain situation awareness. In several real-world search and rescue operations, it was estimated that the robots were stationary about half of the time because their operators tried to understand what was going on around them (Murphy et al. 2005).

Situation awareness measurements can be grouped into three categories; explicit, implicit and subjective (Yanco et al. 2004a). Subjective measurements are self-reporting of SA, like 3D SART (Endsley et al. 1998). Implicit measures are measurements that are assumed to correlate with SA, as the number of collisions. A high number of collisions or near collisions is expected to correlate with low situation awareness.

Explicit measures consist of pausing the experiment to quiz the participants about their knowledge of the location and situation of the robot. This is an accurate sample of the participants' awareness at a specific time in the experiment. Unfortunately, explicit measures will interfere with the rest of the experiment.

3.1.1.6 Qualitative Results

Some articles present observations that were done by the experimenters or suggestions by the participants. These qualitative results are useful for identifying problems and to get ideas for improvements, which is important when applying a user-centered design (Adams 2002; Beaudouin-Lafon et al. 2003). It is assumed that qualitative results often are used for this, even if it is not mentioned in an article. Qualitative results from participants that have experience with the tasks the robots are intended for are especially valuable.

3.1.2 Relevant Usability Tests for The Development of Remote Inspection

The experiments and findings that are presented here is a selection that are considered relevant for the prototyping and evaluation of the remote inspection system, from the full review in article G. The consequences of these findings for remote inspection are discussed in the relevant parts of 3.3. In article G, the experiments have also been categorized and summarized in tables.

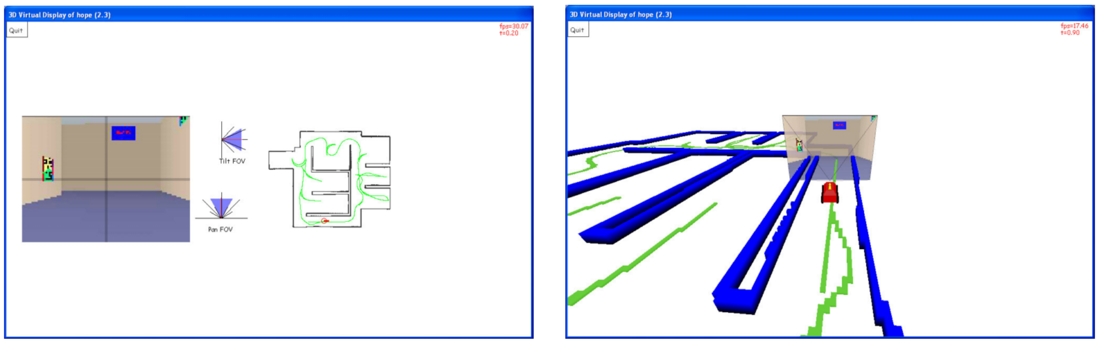
3.1.2.1 Displaying Information

The user of a telerobot is not able to see the robot or its environment. Both the task the robot is used for, and the control of the robot are dependent on information from the sensors on the robot. How it is displayed to the user is thus important for the telerobot usability.

Most user interfaces for telerobot control will display either video streams from cameras on the robot, a map of the robot's environment or both. Several experiments have been performed to compare different variations of this. For robots used to explore an unknown environment, it is common to use information from distance sensors to build a map of the robot's environment. These maps can be normal 2D maps where the user sees the robot and its environment from above. The map information can also be displayed in three dimensions from the perspective of the robot or slightly behind, which is called virtual reality.

It seems that "there is useful navigational information, in both the map and the video sets of information, integrating the information can yield better results than using either map or video individually." (Nielsen et al. 2006a) However, in experiments were a video display and a map display have been shown side-by-side (*video+map*), as shown in figure 3.1(a); it has not performed better that the best of the video or map alone (Nielsen et al. 2005; Yanco et al. 2006; Yanco et al. 2007). This implies that the users only managed to use one of the displays at the time.

Augmented reality (Ricks et al. 2004) is a method for *fusing* a video into a 3D representation of the map, as shown in figure 3.1(b). The two displays shown in figure 3.1 are showing the same information, but it is easier to pay attention to all the information


3D Virtual Display of hope (2.3)
Quit
Tilt FOV
Pan FOV
3D Virtual Display of hope (2.3)
Quit

(a) A video+map display. (b) An augmented reality display.

Figure 3.1: Examples of different types of displays from Nielsen et al. (2005).

in the augmented reality display. Several experiments (Nielsen et al. 2005; Ricks et al. 2004) have shown that this was beneficial, especially for situation awareness.

There are also simpler methods for fusing information, e.g. to show distance indicators together with a video. These indicators can be organized around the video display (Eliav et al. 2011), overlaid on the video as a HUD (Eliav et al. 2011) or as a separate display besides the video (Yanco et al. 2007). Experiments (Eliav et al. 2005; Eliav et al. 2011) suggests that the HUD solution is preferable. This is as expected, since the user can notice changes in the symbols overlaid the video without looking away.

3.1.2.2 View Perspectives

One of the potential advantages of virtual and augmented reality interfaces was identified as the ability to view the robot and its environment from a third person perspective. To test whether this actually is an advantage, one should compare two interfaces that are equal except the view perspective. We have identified two experiments that did such a comparison. The results showed that a third person perspective was an advantage for both video (Keyes et al. 2006) and map (Bruemmer et al. 2006) based interfaces. Especially situation awareness was improved, probably due to the ability to see the robot within its environment, which makes it easier to see how far it is from obstacles.

Telerobots with cameras often have pan/tilt/zoom (PTZ), which allow the user to turn the camera and look around without moving or turning the robot. This has a positive effect on the usability, especially situation awareness (Hughes et al. 2004; Nielsen et al. 2006b). For many robots, it is also faster and more convenient to turn the camera than the robot.

A potential disadvantage with PTZ cameras is that users can be unaware of the direction the camera is facing. This can cause confusion and loss of situation awareness, especially if the user incorrectly assumes that the camera is centered when moving (Hughes et al. 2004; Yanco et al. 2004b). It is important to communicate the direction of the camera to the operator. To make it easy to center the camera, some systems have a button that does this automatically (Baker et al. 2004).

3.1.2.3 Levels of Autonomy

The operator of a telerobot has to make due with limited information about the robot's environment, and there will often be some communication delay. This makes the control of telerobots a challenging task, which can be confirmed by the evaluation of robot search and rescue during the World Trade Center disaster (Casper et al. 2003). In this real-world situation, two operators were needed to control each robot, which is an indicator of how difficult this can be, especially in a stressful situation.

The *level of autonomy* describes to which extent a system, like a telerobot, is able to operate without the interaction of a human. The level of autonomy has been shown to have an effect on both situation awareness and workload in other applications (Endsley et al. 1999), thus it is likely that different levels of autonomy for telerobots will have an effect on these and other usability goals.

Both computer and human control have their benefits, and the optimal compromise between the two are likely to be dependent on the current task, robot system etc. Parasuraman et al. (2000) start their article with the following question: "Technical developments in computer hardware and software now make it possible to introduce automation into virtually all aspects of human-machine systems. Given these technical capabilities, which system functions should be automated and to what extent?".

Sheridan (1992) introduced a scale of ten levels of automation, from the operator controlling the robot directly; to an autonomous control system that completely ignores the operator. These can be applied to different stages of information processing and decision making to describe a system. Although it would be possible to describe telerobot systems with this method, we have instead grouped the most common control schemes. These are ranged from low to high autonomy.

Teleoperated is when the operator has full and direct control over the robot, typically controlled with one or multiple joysticks.

Safe mode is the same as teleoperated, except that the control system will intervene to prevent collisions and similar.

Waypoint control lets the operator specify waypoints the control system navigates between. If the robot has a camera, it is often controlled separately.

Shared mode means that the control is shared between the user and the control system. How the control is shared varies between implementations.

- User initiative or adjustable autonomy allow the user to adjust how the control is shared between the control system and itself.

- Robot initiative or adaptive autonomy allow the control system to adjust how the control is shared between the user and itself.
- Mixed initiative is a mix of user and robot initiative, where both can adjust the how the control is shared.

Autonomous control systems are in full control over the robot. The operator does not interact, or interacts very little with the robot.

For experiments with real robots, safe mode is normally used instead of teleoperated, to reduce the chance of the robot damaging itself or its environment. In an experiment that compared the two for a real robot (Marble et al. 2003; Marble et al. 2004), the participants felt most in control when using safe mode, as they did not need to worry about colliding.

Higher autonomy (waypoint and shared mode) assists the operator, and the results show that participants spend less of their time interacting with the robot than when using teleoperation or safe mode (Bruemmer et al. 2008; Crandall et al. 2002; Few et al. 2008; Nielsen et al. 2008) and performed better on secondary tasks (Crandall et al. 2002). These are indicators of lower workload, and means that the operator spends less of their time navigating the robot and more time on their main task, e.g. searching for targets. Experiments show that higher autonomy correlates with higher effectiveness in such tasks (Bruemmer et al. 2003; Bruemmer et al. 2005; Bruemmer et al. 2008; Few et al. 2008; Goodrich et al. 2007; Nielsen et al. 2007; Nielsen et al. 2008).

3.1.2.4 Touch Screen Interfaces

Most telerobots are controlled from desktop computers with keyboard, mouse and sometimes joysticks. Joysticks are especially useful for teleoperation, as it provides intuitive and accurate control. The versatility of the joystick as an input device is demonstrated by Song et al. (2007). The most common alternative to keyboard, mouse and joysticks are touch screens, which have become increasingly popular with the increased use of smart phones and tablet computers. When a large touch screen interface was compared to a traditional and well-tested joystick based interface (Keyes et al. 2010), there was no significant difference in usability. When different ways to control a robot with a smart phone was compared (Truar et al. 2012), the best results were achieved when using the touch screen.

A touch screen is a “blank canvas on which control surfaces are dynamically created” (Micire et al. 2009b), thus it can be tailored to the specific application. There are *limitless* interaction methods, which also is a challenge as it can be difficult to choose the most intuitive touch gesture for a specific command. This was studied by Micire et al. (2009a), where 31 participants were asked to describe how they would prefer to interact with the touch screen to control robots to do specific tasks. An overview of the popularity of the different types of gestures for different commands was presented. These results are a good starting point for defining which touch gestures to use for a touch based robot control system. Similarly, six participants were observed while controlling a robot with a touch interface (Micire et al. 2009b). The behavior of each was described in detail, and it was found that they developed their own individual interaction styles that varied more than the experimenters had anticipated.

3.2 The Laboratory

To evaluate remote inspection we have built a laboratory consisting of generic industrial equipment as seen in figure 3.2. The equipment is there to be observed, not used, so it only needs to be visually similar to industrial equipment to be a sufficiently realistic *mock-up* of a wind turbine. That the equipment has little in common with the equipment in a wind turbine is not relevant. The laboratory can be inspected both in person and remotely, allowing a direct comparison between the two inspection methods.

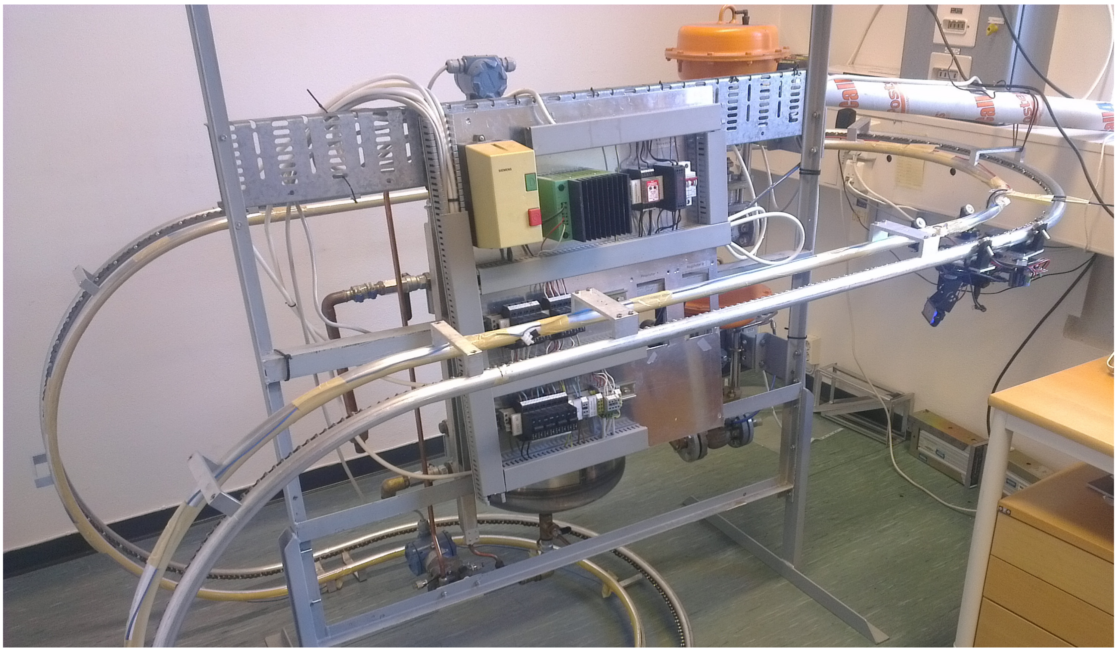


Figure 3.2: A section of the lab for evaluation of remote inspection

In the laboratory, we have defined several *targets* that the participants should search for during inspections. These can be added and removed from the laboratory between each inspection, which allowed us to be in control of the laboratory and the targets.

Evaluations in a laboratory, where the experimenters have full control, are more suitable for experiments than an actual wind turbine. Wiggelinkhuizen et al. (2007) described the problem of relying on naturally occurring failures for evaluating the ability to detect them. In the CONMOW project, condition monitoring was tested on five turbines for two years with no major failures. Other data of interest was collected, but this demonstrates a problem with testing a prevention system by waiting for naturally occurring failures and events. Other advantages of doing experiments in laboratories are that there are less safety concerns, and it is less time consuming to do experiments.

3.2.1 The Targets

Two types of targets were used in the laboratory. 12 *error markers* that represent errors in the equipment that are unknown to the participants were defined. In addition to the error markers, approximately 20 locations in the laboratory were used to hide *paper clips*.

The error markers were designed to resemble actual errors, and to be as realistic as possible, but due to the requirements of the experiments, realism was not the only criteria. It was considered more important that the participants, with their lack of experience from inspections, were able to recognize the error markers. Thus, the error markers were designed so most people would understand that maintenance would be necessary. Since the experiments will consist of several inspections performed in sequence, each with different error markers visible, they also had to be easy to add to and remove from the laboratory.

The paper clips were used as a known object for the participants to look for. They can resemble a known symptom or pattern that an inspector will actively search for. The shape of the paper clips made it easy to attach and detach them to the equipment.

The laboratory was built to compare the probability that an error is detected with remote and manned inspections, to demonstrate whether remote inspection is a possible alternative. The targets were therefore designed to be detectable for both inspection types. The laboratory was not designed to evaluate whether remote inspection is capable of detecting the same errors as manned inspection, as this would be meaningless unless the errors were realistic and the participants had the appropriate experience. For the experiments presented in this thesis, we assume that a remote inspection robot, intended for use in an actual wind turbine, can be equipped with sensors capable of detecting the same errors as maintenance personnel on site.

3.2.2 Measurements in the Laboratory

Based on the usability goals presented in 3.1.1, the following methods for evaluating the usability of remote inspection was used in one or more of the experiments performed in the laboratory.

3.2.2.1 Effectiveness

- The detection rate for error markers measure the effectiveness for identify errors with *unknown* symptoms
- The detection rate for paper clips is a secondary measurement of the effectiveness, thus it is considered less important. The paper clips represent errors with symptoms that are *known* to the participants before the inspection.

3.2.2.2 Efficiency

As remote inspections do not require personnel to travel for accessing the turbines, it is acceptable that the inspections themselves take longer to perform. Therefore, it was not a priority to measure the efficiency, and the temporal workload component in the NASA-TLX survey is the only related measurement. The participants were given more time for remote inspections than manned, as it was expected to be less efficient.

3.2.2.3 Satisfaction

- A Van der Laan survey (Van Der Laan et al. 1997), which evaluates the subjective satisfaction of the participants and the general usability of the system.
- Comment text field in final evaluation.
- Informal interview after inspections.

3.2.2.4 Workload

A NASA-TLX survey (Hart et al. 1988) is used to get the participants' subjective evaluation of their workload.

3.2.2.5 Situation Awareness

There are no specific measurements for situation awareness. Due to the relative small size of the laboratory, and that the robot moves on a predetermined path, situation awareness is considered less relevant.

However, this is expected to be a larger problem in an actual wind turbine, where the environment is larger and cluttered with equipment. Evaluation of situation awareness would be important in such a setting.

3.2.3 Types of Usability Tests

Different types of usability tests are performed for different reasons. Three different types that can be used for evaluation of remote inspection, and other similar concepts, are described in table 3.1.

Two pilot experiments and two larger quantitative experiments have been performed and are described in this thesis. The pilot experiments gave some preliminary results, but more importantly they evaluated the laboratory and the test procedure so the following experiments could be improved, which is discussed in 3.5.1. The two quantitative experiments had 21 and 31 participants, and thus the results are more reliable than in the smaller pilot experiments. These are referred to as the first and second experiment.

Pilot experiment	
Purpose • Evaluate the lab, error markers and the experiment procedure. • Identify problems with the prototype.	**Defining Characteristics** • Few participants, no special requirements. • Provide only preliminary results. • Mainly an evaluation of the experiment itself, not the telerobot.
Quantitative experiment	
Purpose • Compare remote and manual inspections to determine whether remote inspection is a viable alternative. • Can also compare different variations of remote inspection.	**Defining Characteristics** • Many participants (20+), not realistic that they should have experience from maintenance. • Each participant perform a number of inspections using both remote and manned inspection. • Each error marker should be used the same number of times with each tested inspection method. • Participants should not be interacted with or asked to think out loud, to avoid influencing the results.
Qualitative experiment	
Purpose • Find usability problems in the prototype. • Get ideas for improvement. • Get subjective comments about the prototype and concept.	**Defining Characteristics** • Few participants, preferably with maintenance experience. • Observations of the participants and their comments are more important than their performance. • Remote inspection can be tested alone or compared with manned inspection. • Less important to have a controlled experiment with accurate quantitative measurements.

Table 3.1: Descriptions of usability test variants for evaluation of remote inspection.

As the laboratory and experiment procedure were designed for participants without experience from maintenance, it was not a priority to recruit maintenance personnel for qualitative experiments.

3.3 The Remote Inspection Prototype

3.3.1 Physical Description

The same physical prototype, shown in figure 3.3, has been used throughout the experiments described in this thesis. A short description of the choices for the physical design is given here.

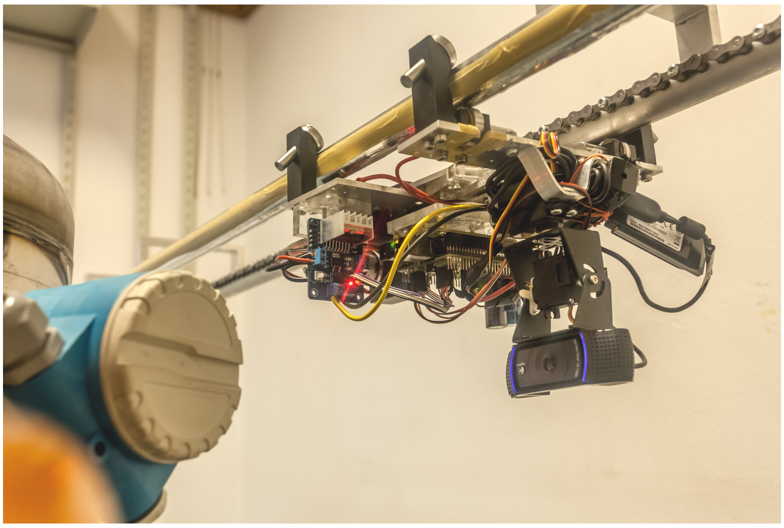


Figure 3.3: The prototype remote inspection robot used in the experiments.

The advantages and disadvantages of a robot that move on a rail are compared with a freely moving robot in table 3.2. We have decided to use the rail solution, because a freely moving robot was considered unnecessarily complex and expensive for our application. Movement on a rail is a low-cost and simple method for getting the robot up from the floor and close to the equipment. The rail can be customized for the individual nacelle so the robot is able to reach all points of interest.

The prototype uses a rail consisting of two aluminum pipes, which it grips to as a roller coaster. It is not possible for the robot to fall off the rail, which could damage the turbine or itself. This is considered an important safety feature. The design was created with simplicity and low cost in mind, and built at the mechanical workshop at our department. Due to inaccuracies from bending the pipes, we experienced some problems with the robot, and it was not able to use the whole rail. The experiments were designed so this limitation would not have any significant effect. For future prototypes, a more reliable rail system should be used. For commercial applications, it will be important that the rail is suitable for mass production and easy installation.

A bicycle chain was attached to the side of one of the pipes. The robot pulls itself

Movement on rail	
Advantages • Simple and reliable. • Can be powered through rails. • Easy to know exact position. • Not dependent on the environment to move.	**Disadvantages** • Can only reach where the rail goes. • Require installation of rail. • Where the robot can move is decided when installing the rail.
Free movement	
Advantages • Can potentially reach everywhere.	**Disadvantages** • Obstacles limit movement. • Reach higher ground (climbing) is difficult. • Needs batteries for power. • Can get stuck or fall.

Table 3.2: Comparison between movement on rail and full mobility.

forward with a cogwheel on this chain, in a simple rack and pinion solution. Such methods are often used for railways with steep gradients, and allow the robot to move on the rail even if it is vertical. This is an important ability, as it is desirable to have as few limitations to how the rail can be designed inside the wind turbine as possible.

3.3.2 Control System

While the physical prototype remained the same for all the experiments, the control system was improved between each experiment based on comments from the participants, i.e. a user-centered development. A summary of how different parts of the system evolved between the experiments are shown in table 3.3. The control system consist of three parts, the robot control computer running on the robot (3.3.2.1), the user client (3.3.2.2) and the control methods (3.3.2.3 and 3.3.2.4).

3.3.2.1 Robot Control Computer

The robot prototype is controlled with a Beaglebone development board with an ARM processor. The Beaglebone is described in detail in 4.2.1. The communication between the Beaglebone on the robot and the rest of the system is over wireless network communication. For use in a wind farm, it is assumed that there is a wireless network inside

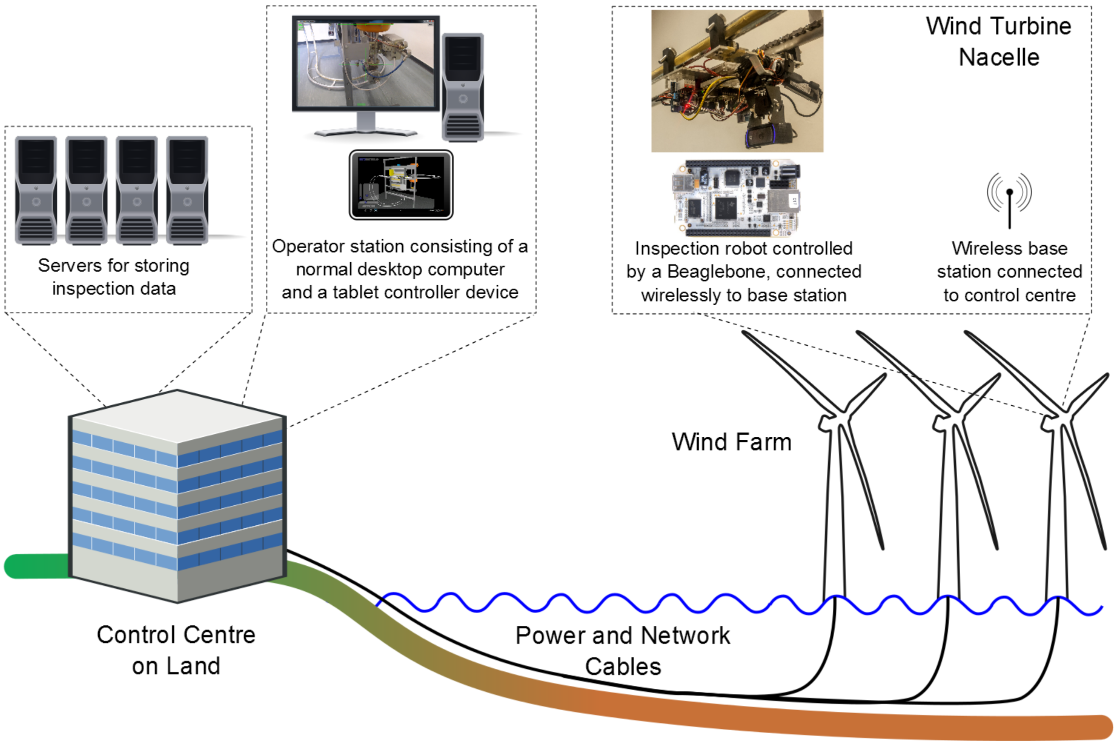

Servers for storing inspection data
Operator station consisting of a normal desktop computer and a tablet controller device
Inspection robot controlled by a Beaglebone, connected wirelessly to base station
Wind Turbine Nacelle
Wireless base station connected to control centre
Wind Farm
Control Centre on Land
Power and Network Cables

Figure 3.4: A description complete control system as intended when installed in a wind turbine.

each nacelle, and a wired connection from the turbine to land, as illustrated in figure 3.4.

For the first experiments, the Beaglebone ran an Angstrom Linux distribution, which is the default operating system for the board. For the last experiment, an Ubuntu distribution patched with Xenomai was used instead. Ubuntu allowed for hardware floating-point arithmetic, and Xenomai provide hard real-time execution of tasks. Although not strictly necessary, as the Beaglebone have abundant computing resources for this application, it is expected to improve the performance and reliability of the system.

The robot is equipped with a motor, with a rotary encoder, for moving along the rail. The rack and pinion solution makes it impossible for the robot to spin, thus the encoder is an accurate measurement of the robot's relative position and speed. The position was reset each time the robot reached the start or end of the rail, by using infrared line detectors to detect markings on the rail. The camera is a Creative 1080p web-camera on a pan and tilt mechanism that was moved with servomotors. All motors and sensors were connected directly to the I/O pins on the Beaglebone or through some simple passive electronic components. The camera connected through USB, and since the encoding was done on the camera, it does not load the CPU of the Beaglebone.

Experiment	**Control method**	**GUI**
Pilot	Teleoperated using a gamepad controller. Movement control is relative to the rail.	Split screen with 2D map and control on the side. Crosshair for camera pan and tilt are overlaid the video.
First	Teleoperated using a gamepad controller. Movement control is relative to the camera direction.	Full screen video. 2D map and a square indicating camera pan, tilt and zoom are overlaid the video.
Second	Assisted control using a tablet.	Full screen video. Crosshair for camera pan and tilt and indicators for robot position and camera zoom are overlaid the video. 3D map and other views on the tablet screen.

Table 3.3: The evolution of the prototype versions.

Pan and tilt for the camera of a telerobot have been found to be beneficial in several experiments, as described in 3.1.2.2. As our prototype moves on a rail, the pan and tilt becomes even more important as the robot cannot turn itself.

For the two main experiments, the robot control program consisted of a combination of code generated with Mathworks Simulink Coder and manually written code. The Simulink model that was used is shown in figure 3.5. The method for including generated code into a normal code project was developed for the prototype and was called *Software Module Real-time Target* (SMRT). Since this method can be used in other applications than for our prototype, it has been described in detail in chapter 4, and is available online under the LGPL v3 license.

The control system can control either the speed or position of the robot and the camera's pan and tilt based on commands from the connected user interface. Sensor information, camera video stream and the status of the robot are sent back to the user interface. The control program acts based on the commands from the user interface. If the connection is lost, the control system stops the robot at its current position.

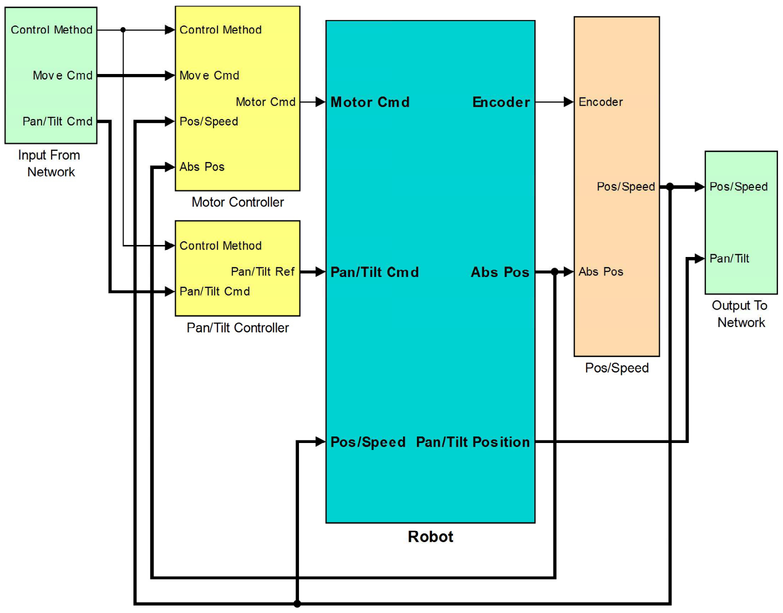


Figure 3.5: The Simulink Model used as part of the robot control system during the second experiment.

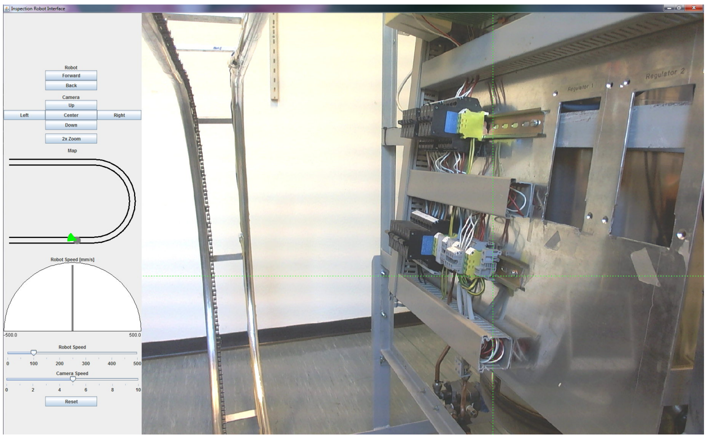
3.3.2.2 User Client

How the look of the user client has evolved between the experiments is shown in figure 3.6. All versions of the user interface share the same focus on displaying the video stream. This is because it is what the participants use to identify errors, thus it is preferable that it is shown as large as possible. This is also the main reason for not using an augmented reality type display that was described to be beneficial in 3.1.2.1. For the two main experiments, the video was shown in full screen.

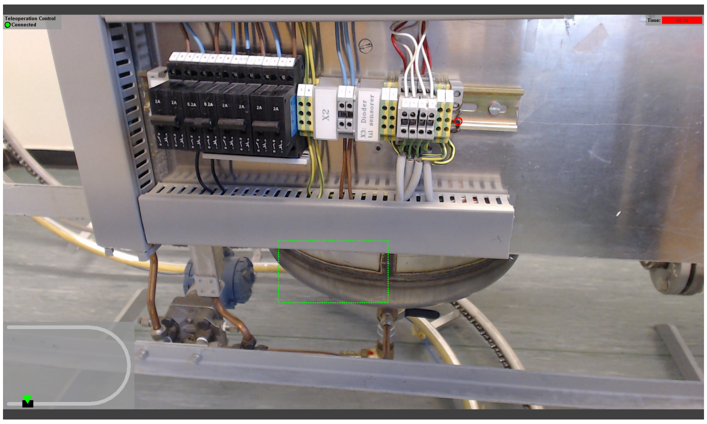
In the pilot and the first experiments, a simple 2D map of the robot and its environment was used, as seen in figures 3.6(a) and 3.6(b). Either shown on the side of the video or overlaid on the corner of the video. As this map was considered too simple for describing the robots position in a complex environment, as a wind turbine, a 3D map solution was developed for the second experiment. It was not shown on the main computer, but instead on the tablet controller as seen in figure 3.7(a). The reason for this is that the map would have taken a significant proportion of the main screen.

All versions have green HUD lines overlaid the video stream. These are indicators of camera position, zoom etc., and are meant to improve the participants' situation awareness. These were not evaluated during the experiments, and except some minor design changes, they remain the same throughout the experiments.

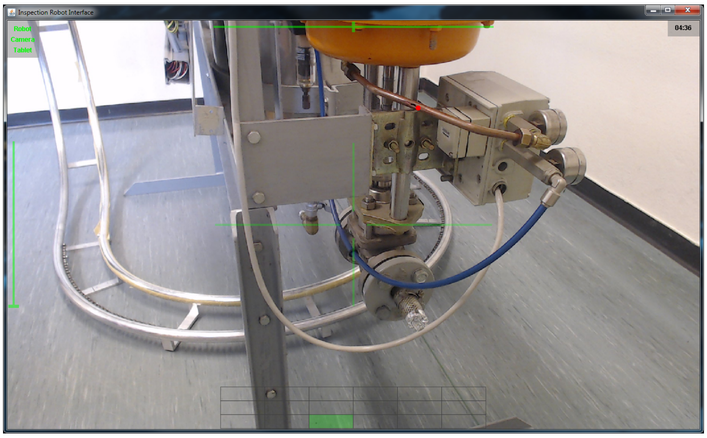
The client was developed with Java, using a *Model-View-Controller* (MVC) (Reen-



(a) GUI used in the pilot experiments.



(b) GUI used in the first experiment.



(c) GUI used in the second experiment.

Figure 3.6: The evolution of the GUI.

skaug 1979) method. It is a modular design method, so different parts of the system can be modified or replaced. The user client used in the experiments will likely be one part of a larger system for controlling several remote inspection systems within a wind farm, thus the modularity of the software is important for future extensions. The client connects to a server running on the Beaglebone on the robot, and communicates via UDP messages for minimal latency. For the second experiment, a simple checksum system was implemented to verify that the correct content was received in the UDP packets.

3.3.2.3 Teleoperated Inspection

The inspection method used in the pilot and first experiments were named teleoperated inspection. The participants controlled the robot directly with two joysticks on a gamepad controller. For both experiments, the robot speed was controlled with the left joystick and the camera with the right.

In the pilot experiments the robot speed was controlled relative to the rail, meaning that forward and backward on the joystick would move the robot forward and backward on the rail. This was considered confusing, as the robot spent most of its time with the camera looking to the side. For the first experiment, this was changed to controlling the robot relative to the direction of the camera. The user's joystick movement will then correspond to the robot's movement when seen from the user's perspective.

3.3.2.4 Assisted Inspection

The intention of assisted inspection is that the control system should assist the users to do better inspections. The first part of this is to have a higher level of autonomy for the control of the inspection robot. Instead of having direct control of the robot, the user moves it between pre-determined *observation points*. For other telerobots, higher levels of autonomy tend to increase the effectiveness and reduce the workload (Bruemmer et al. 2003; Bruemmer et al. 2008). As this robot has fewer degrees of freedom than a freely moving robot, the control system for the increased autonomy is relatively simple. However, the same benefits of autonomy are expected, e.g. control of the robot should be easier for the participants, so they can focus more on their inspections.

The concept of assisted inspection was first tested in the first experiment, as an interactive simulation (Beaudouin-Lafon et al. 2003). This means that a computer program that simulated how the robot would have worked was used instead of the actual robot. The video shown to the users was from the robot's camera, but it had been pregenerated, and was not *live*. Assisted inspection using the robot was implemented for the second experiment, which is the version described here. In addition to the higher level of autonomy, a tablet computer was introduced as an input device. This was a Samsung Galaxy 10.1 tablet running Android. As described in 3.1.2.4, touch screens have the benefit of being highly customizable as an input device. The gamepad on the other hand, was suitable for teleoperated control, but was less suitable for the assisted inspection control.

The observation points used in the second experiment were determined by defining six robot locations on the rail. These were found based on how all parts of the laboratory could be covered with the fewest number of locations, and did not consider where the targets would be. For each of the locations the camera could look upwards, straight or downwards. This means that there were a total of 18 observation points. This gave the participants on average 17 seconds on each observation point, when not subtracting the time for moving between them, thus the available time was limited. The users could move between these locations by swiping up, down, left or right on the tablet computer.

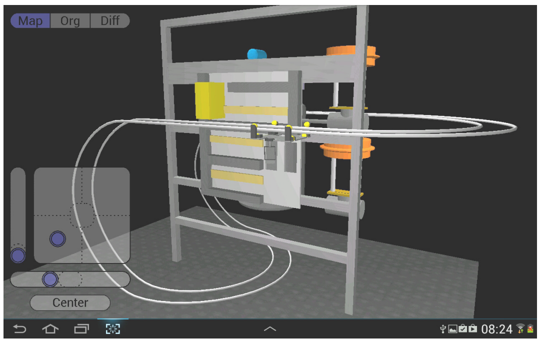
In addition to being a useful input device, the tablet functioned as a *second screen* of the interface. Similar concepts have been tested for interactive television (Cruickshank et al. 2007) for displaying electronic program guide etc., and more recently for game consoles (Carey 2012). The Nintendo has the Wii U controller with a touch screen and Microsoft has the SmartGlass smart phone app for interaction with the Xbox 360.

It can be difficult to pay attention to two displays on the same screen at the same time (Keyes et al. 2006), thus it is likely even more difficult to pay attention to two different screens. This implies that we cannot expect the users to notice any information presented on the secondary tablet screen. Therefore, we limit this screen to display information that can assist the user but are not essential. The user can check the information at his convenience, but can otherwise ignore it. During the second experiment, the participants could switch between three views on the tablet computer:

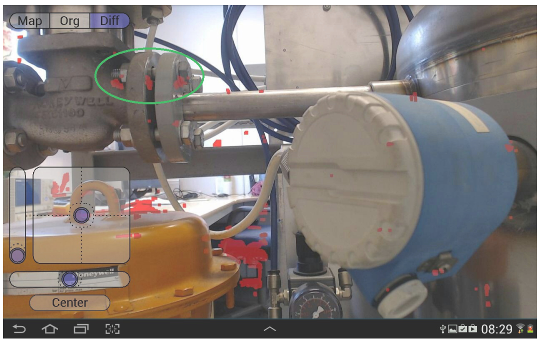
3D Map View shows a 3D model of the robot's environment seen from behind it, as seen in 3.7(a). Most control systems for telerobots have a map of the robot's environment. For search and rescue robots, such maps are typically built based on information from distance sensors. For remote inspection of wind turbines, it is easier to attain a map of the robot's environment, since the layout of the turbine is known. For this view, we assume that a 3D model of the turbine interior is available. An operator of remote inspection is less dependent on the map for navigation, as the robot moves on a rail, thus it is considered justified to have the map on a secondary screen.

Original View shows an image taken from the robot's current position when the laboratory was in original condition, i.e. no visible targets. It takes advantage of the possibility to collect and store historical information gathered by the robot. In a real system, the operator would be able to browse through multiple versions of the same image and observe the condition change over time. Although it is possible to store images from previous manned inspections as well, this cannot be automated in the same way, nor be an integrated part of the inspection system.

Difference View is an extension of the original view, where image analysis is used to compare the original image with the current camera image. A background subtrac-



(a) Tablet showing a 3D map of the laboratory.



(b) Tablet showing a difference view. Red marks difference between previous and current image. The bolt within the green oval is missing in the camera image and the three regions of red are due to this.

Figure 3.7: Different views for the tablet controller. Controls for manual adjustments are shown in the lower left corners.

tion algorithm (Piccardi 2004) implemented with OpenCV (Bradsky et al. 2008) was used. A typical example of how this would be seen by the users is shown in figure 3.7(b).

Due to the inaccurate position and camera direction, it was not always possible to get an acceptable match. This problem was reduced by dividing the image into an 8 by 5 grid, where each cell was translated (moved) on the original images to get the best possible match before doing the background subtraction. This was not an optimal solution, as it was inaccurate and computationally demanding, but was good enough to evaluate the concept. Each image comparison took between 1 and 2 seconds, and ran continuously. This means that the red markings on the tablet image were updating every other second. Together with time for the camera to focus and adjust light, it typically took 5 seconds from arriving at an observation point until a usable difference image was shown.

To improve the image matching, it would be necessary to use other image alignment methods (Szeliski 2006) that can do rotations, perspective transform etc. in addition to comparing two flat images. If the robot position and camera direction were more accurate, it would also be easier to compare the images.

3.4 Results

A short description of the individual experiments and their results are presented here. The experiments are described in detail in articles C, E and F. All error bars that are shown in the graphs of this section represent 95% confidence intervals.

3.4.1 Pilot Experiments

Two pilot experiments with 4 participants each were performed. The participants did two inspections, one remote and one manned. Two groups of targets, with 4 error markers and 4 paper clips each, were defined, and these were rotated between the inspections. The participants had 4 minutes for each inspection.

The second pilot experiment differed from the first, as the task of looking for paper clips was not used. The participants were also allowed to test the remote inspection system before their inspections. This was not permitted in the first.

3.4.1.1 Errors and Paper-Clips

The results from the experiment are seen in figure 3.8. Due to the small number of participants, these results are considered preliminary.

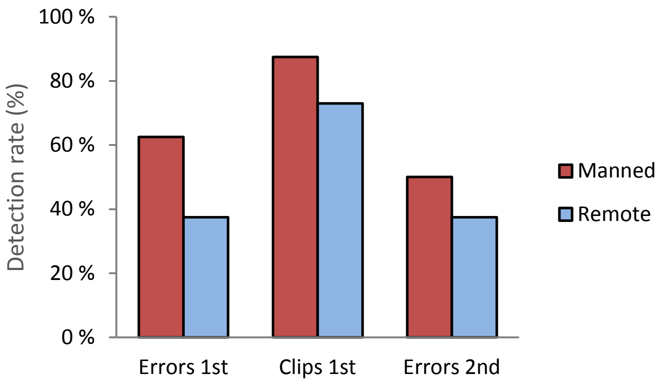


Figure 3.8: Results from the pilot experiment. Error bars are not shown due to the low number of participants.

3.4.2 First Experiment

The first larger experiment was performed as a quantitative usability test, where 21 participants did 9 inspections each, a total of 189 inspections. The number of targets in the inspections was randomly assigned, with between 0 and 2 error markers and 1 and 4 paper clips for each inspection.

Three different inspection methods were tested, teleoperated remote inspection, simulated assisted remote inspection and manned inspection. Each participant performed 3 inspections of each type in a randomized order. The participants had 3 minutes for manned inspections and 4 minutes 30 seconds for the remote.

3.4.2.1 Errors and Paper-Clips

The combined number of error markers each of the participants found, was used to calculate their detection rate for each of the three methods. The same was done for the paper clips. The average detection rates are shown in figure 3.9. The detection rate of teleoperated remote inspection is 68% of the detection rate of manned inspection for error markers, and 90% for paper clips.

The results were analyzed with a one-way ANOVA with an alpha value of $.05$ and a Tukey HSD post test, which found the following:

- No significant difference for error markers ($F(2,60) = 0.64, p = 0.53$).
- No significant difference for found paper clips ($F(2,60) = 1.49, p = 0.23$).

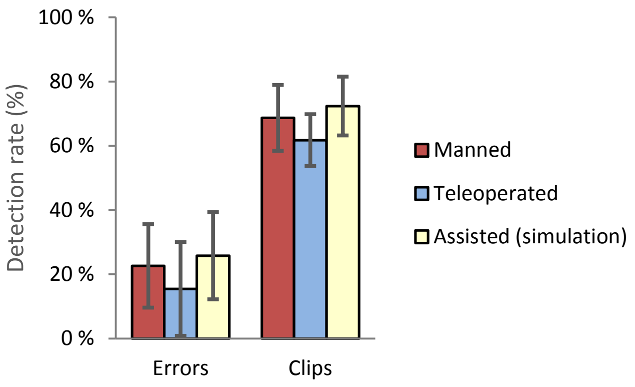


Figure 3.9: Average detection rates for manned (μ_M), teleoperated (μ_T) and assisted inspections (μ_A).

3.4.2.2 NASA-TLX

The average NASA-TLX results for the three different inspection methods are shown in figure 3.10. The results were analyzed with a one-way ANOVA with an alpha value of $.05$ and a Tukey HSD post test, which found the following:

- Significant difference for mental workload ($F(2, 183) = 6.0, p = 0.003$). The post hoc test found that teleoperated ($M = 40.2, SD = 20.3$) was significantly different from both manned ($M = 32.0, SD = 16.6$) and assisted ($M = 28.9, SD = 19$).
- Significant difference for physical workload ($F(2, 183) = 92.57, p < 0.001$). The post hoc test found that manned ($M = 45.2, SD = 20.4$) was significantly different from both teleoperated ($M = 13.8, SD = 13.4$) and assisted ($M = 10.3, SD = 12.2$).
- Significant difference for temporal workload ($F(2, 183) = 13.6, p < 0.001$). The post hoc test found that teleoperated ($M = 45.9, SD = 21.8$) was significantly different from both manned ($M = 33.3, SD = 17.8$) and assisted ($M = 25.9, SD = 24.6$).
- Significant difference for effort ($F(2, 183) = 10.43, p < 0.001$). The post hoc test found that assisted ($M = 34.7, SD = 18.8$) was significantly different from both manned ($M = 49.5, SD = 15.5$) and teleoperated ($M = 43.0, SD = 19.8$).
- No significant difference for performance ($F(2, 183) = 0.7, p = 0.5$).
- No significant difference for frustration ($F(2, 183) = 2.71, p = 0.07$).

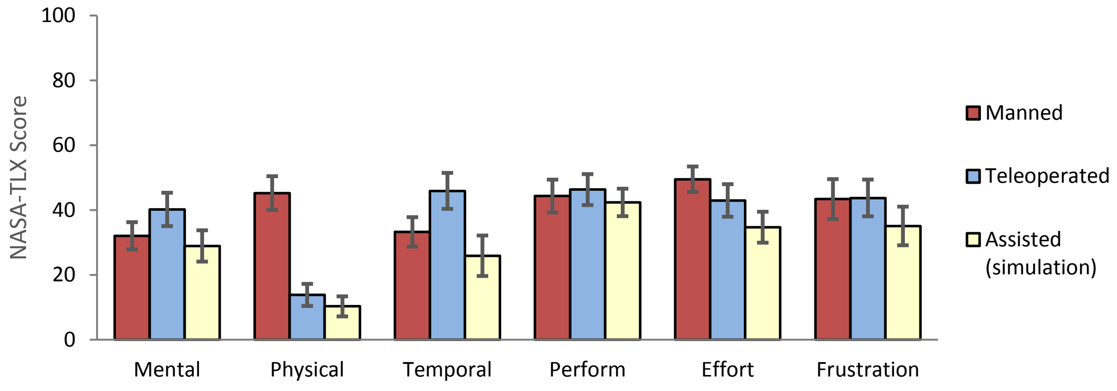


Figure 3.10: Average NASA-TLX results for manned, teleoperated and assisted inspections.

3.4.3 Second Experiment

The second experiment was similar to the first, with some improvements to the procedure and 31 participants. The presented results are based on 30 participants doing 4 inspections each, a total of 120 inspections. Four groups of targets, with 3 error markers and 4 paper clips each, were defined, and these were rotated between the inspections.

Two inspection methods were tested; assisted inspection implemented on the robot system and manned inspection. Each participant did two inspections of each type in a randomized order. The participants had 3 minutes for manned inspections and 5 minutes for the remote.

The results of one participant were not used because of severe technical problems with the robot during both of his remote inspections. In addition, one manned inspection was by accident performed without paper clips, so the paper clip measurement for this and the corresponding remote inspection have been omitted in the analysis.

3.4.3.1 Errors and Paper-Clips

The average detection rates for the inspections sorted for inspection methods are shown in figure 3.11(a). The detection rate of teleoperated remote inspection is 83% of the detection rate of manned inspection for error markers, and 85% for paper clips.

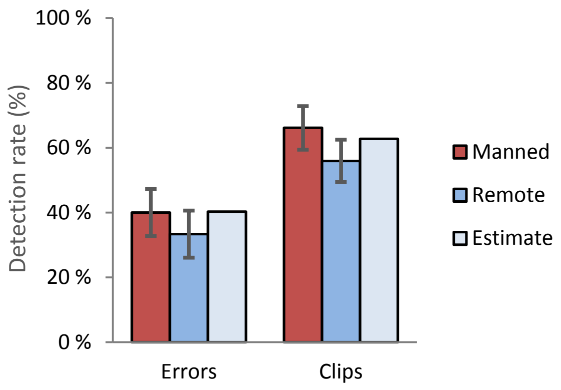
The average difference in detection rates ($\mu_M - \mu_R$) between two corresponding remote and manned inspections are shown in figure 3.11(b). The positive values means that the manned inspection performed better.

The results were analyzed with a within-subject design, one-tailed t-test with an alpha value of $.05$, which found the following:

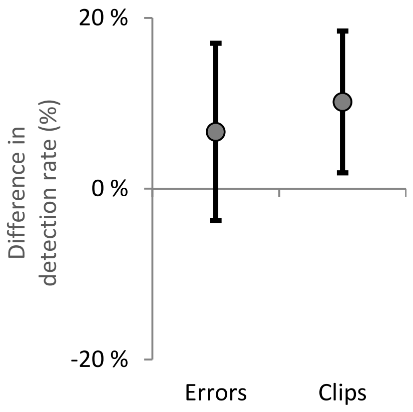
- Not significant for error markers
  $(M = 6.7\%, SD = 40\%), t(59) = 1.286, p = .102.$
- Significant for paper clips
  $(M = 10.3\%, SD = 32\%), t(58) = 2.451, p = .009.$

Many participants did not manage to visit all the observation points during their remote inspections, i.e. did not properly complete their inspection. Our estimated detection rate, shown as the third bar in figure 3.11(a), is based on the ratio of targets that was found in the part of the inspection the participants actually completed. Following the same estimate, the average time for completed remote inspections would be approximately 5 minutes and 30 seconds, almost twice that of manned inspections.

There were not found any significant correlations between the participants' experience with computer games or remotely controlled toys and the number of found targets.



(a) Average detection rates for manned (μ_M), remote inspections (μ_R) and an estimated result for remote inspection if given more time.



(b) Average difference in the detection rates with manned and remote inspections ($\mu_M - \mu_R$).

Figure 3.11: The detection rates grouped on inspection method.

3.4.3.2 NASA-TLX

The average NASA-TLX results for the two different inspection methods are shown in figure 3.12.

The results were analyzed with a within-subject design, one-tailed t-test with an alpha value of $.05$, which found the following:

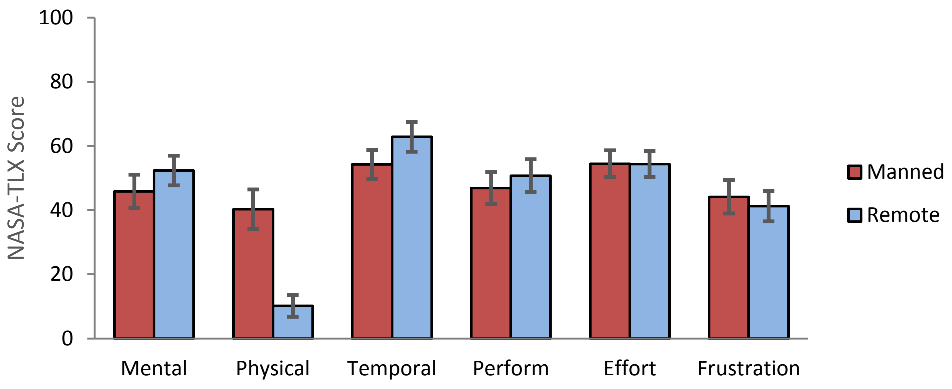


Figure 3.12: Average NASA-TLX results for manned and remote inspections.

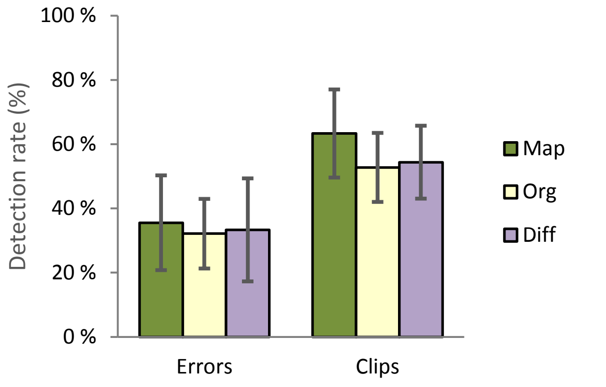
- Significant for mental workload $(M = -6, SD = 19), t(59) = -2.601, p = .012.$
- Significant for physical workload $(M = 30, SD = 25), t(59) = 9.177, p < .001.$

- Significant for temporal workload $(M = -9, SD = 19), t(59) = 3.467, p = .001$.
- Not significant for performance $(M = -4, SD = 21), t(59) = -1.405, p = .165$.
- Not significant for effort $(M = 0.1, SD = 17), t(59) = 0.039, p = .969$.
- Not significant for frustration $(M = 3, SD = 21), t(59) = 1.086, p = .282$.

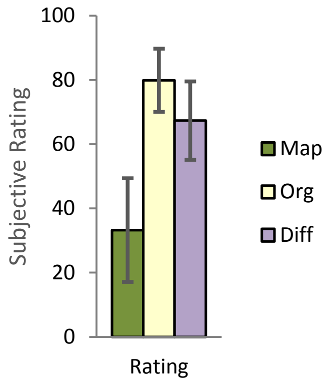
3.4.3.3 Tablet Controller View

The map and difference views were used 29% of the time each, while the original view was used 42% of the time. The results were grouped based on which tablet controller view that was perferred for each inspection. These results are showed in figures 3.13 and 3.14.

The results were analyzed with a one-way ANOVA with an alpha value of $.05$ and a Tukey HSD post test, which found the following:



(a) Average detection rates for map, orginal and difference views.



(b) Participants' subjective usefulness rating for map, orginal and difference views.

Figure 3.13: The detection rates grouped on the most used tablet view.

- Significant difference for rating $(F(2, 57) = 15.95, p < .001)$. The post hoc test found that map $(M = 33, SD = 29)$ was significantly different from both original $(M = 80, SD = 25)$ and difference $(M = 67, SD = 24)$.
- Significant difference for mental workload $(F(2, 57) = 3.80, p = .028)$. The post hoc test found that orginal $(M = 58, SD = 19)$ was significantly different from difference $(M = 44, SD = 11)$.
- No significant difference for error markers $(F(2, 57) = 0.07, p = 0.933)$.
- No significant difference for paper clips $(F(2, 57) = 0.8, p = 0.454)$.

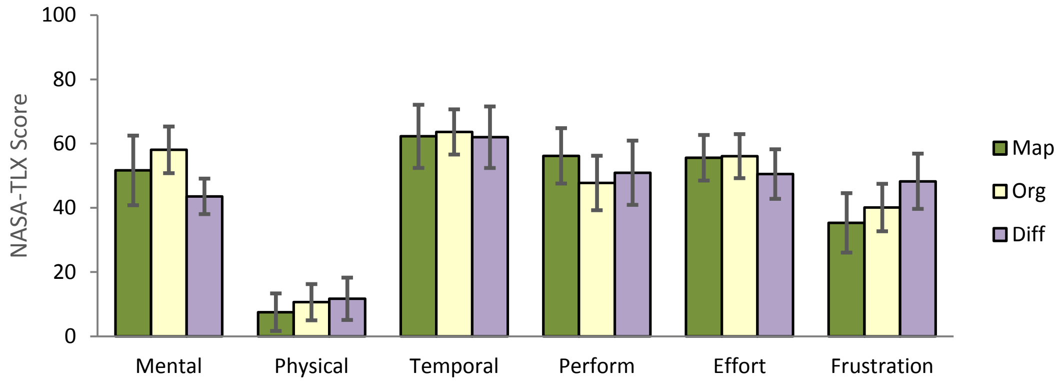


Figure 3.14: Average NASA-TLX results for map, orginal and difference views.

- No significant difference for physical workload ($F(2, 57) = 0.434, p = 0.65$).
- No significant difference for temporal workload ($F(2, 57) = 0.053, p = 0.949$).
- No significant difference for performance ($F(2, 57) = 0.89, p = 0.416$).
- No significant difference for effort ($F(2, 57) = 0.712, p = 0.495$).
- No significant difference for frustration ($F(2, 57) = 2.203, p = 0.12$).

3.4.3.4 Van der Laan

The results from the Van der Laan survey (Van Der Laan et al. 1997) given to the participants after they completed all four inspections are shown in figure 3.15.

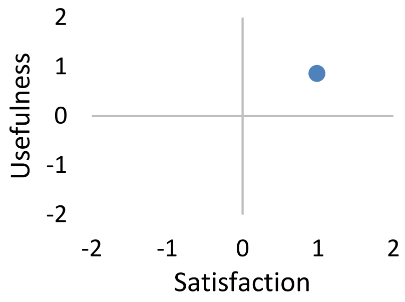


Figure 3.15: Van De Laan results for the remote inspection prototype.

3.5 Discussion

This is a discussion for the series of experiments. It is based on the discussions in articles C, E and F, but with a stronger emphasis on discussing the combined results.

3.5.1 Experience from the Pilot Experiments

The experience gained from performing these experiments was more important than the results. We found that it was important that the participants could test the remote inspection system before they started their inspection. When not given this opportunity, the participants spent a large portion of their time getting used to the system, which had a negative effect on the results. It is also realistic that the operators of such a system would have some training before using it.

In the first pilot experiment, it was observed that some participants gave up on finding more error markers and focused entirely on finding paper clips. In the second pilot experiment, when the paper clip task was not used, a similar observation was made, except that without the paper clip task the participants gave up the inspection entirely. The paper clip task functioned both as an additional measurement and as a motivation for continuing the inspection. Due to this, the paper clips were used for the rest of the experiments.

3.5.2 Comparison of the Usability of Remote and Manned Inspections

When considering the results from all experiments, remote inspections have a lower average detection rate than manned inspections. However, the difference is not large, and has been reduced as the remote inspection prototype has been improved. It is important to remember that the experiments were performed approximately 10 minutes after the participants used the robot for the first time. This is compared with doing examinations in person, which is a commonplace task for most people.

Remote inspection performed best compared to manned inspection in the second experiment, where the assumed best version of the prototype was used. It was also the experiment with the most precise results. The difference in detection rates were significant for the paper clips, but not for the error markers. This suggests that remote inspection is less effective, but only marginally so.

If remote inspections have a lower detection rate, it will result in a higher risk of turbine failures, with the cost and downtime associated with this. This problem can be reduced by performing remote inspections more often, as two remote inspections would combined have a higher detection rate than one manned inspection. This assumes that a remote inspection system is capable of detecting the same errors as manned inspections. This assumption should be examined in future research.

It is also possible that improving the remote inspection prototype would reduce or eliminate the difference between remote and manned inspections. The assisted inspection simulation in the first experiment performed as well as manned inspection, and it only used information gathered by the robot. The mental workload was also the same as for the manned inspections, while it was significantly higher for regular remote inspections. It did have the advantage of less delay and other problems with the prototype, but it means that there is a potential for remote inspection to be as good as or better than manned inspection given an improved implementation.

It was common that participants ran out of time before they had inspected the whole laboratory remotely. That the participants had a short time available for the remote inspections were also supported by the significantly higher temporal workload ratings for remote inspections and the large number of comments that mentioned this as a problem. In a real setting it is not very likely that an inspection would be aborted before completion due to time, which some of the participants in the experiment had to do. This means that some targets were never seen by some of the participants. Since the intention of the experiments were to examine whether remote inspections were able to detect targets as well as manned inspection, it will skew the results that some targets were never seen by some participants.

To get an indication of what the detection rate would be if the participants had enough time to complete their inspections, an estimate was calculated for the second experiment. It was based on the ratio of found targets within the part of the laboratory that each participant actually visited, and was almost identical to the results from the manned inspections. Although it is difficult to determine how reliable this estimate is, it does indicate that given more time to complete the inspections properly, the remote inspections could be as effective as the manned. Since it is acceptable that remote inspections takes longer time than manned, the estimated results are relevant.

Another problem in the experiments was that contrary to the plan of having targets that were possible to detect with both inspection methods, one error marker was in practice not possible to detect with the remote inspection and no participants did so in all of the experiments. This was not due to a limitation in the concept of remote inspection, but to the quality of the camera. In retrospect, this error marker should not have been used. It did not make a large difference in the first experiment, but in the second this particular target were found 7 times during manned inspections, more than half of the total difference between the two inspection methods.

3.5.3 Comparison of Teleoperated and Assisted Inspections

In the first experiments, it was clear that having direct control of the robot with a joystick control was not optimal. Too much time and attention were given to controlling the robot, which means that less remains for the inspection. The assisted inspection simulation tested in the first experiment demonstrated that without any communication delays or similar with the robot, the remote inspection could potentially be as good as or better than manned.

There were two main changes between the teleoperated inspections from the first experiment to the assisted inspection in the second. The control was easier, with a way-point type control, and the tablet was introduced as an input device and a secondary screen. This resulted in an improved effectiveness for remote inspection in the second experiment when compared to the respective manned inspections. The comments from the participants also support that the assisted inspection control method was an improvement. No participants were observed to have any problems with the controls.

However, the efficiency was not improved, and perhaps reduced, as it was still difficult for many participants to complete their inspections properly within the increased time available. The control of the robot was easier, but it still took some time to move the robot, which can explain some of the low efficiency. Two other likely reasons were identified.

- Assisted inspection encouraged the participants to be systematic and thorough. It was observed that some participants studied the image from each observation point closely and found all or almost all targets there before moving on. The problem was that they did not have enough time to be equally thorough on the whole laboratory. Even if this had a negative effect on the performance of assisted inspection in this experiment, we argue that it is positive that the system encourages this type of approach.
- Active use of the original and difference views on the tablet required time, especially the difference view as the time to stabilize the image and for running the algorithm was significant. These views were still considered useful by the participants. The use of these views did not change the detection rate, which suggests that any advantage provided by these were not larger than the disadvantage of having less time for the rest of the inspection.

3.5.4 Evaluation of the Experiments

There was a low precision in the results due to high variance. This is expected when doing experiments with people. The problem with low precision was especially pronounced for the error markers in the first experiment, as seen in the large 95% confidence intervals in figure 3.9. The main reasons for this were identified as:

- Few error markers divided between many inspections.
- The error markers being harder to find than intended.
- Limited number of participants.

The reason for having few error markers for each inspection was that it was considered realistic that such errors were uncommon. Unfortunately, the participants tended to get frustrated and gave up when not finding any or only a few error markers. For the second experiment, it was three error markers per inspection, compared to between zero and two in the first experiment. A more thorough description of the inspection task was also given to the participants. Due to this, there was a higher ratio of found error markers in the second experiment. Combined with more error markers in total, this means that each finding had a smaller effect on the results, and subsequently the variance in the results was lower.

The lower variance and increased number of participants resulted in lower 95% confidence intervals for the second experiment. An example is that in the first experiment, 15% of the error markers were found with remote inspection, and the confidence interval was 15%. For the second experiment, 33% of the error markers were found with a confidence interval of 7%. The confidence interval is less than one quarter of the measured value instead of the same size. This makes the results of the second experiment significantly more reliable.

How difficult it was to detect each of the targets varied. This was controlled for in the experiments by having each target being used the same number of times for the different tested inspection methods. In the second experiment, predetermined groups of targets, intended to be of similar combined difficulty, were always shown together. This resulted in a well-balanced experiment.

3.5.5 External Validity

Our test environment has not been compared with an actual wind turbine, thus the results can only be considered an indication of how a system for remote inspection would function in a real scenario. Testing in a real wind turbine would be expensive, time-consuming and there would be safety concerns to consider. A controlled experiment where errors are added and removed on demand would be impossible, or at least very impractical.

Although the error markers are not authentic errors, they are representative in the sense that they are difficult to find, unknown to the participants, but recognizable as errors when detected.

Since one of the participants in the second experiment had relevant experience, we asked him to comment on how realistic he found the error markers to be. According to him, some of the error markers were highly realistic, and was symptoms that he actively searched for due to his training. Some of the other error markers, which were frequently identified by other participants, he was unable to find, as he did not expect them. This is a strong indicator that these error markers were not realistic, but since other participants correctly identified them, they are still considered appropriate targets in these experiments with participants without inspection experience.

Chapter 4

Simulink Coder Generated Code as a Module within a Software Project

This chapter presents the following work based on articles A and D:

- 4.1 is a general description of how Simulink Coder can be used for embedded control systems, and how to modify it for a specific use.
- 4.2 presents the *Software Module Real-time Target* (SMRT) solution, which is a different approach to using the code generated from Simulink. This solution was developed as part of the prototype described in chapter 3, but has evolved into a separate project.
- 4.3 discusses SMRT and its use.

All code associated with SMRT, instructions for using it and description of how to port it for other platforms are found on the SMRT web site [1], under the LGPL v3 license.

4.1 Simulink Coder for Embedded Linux

4.1.1 Simulink Coder

Simulink is a part of MathWorks Matlab that can be used for graphical modelling and simulations of dynamic systems. A Simulink model consists of blocks with different functionality that can be connected together to make complex systems.

A Simulink simulation will calculate the state of each Simulink block at specified points in time during the simulation. The intervals between these points in time are called *time steps*. How the states of the blocks change between the time steps depend on their properties and how they are interconnected. The simulation calculates how the states of the blocks in the model change over time.

[1]http://www.itk.ntnu.no/smrt/index.php

In control theory, it is common to model a system in Simulink and then design a controller for it. An example of such a system would be the yaw drive of an offshore wind turbine, and the controller is intended to keep the turbine pointing towards the wind. Simulations can then be used to find how well the controller performs under different conditions. A large number of controller parameters can be tested in a shorter time and a lower cost that if it was done on the actual system. There will also be no damage to the system if the controller misbehaves.

Simulink Coder (MathWorks 2012), or Real-Time Workshop as it was named when paper A was written, is a toolbox for Simulink. It generates code that when executed will do the same calculations as a Simulink simulation of the same model. As the generated code is an accurate representation of a Simulink model, it can be used to implement a control system that is identical to a controller modelled in Simulink. After testing with simulations, it will be necessary to test against the actual system, and Simulink Coder offers a method for continue using Simulink for this. If we assume that the code generation process is perfect, it will generate code that is a correct implementation of the model. The alternative to this would be to write code that implements the Simulink model manually and updating it every time the model changes. This is not cost effective, and the result will be unreliable because every change in the model could possibly introduce coding errors.

4.1.2 Host and Target

An embedded system is often intended for a specific task and can lack equipment like screens, keyboard etc. that a developer normally would use. Therefore, it is common to use a normal laptop or desktop computer for developing and compiling the programs that will run on the embedded system. This computer is referred to as the *host computer*, while the embedded system is referred to as the *target computer*.

It is easy to get Simulink Coder to generate code from a Simulink model. However, depending on the requirements of the system, it can be difficult to have it build a program that runs on the target computer. It will often be necessary to have specific code for the target computer, use cross-compilers, and write code manually for drivers, hardware interfaces and complex functions (Skavhaug et al. 2002). This requires both experience with the embedded platform that is used and knowledge of how code is generated by Simulink Coder. However, when set up correctly, it is easy and efficient for users without this experience to do changes in the Simulink model, generate code and test it.

4.1.3 System Target

A *system target* is a set of files that define how Simulink Coder should generate and build code. The default target is called *Generic Real-time Target* (GRT) and is included in the basic toolbox. This generates platform independent C-code that implements the

Simulink model, and builds it for the host computer. Another alternative is the *Embedded Real-time Target* (ERT), which provides the developer with a more detailed control of the code generation. Unfortunately this require an additional and expensive toolbox, thus we avoided it during our work, and used GRT instead. There are also system targets that are platform-specific, meaning that they are intended to be an easy to use solution for a specific embedded platform.

Custom system targets can be created by copying the files of an existing system target. It can then be modified to generate and build code for the target computer with the necessary requirements. Three files define GRT:

System target file is a TLC-file that defines how the code is generated and which options that are available. In most cases, it will not be necessary to change this.

The makefile template is used by Simulink Coder to generate a Makefile, which in turn determines how the code is built. For GRT it comes in different variants depending on the build environment. Changes in how the code should be built can be specified here.

The main-file is a C code file with a main function, the first function executed by a C-program. The main function in this file will manage the initialization and execution of the model using the generated code. Changes to how the generated code is initialized and executed can be specified here.

A custom system target, based on the generic real-time target, can be designed to produce a program that implements a control system on a target computer. Two changes are needed to accomplish this, and possibly more depending on the requirements. Firstly, the makefile template must be changed so the program will be built correctly for the embedded platform. This will often consist of specifying that a cross-compiler should be used, and possibly other build instructions.

Secondly, the default GRT main file does not execute the code in real-time, which means that when the state of all blocks at one time step is calculated, the system immediately starts calculating the next time step. This is a suitable behavior for simulations, as you would like to do the simulation as quickly as possible, but when interacting with a system in the real world, the control system must operate in real-time. Code that control when each time step is executed can be added to the main file.

4.1.4 S-functions

A custom system target can be used to specify how code is generated and built, but these are target specific settings and do not modify the behavior of the Simulink model itself. If it is necessary to define your own functionality that is not available in the basic Simulink blocks, then *S-functions* should be used. These are custom Simulink blocks that can be divided into three categories:

Non-inlined S-functions are defined by a C-file following a certain template that defines how the block behaves in a simulation. Code generated for the block will behave as it does in simulations. The C-file is compiled within Matlab on the host computer, which means that it is not possible to use target specific code with non-inline S-functions. Code generated by non-inline S-functions will have reduced performance compared to code generated by inline S-functions.

Fully inlined S-functions are defined by a C-file and a TLC-file. The C-file is the same as for non-inlined S-functions, while the TLC-file defines how code is generated for the block. This gives the developer full control of how code is generated, and target specific code can be used. The block's behavior in simulations and in the generated code can be different, since they are defined in two different files. The downside is that the TLC language is Simulink specific, and can be difficult to work with for developers without prior experience.

Wrapper S-functions are a variant of fully inlined S-functions, with a third file containing code that is shared between the two others. This makes it possible to have the same functionality in simulations and when executing the generated code, without the reduced performance of non-inline S-functions. Only some TLC code is needed to make a wrapper S-function.

For embedded control systems, there are two main uses for custom code through S-functions. Firstly, there could be some specific logic or mathematical function that is easier to define with code than with Simulink blocks. This can often be non-inlined S-functions, or possibly wrapper S-functions, as the block should do the same in simulations and in the control program.

The second use is to access a resource the target system has, e.g. I/O signals. The host will normally not have the same resource, thus a fully inlined S-function should be used. The TLC-file should specify how to generate code that uses this resource. It is often desirable to define an alternative behavior for when the block is used in simulations.

4.2 Software Module Real-Time Target

4.2.1 Xenomai Linux on Beaglebone

The Beaglebone[2] is a credit-card sized ARM development board (figure 4.1). It was used as the control system for the robot prototype that was described in 3.3, and for the development and evaluation of *Software Module Real-time Target* (SMRT). Its small size and affordable cost makes it a suitable embedded device for many control system applications where Simulink and SMRT can be used. Compared to similar devices, the Beaglebone has the advantage of a large number of I/O pins, including I2C, UART, GPIO, PWM and ADC.

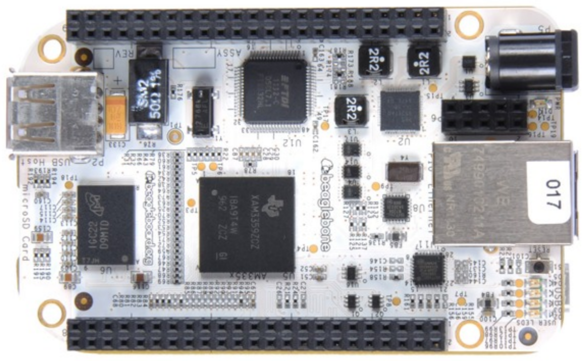


Figure 4.1: The Beaglebone development board.

The Ubuntu distribution was used for the same reasons as given in 3.3.2.1. The hardware floating-point arithmetic is considered especially important when running Simulink Coder generated code, as it relies heavy on floating-point variables.

SMRT and the Beaglebone are suitable for implementing control systems that interacts with the outside world. Such systems will often have real-time constraints, meaning that in addition to producing a correct result, the result also has to be produced at the correct time. A hard real-time constraint means that if a result is produced too late the system will fail its purpose, possibly with dire consequences.

Ordinary Linux is not considered a real-time system, because it is not predictable enough to guarantee that real-time constraints will be met. This is because the Linux kernel is not fully pre-emptive. If it is currently executing critical code in the kernel, other tasks will not be allowed to run until this has finished. It might be a rare event

[2]More information about Beaglebone at: http://www.beagleboard.org/bone

that another task is significantly delayed, but when operating on a millisecond scale, a one in a million event is likely to happen once every 17th minute.

There are several additions to Linux that allow tasks to run with higher priority than the Linux kernel itself, with pre-emptive scheduling, i.e. supporting hard real-time. Xenomai (Gerum 2004) was chosen because it is a well-maintained, documented and active project with an available port for the Beaglebone. Other well-known alternatives are RTAI, RT-Linux (Barabanov et al. 1996) and Preempt-RT. Barbalace et al. (2007) compared the real-time performance of Xenomai with regular Linux, RTAI and VxWorks, where Xenomai was shown to be a valid alternative to both VxWorks and RTAI.

Xenomai uses a co-kernel approach, which means that a small real-time kernel runs besides the Linux kernel on the same CPU. This kernel does the scheduling for all the real-time tasks, and when idle, it allows the Linux kernel to run. All interrupts are also processed by the real-time kernel before they are sent to the Linux kernel. We followed the instructions from a web page [3] to port Xenomai for the Beaglebone. The port consists of combining the Linux kernel patches necessary for both the Beaglebone and for Xenomai.

4.2.2 SMRT

Some of the challenges with setting up Simulink Coder for an embedded system are discussed in section 4.1. Several methods for automating the code generation and build process for different hardware and software platforms have been presented (Gong 2000; Quaranta et al. 2001; Teng 2000), including article A.

Many of the alternatives to SMRT, both provided by MathWorks as part of Simulink Coder or other toolboxes and third parties, are solutions that create a ready to use program for a specific embedded platform. This will be a suitable solution for many projects, and are typically used by control engineers that want to implement their algorithms on hardware with minimal need for coding. The disadvantage is that it requires knowledge of how Simulink Coder generates code and the TLC language to do customizations or add your own code to the project.

SMRT is not intended to be an alternative to this, or to be a method for setting up a control system with minimal effort. It is intended for developers that are developing an embedded system which consist partly of code generated from Simulink Coder, thus flexibility and easy integration with other code and development tools are important. A different approach is used, where the main file, which compiles into a standalone program, is replaced with `smrt.c`. This files does not have a main function, instead it has functions for initializing and executing threads that run the generated code. SMRT can be built as a shared library, and its functions can be used by other software projects where the functionality of the Simulink model can be used. Figure 4.2 compares the

[3]http://yapatel.org/wiki

relationship between the generated and custom code in SMRT compared to a more typical solution.

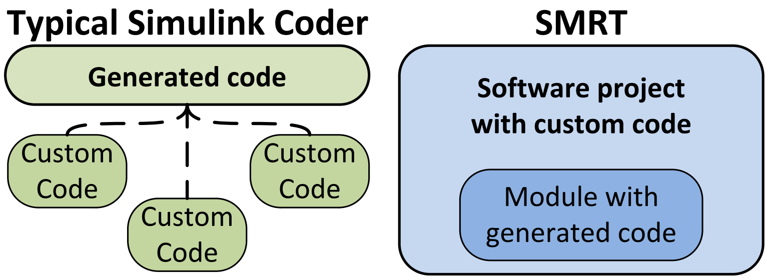


Figure 4.2: The relationship between generated and custom code in SMRT compared to a typical use of GRT

The intended users of SMRT are embedded developers that are not expected to be experts in how Simulink Coder works. Because of this, SMRT has been designed to require no specific knowledge about Simulink Coder, its code generation process, nor the TLC programming language. SMRT has been tested in Eclipse, but other development environments should be possible to use too. If Eclipse is used, all configurations of SMRT can be found within the project, where an experienced Eclipse user would expect to find them, instead of fragmented in several Matlab and Simulink specific locations as seen in table 4.1. SMRT has also been configured to use external mode in Simulink, which means that the value of signals on the running embedded system can be viewed in near real-time, within Simulink. Some block parameters can also be adjusted without restarting the program.

4.2.3 SMRT Multitasking

A Simulink model using a fixed step solver, which is necessary for Simulink Coder, have a model sample time that is the interval between the time steps of a simulation. This value can be set manually or chosen by Simulink. When running a simulation, all blocks will be executed at each of these time steps, i.e. *single-tasking*. In SMRT, single-tasking is implemented by starting one periodic Xenomai real-time task that executes the whole model.

Some parts of a control system might need to respond fast, and need a short sample time, while other parts can be updated less often. If this would be implemented with single-tasking, all blocks in the system would have to operate at the same sample time as the part that required the fastest response, thus it would require more CPU cycles than necessary. To avoid this, it is possible to specify the sample time of each individual block in a Simulink model, i.e. *multitasking*. In SMRT, multitasking is implemented by starting one periodic Xenomai real-time task for each of the sample times in the model.

Configuration	**GRT**	**SMRT**
Options for Makefile	Simulink config	NA
Build instructions	Makefile template	Eclipse project
Build options for external code	Makefile template	NA
Cross-compilation	Makefile template	Eclipse project
Compiler and linker flags	Makefile template	Eclipse project
Periodic execution	`main()` file	SMRT Library
Implementation of multitasking	`main()` file	SMRT Library
Calls to custom code	`main()` file and S-functions	Eclipse project
I/O and hardware interfacing	S-functions and external code	Eclipse project
Aperiodic events	External code	Eclipse project

Table 4.1: Where configurations are stored in GRT and SMRT using Eclipse.

Each task is given a priority depending on their sample time, the lower the sample time, the higher the priority. This follows the *rate monotonic scheduling* (RMS) principle (Burns et al. 2001; Liu et al. 1973). As the number of tasks and their sample times are known when building the system, it is appropriate to use RMS. As the SMRT tasks are likely to be part of a larger program with its own tasks, these must be considered if utilization tests or response time analysis are used to determine the schedulability of the whole system.

When using multitasking, Simulink prevents direct connections between blocks of different sample times. This is typically resolved by using *Rate transition blocks*. As a part of our SMRT implementation for Xenomai, we have developed an S-function, a variation of a rate transition block that uses a Xenomai mutex for protecting a variable that is shared between the tasks. Since Xenomai mutexes implements priority inheritance (Sha et al. 1990), we avoid problems related to unbounded priority inversion, and a high priority task is guaranteed access to the variable as fast as possible. The S-function is implemented in TLC, but knowledge of TLC is not necessary for using it.

Task	Period	Execution time	Priority
A	$40ms$	$10ms$	94
B	$50ms$	$12ms$	93
C	$70ms$	$20ms$	92

Table 4.2: Execution time and periods of tasks used in scheduling test.

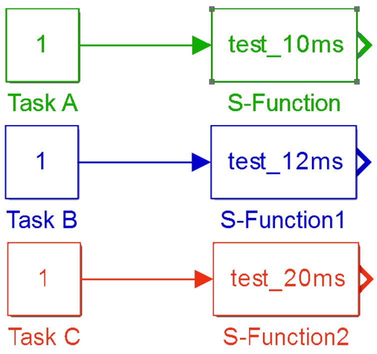


Figure 4.3: Simulink model used in the test (colors indicates period of tasks).

4.2.4 Task Scheduling Test

The SMRT library has been implemented to execute its multiple tasks according to the rate monotonic principle. We wanted to test that it behaved as expected, and compare it with other uses of generated code. Three S-functions were created that performed busy-wait loops measured to take $10ms$, $12ms$ and $20ms$ of execution time.

In a small test, each of these S-functions was executed periodically, as defined in table 4.2. Their combined utilization is $77.6\% \leq m(2^{\frac{1}{m}} - 1)$ for $m = 3$ (Liu et al. 1973), meaning that an ideal rate monotonic scheduling would guarantee no deadline misses.

The Simulink model was run with three different configurations. The first used GRT modified to run each Simulink task as a Xenomai real-time thread. The second was SMRT using Xenomai threads, while the last was SMRT using normal Linux threads. The response time was defined as the time from when a period was supposed to start, until its execution was completed. This was measured over a period of three hours, and the results are shown in table 4.3.

Using response time analysis (Burns et al. 2001) on the set of tasks, we find that the theoretical worst-case response time is $10ms$ for task A, $22ms$ for task B and $64ms$ for task C. The worst-case results when using Xenomai (both GRT and SMRT) were, in

	Results	GRT + Xenomai	SMRT + Xenomai	SMRT + Linux
	Average response time	$10.3ms$	$10.0ms$	$10.3ms$
A	Max response time	$10.4ms$	$10.1ms$	$198.5ms$
	Ratio of deadline missed	0.000%	0.000%	0.003%
	Average response time	$17.7ms$	$17.0ms$	$17.2ms$
B	Max response time	$22.9ms$	$22.2ms$	$230.9ms$
	Ratio of deadline missed	0.000%	0.000%	0.004%
	Average response time	$44.7ms$	$41.7ms$	$41.0ms$
C	Max response time	$66.5ms$	$64.0ms$	$334.6ms$
	Ratio of deadline missed	0.000%	0.000%	0.007%

Table 4.3: Results from simulations running for 3 hours.

this test, indistinguishable from these theoretical values. This is as expected and demonstrates the hard real-time capabilities of Xenomai. It also suggests that SMRT correctly follows the rate monotonic principle. Whether SMRT or GRT was used had no observable effect on the scheduling, nor was it expected.

SMRT and Xenomai had predictable worst-case performance in this test, but this was because the used blocks have specifically been designed to have predictable and known execution times. This does not imply that other Simulink models have predictable execution times, even when running in Xenomai. This is because the temporal performance of the different Simulink blocks varies. If real-time performance is important for Simulink generated code, then the blocks in the model should be chosen carefully. Information about how different blocks are expected to behave is given at page 1-93 in the Simulink Coder user guide (MathWorks 2012).

As expected, using normal Linux threads resulted in significantly higher worst-case execution times. There were also multiple deadline misses during the 3-hour test. This shows that when SMRT is used for applications that have hard real-time requirements, normal Linux is not suitable.

4.2.5 Input and Output S-functions

S-functions are used to include custom code into a Simulink model, but can be difficult to use as they are coded in the Simulink specific TLC language. It is intended that users of SMRT should not need to use TLC. To allow for custom code without using TLC, two S-functions were developed that run code specified in C-functions defined in a normal code file.

The input S-function is a Simulink block with one output signal. For each time step, this block calls the C-function with the name that is specified in the block parameter. This function will return a value, which will be set as the output signal of the block. What this function does and how the return value is calculated is defined in the C-file outside of Simulink, specified by the developer. If the developer fails to specify a C-function with the correct name and type in the project, there will be an error when linking with the SMRT library. Multiple such blocks can be used in the same model, but each should specify a different function name. This S-function allows custom functionality of a Simulink block to be defined together with the rest of the code project, outside of the Simulink context, and without any TLC code. This makes it easy for programmers with experience from embedded systems and the C-language to create custom Simulink blocks.

A similar output S-function is also available, which is a block with one input signal. Each time step, the function specified in the block parameter is executed, with the input signal as one of the parameters. This has the same benefits as the input S-function.

Figure 4.4 shows how the generated code is a part of the larger written code, and how the input and output S-functions can act as the interface between the two parts. The figure also shows that some parts of the whole application can be implemented without using the Simulink generated code. This demonstrate that with SMRT, the developers can chose freely which parts of the system that it is suitable to implement in Simulink, and which are more practical to implement manually.

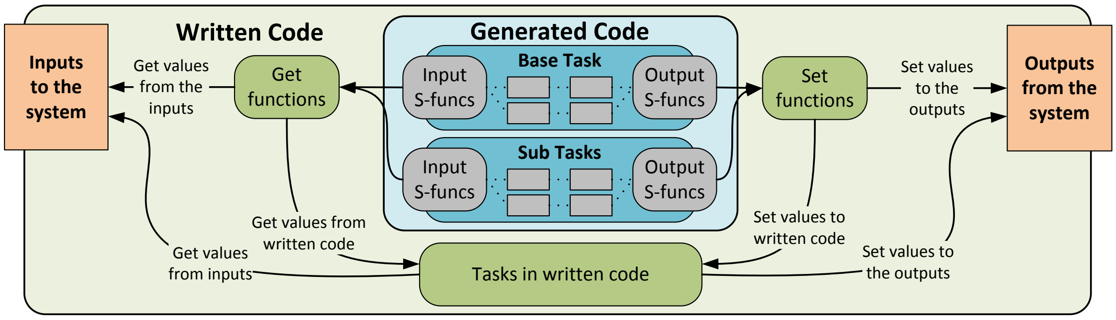


Figure 4.4: Graphical representation of communication between generated and custom code in SMRT

A typical use of these input and output S-functions are for communicating with the I/O of the embedded system. Set and get functions for the different I/O pins of the system can be created, and the S-functions can specify these in the parameters. The S-function will then either read or write to the I/O at every time step.

Although the S-functions are developed with TLC, it is not needed for using them. They are also platform independent, as their generated code only consists of a function call. The current versions only have one input or output, and do not do anything in a

Simulink simulation, but these are possible improvements. A third variation, with both input and output signal could be beneficial in some situations, and should be developed.

4.2.6 Xenomai Real-Time Drivers for Beaglebone GPIO and PWM

Xenomai tasks can run at a higher priority than the Linux kernel. However, if the Xenomai task uses a Linux system call, this will run in the context of the Linux kernel, and thus lose its real-time capability. This means that a Xenomai task that access the I/O of the Beaglebone using Linux drivers will lose its real-time priority, and introduce unpredictable delays. To circumvent this, I/O drivers can be developed using the Xenomai real-time driver model (RTDM).

RTDM drivers are similar to normal Linux drivers, but use RTDM functions instead of the Linux equivalents. When built they are loaded into the Linux kernel just as another kernel module. Real-time task can then interact with them. Since SMRT with Xenomai is intended to have hard real-time performance, we have developed RTDM drivers for the GPIO and PWM I/O pins for the Beaglebone, which is available at the SMRT project web site.

4.2.7 Porting SMRT to other platforms

The input and output S-functions are platform independent, and can be used unchanged for SMRT regardless of the target platform. With some modifications, it should be possible to use these S-functions for other Simulink Coder projects as well.

The SMRT implementation is not specific to the Beaglebone hardware, and it should be possible to run SMRT unchanged for Xenomai on other hardware.

As the current implementation of SMRT uses several Xenomai specific functions, it would require some work to port SMRT to other operating systems. However, all these functions have POSIX equivalents, thus it should be possible. If ported to a non real-time system, increased response times and missed deadlines as shown with Linux in 4.2.4 should be expected.

4.3 Discussion

The Software Module Real-time Target use the code generated from Simulink Coder as a software module available for other parts of the project to use. With this method, both the configuration and custom code can be specified with tools an embedded developer is comfortable with, e.g. C-code, the Eclipse IDE and Makefiles. S-functions have been developed to simplify the interface between generated code and the rest of the program, thus there is no need to develop S-functions for this or learn the TLC-language.

SMRT for Xenomai Linux has been tested on a Beaglebone development board. A test of the scheduling showed that it was seemingly identical with the theoretical rate monotonic scheduling and as good as other uses of Simulink generated code. There was not found any undesirable behavior for hard real-time. The SMRT code and instruction for using it is available online under the LGPL v3 license.

SMRT was used for the development of the control system for the robot prototype described in 3.3. It made it possible add code generated from Simulink into a code project, and allowed simple interaction between the generated and written code. Simulink external mode was useful for debugging and testing the control system, as it allowed the robot to run the program, while values were shown in near real-time in Simulink.

Chapter 5

Concluding Remarks

5.1 Conclusions

5.1.1 Remote Inspection of Offshore Wind Turbines

This thesis has presented the concept of remote, robotic inspections of offshore wind turbines as a cost- and time-effective alternative to the manned inspections performed today. The potential economic benefit to remote inspection was demonstrated with a life-cycle cost simulation tool called NOWIcob. The results suggested improved availability and a reduction in the cost of energy when remote inspections were used.

An assumption for the NOWIcob simulation was that remote inspections are as effective as manned, meaning that the same number of problems is identified with both alternatives. A series of usability tests, two smaller pilot experiments and two larger experiments with 21 and 31 student volunteers have been performed to evaluate this assumption. A remote inspection prototype was developed and used for inspections in a laboratory and compared to manned inspections.

A comprehensive review of usability tests of mobile telerobots found in the literature has been performed as part of the preparation of our own experiments. It both describes the best practice for performing such experiments, and has an overview of their results. No other experiments with a direct comparison of participants performing a task with a robot and in person were found in the review. This could be because telerobots often are considered for tasks that are impossible or too dangerous for humans to do. For remote inspection, this is not the case, and it was necessary to get a direct comparison between the two methods.

The last experiment was the largest and had the most reliable results. Based on these we find that remote inspections with the current prototype and test procedure were able to find approximately 84% of the errors that manned inspections found. This means that it cannot be considered equally effective, but it is also a promising result. The difference between remote and manned inspections was smaller in this last experiment than in the earlier ones. This is likely to be due to improvements of the prototype. Further improvements are expected to reduce the difference even more.

It was identified several strong indications that most or all of the difference between the inspection methods was because many participants did not have enough time available to properly complete their remote inspections. Some of the reasons for this are due to the participants having to wait for the robot system, and some could be because the assisted inspection method encouraged systematic and thorough inspections. Given more time, remote inspections are likely to be as effective as manned. Since remote inspection does not require time consuming transportation, it can be beneficial even if the inspections take longer to perform. The system should still be developed to be faster and more efficient to use.

Our conclusion is that although an effectiveness equal to that of manned inspections has not been demonstrated, it performed well enough to justify continued research on the topic. Further development should focus on reducing the time the operator has to wait for the robot, and on including more and better sensors.

5.1.2 Simulink Coder Generated Code as a Module within a Software Project

Software Module Real-time Target (SMRT) was developed to simplify the development of the telerobot prototype. It has become a project on its own, which can be useful for other projects. It is intended to make it easy to use code generated by Simulink Coder as a part of larger project, in an environment the developer prefers. Most other solutions make a standalone program from the generated code, within the context of Matlab and Simulink.

SMRT is intended for embedded developers that are not experts in Simulink Coder. This means that it is not necessary to know how Simulink Coder generates code or how to use the Simulink specific TLC language. If another method is customized for the specific requirements of a project, this knowledge will most likely be necessary.

5.2 Future Work

There are no plans for continuing the series of experiments using the current laboratory and prototype. As the results from the laboratory is promising, it is more important to focus the research on evaluation in a real environment. An improved prototype capable of detecting the same errors, as it is possible to find during manned inspections should be developed.

The project will be continued within the EU project *Leanwind*, with the installation of a full-featured prototype in an offshore wind turbine. The lessons learned during the experiments will be used for the development of this system, to ensure its usability.

One of the advantages of evaluating remote inspection in a laboratory is, as argued in this thesis, that a large number of participants can do inspections in a controlled environment within a short time frame. To find out how to evaluate a prototype in a offshore wind turbine in operation will be a challenging task.

There is currently no known need for doing updates to the SMRT concept for Xenomai on Beaglebone. However, it is possible to continue the development of SMRT to work on other software and hardware platforms.

One of the problems of using Simulink Coder is the high cost of the software. Because of this, it could be an idea to explore the use of SMRT with open source alternatives to Simulink.

Chapter 6

Original Publications

This chapter contains five published peer-reviewed conference papers, as well as two submitted journal paper manuscript.

Paper A Adaption of MathWorks Real-Time Workshop for an Unsupported Embedded Platform

Published in the *Conference Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications* (SEAA 2010) at Lille, France, September 2010.

Adaption of MathWorks Real-Time Workshop for an Unsupported Embedded Platform

Øyvind Netland
Norwegian University of
Science and Technology
Department of Engineering Cybernetics
Email: oyvind.netland@itk.ntnu.no

Amund Skavhaug
Norwegian University of
Science and Technology
Department of Engineering Cybernetics
Email: amund.skavhaug@itk.ntnu.no

Abstract

This paper describes how to configure MathWorks Real-Time Workshop to automatically build a control system application for an unsupported embedded computer platform. The application can be generated with a single command, allowing for quick iterations of testing and debugging. The I/O interface and control algorithm of the application is described in a Simulink model. In this paper an AVR32 embedded computer running AVR32-Linux was used, but it should also be relevant for adapting Real-Time Workshop for other embedded platforms.

1 Introduction

Embedded computers are often used to control a physical system. This can be everyday objects, from the microwave in your kitchen to the brakes of your car, or an industrial system like a part of a factory or a ship. Control systems becomes increasingly more complex, which leads to increased chance of failure, that can result in serious economical consequences or personal injury.

A control system is often modelled with a tool like Simulink, a toolbox for the Matlab program from MathWorks. In Simulink, a dynamical model can be developed and simulated, until it is ready to test on the actual physical system.

Translating a Simulink model into computer code, and updating it every time the model changes is a difficult and time consuming task, that is prone to errors. It is not very cost effective, and the result will be unreliable.

The Simulink add-on Real-Time Workshop (RTW), is a tool for rapid software development. It can rapidly generate code that performs the same calculation as a Simulink simulation, free from human coding errors. Failures of control system are often more serious than for other computer systems, since losing control over a physical system can create dangerous situations. It can have large economical consequences and threaten the reputation of responsible parties.

Using RTW for rapid software development of control systems, is a well known and used concept, for both development and for teaching purposes. [11] discusses how RTW and similar tools have been used as student exercises at the department for engineering cybernetics at NTNU since the mid 1990s. [6] shows a successful usage of RTW for teaching digital signal processing.

A RTW system target defines how code is generated. Some, like the xPC-target, generates code for a specific platform and purpose. These can generate code that with no modifications can be used for real-time control systems [9, 10]. Others, like the generic real-time target (GRT), generates code that can be used on different platforms and purposes, but might require some modifications.

This paper describes how a custom RTW system target, based on GRT was created. It generates real-time control systems for AVR32-Linux described in the following parts.

- Background: Present background information.
- Preliminary tests: Tests of the selected platform.
- AVR32 Real-Time Target: RTW target for creating real-time control systems for AVR32.
- S-functions for I/O: Custom Simulink blocks.
- Demonstration Experiment: Testing of the complete systems capabilities.

The contributions from this work is a RTW system target for AVR32-Linux and a description of how to make a similar solution for other unsupported platforms.

2 Background

2.1 Real-Time Control system

A real-time system [3] is a computer system with real-time constraints, which means that a calculation must be both correct and completed at the correct time. Failing to do so will result in failure or reduction of the performance of the system.

Most control systems have a periodical execution. Each period read measurements, executes a control algorithm, and sets actuator commands. A system like this have a real-time constraint, since it must perform these operations before the end of the period in order to work.

2.2 Simulink and Real-Time Workshop

Simulink is a Matlab toolbox, and makes a numerical simulation of a dynamical system, with a graphical block diagramming tool. A simulation of a model consists of a numerical calculation of the dynamical system for every time step. The time step can be fixed or variable, but most RTW system targets like GRT only supports fixed, so only fixed will be used here.

Real-Time Workshop is a Simulink add-on for generating C code based on a Simulink model. The RTW generated code will perform the exact same numerical calculation as the Simulink program would do. If the time steps in the RTW generated code executes periodically, with a period equal to the fixed time step, then the resulting program will be an implementation of the model in real-time.

2.3 RTW System Target

When using RTW, the first option the user is presented with is which RTW system target to use. It defines how the code should be generated and compiled. The RTW User's Guide [13] explains how it generates code, and the different targets that are available. The three RTW systems that will be best for an unsupported embedded system is listed below. All of these must be modified to execute the time steps in real-time.

Generic real-time target (GRT) generates code that can run on many different platforms with little or no modification.

GRT malloc is the same as GRT, but all variables used during execution are dynamically allocated when needed instead of being declared in advance.

Embedded real-time target (ERT) generates code especially for embedded platforms. The code is more efficient, and more suited for production than GRT. It also have options for more detailed control of code generation. RTW Embedded Coder [12] is required, which makes it less available.

Other targets are also included in RTW and Embedded Coder, but these are specialised for a specific platform or usage.

A RTW system targets is defined by a few files, and changing these will change the way code will be generated. To make a new RTW system target, the files from an existing target can be copied and modified.

2.4 Simulink S-function

S-functions are Simulink blocks written in a computer language, and are used when none of the blocks in the Simulink block library have the required functionality. There are three different types of S-functions:

A noninlined S-function only have a S-function code file, which will be used both for Simulink simulations and for RTW code generation. This is the simplest S-function since it only requires one file, but the code generated by RTW will be ineffective.

A fully inlined S-function uses a S-function code file during Simulink simulations, and a target block file for RTW code generation, which gives optimal performance for both situations. The functionality of the S-function in Simulink can differ from the functionality of the RTW generated code, since it is defined by two different files.

A wrapper S-function have a third file which defines the functionality of the S-function. Both the S-function code file and the target block file will call functions from this third file. This allows the functionality of both Simulink simulations and RTW generated code to be defined by the same file, without any loss of performance.

2.5 AVR32

AVR32 [2] is a processor architecture that was developed by Atmel Norway, who are the producer of the 8-bit AVR micro controller. The first CPU of this architecture was the AT32AP7000 [1], that was released in 2006. AVR32 is a 32-bit processor, for the embedded marked. It is designed to have high code density and low power usage.

AVR32-Linux is a port of the Linux kernel for the AVR32 architecture. Together with some basic tools (GNU toolchain, Busybox and uClibc), it is a complete lightweight Linux distribution with development tools. Further information about the AVR32 Linux project can be found at *avr32linux.org*. The kernel version used for this work was 2.6.20.

The STK1000 development board was used, with the STK1002 daughter-board. The CPU model was the AT32AP7000. Other CPUs with different properties have been released since then, and more are probably following. None of these are discussed here.

3 Preliminary tests

3.1 Test of RTW generated code

After selecting a hardware and software platform, it is important to test and verify its ability to perform its intended task. Since the platform must be able to run RTW generated code, it is a natural choice to determine this first. Prerequisites for this test is a working compiler and other necessary tools for building applications for the embedded computer. The GRT target should be tested first, since it is generic and widely available. If it is not possible to run GRT code on the selected platform, the ERT target can be considered if available.

The GRT target was used for testing RTW code for AVR32-Linux. Since it is running Linux, the UNIX makefile was used for compilation. It was modified to use the AVR32-Linux compiler, by setting the following variable `CC=avr32-linux-gcc`.

Other platforms must set this variable to its own compiler, which can give compiler errors specific to the platform. How these are solved, if it is even possible is not discussed here, since it will require knowledge of the target architecture and operating system. For AVR32-Linux, the `-m32` compiler flag had to be removed, since it is not recognized by the AVR32 compiler.

3.2 Timer Precision Test

To execute the RTW generated code periodically, the operating system must provide a timer, with a period equal to the Simulink models time step. What period is required, will depend on the physical system, if it is fast changing it will require a short time step period.

To fulfil the real-time constraint of a periodic control system, two conditions must be met. Firstly, the system must be able to create precise timers. Secondly, all calculations needed for a time step must be completed during that period. The shorter the period, the harder the second condition will be, since the same calculations must be done in less time. If the real-time constraint is broken, the control system will not behave correctly, and give undesirable results.

To implement timers on the AVR32-Linux platform, an internal Linux timer, with a maximum resolution of 1000Hz or 1ms, was used. AVR32 is considered to have too low capacity to run RTW code with a shorter time step than 1ms, so using the internal Linux timer is acceptable. This is a limitation of the hardware, Linux is able to generate shorter periods [4].

Table 1. Results of preliminary tests

CPU	**Pentium 4**	**AT32AP7000**
OS	**Ubuntu Linux**	**AVR32 Linux**
Clock	**2.4GHz**	**140MHz**
Average Periode	**100013μs**	**9995μs**
Periode Std.Dev.	**2057μs**	**7μs**
μs/float	**0.00273**	**0.492**
cycles/float	**6.55**	**68.8**
μs/fixed	**0.00269**	**0.123**
cycles/fixed	**6.47**	**17.2**

It is not enough that the timer has sufficient resolution, it must also be accurate. This was tested with a small program that logged the time between 10000 periods of 10ms generated by the operating system. For comparison, the test was run on an Ubuntu Linux desktop computer as well as the AVR32-Linux system. A standard Ubuntu Linux have a minimum timer period of 10ms, which is the reason this period was used. 10ms is also a realistic period for the AVR32 system. The averages and standard deviations are shown in Table 1.

The average time is very close to 10ms for both platform. A 0.05% deviation for the AVR32 is very good. The standard deviation describes how accurate the timers are, and the AVR32-Linux performs much better than the Ubuntu desktop computer. A standard deviation of 0.07% of the period is not a noticeable inaccuracy.

The most likely reason that the AVR32 performed much better than the Pentium 4, is that the AVR32-Linux kernel was a preemtable kernel, while the Ubuntu kernel was not. [8] discusses the how a preemtable kernel will give lower latency for audio/video, and it will have the same effect for timers. If the kernel is busy when the timer triggers, it can interrupt a preemtable kernel, but it has to wait if the kernel is unpreemtable. Preemtability has been a compile option for all 2.6 versions of Linux, but normal desktop systems like Ubuntu does not use it by default.

The time measurements in these tests used the `GetTimeOfDay` function in Linux. It returns results in microseconds, but might not be that accurate. It can also be a problem that the system is used to measure itself. If the timers are wrong because of a problem in the system, measurements from the same system might not be able to detect it. This method was still used, because it is more practical than setting up an external measurement system. Because of a large sampling size and since accuracy down to 1μs is not necessary, this measurement method was considered to be accurate enough for this purpose.

3.3 Floating-point performance

If the selected platform has any other potential weaknesses, these should also be tested. For the AVR32 platform, one such potential weakness was floating-point operations. The AT32AP7000 CPU does not have a floating-point unit (FPU), witch means low floating-point performance. Simulink models use floating-point numbers by default, but can be converted to fixed-point [5], which might require swapping some unsupported blocks with supported ones. It is important to know the CPU floating-point performance, so it can be considered when deciding if this extra work is necessary and have the desired effect.

A test was performed, consisting of 100 million multiplications. Floating-point and fixed-point multiplications were tested on both the AVR32 and the Pentium 4 platform, the time and number of CPU cycles used in average for each test is shown in Table 1.

AT32AP7000 CPU used 4 times as many cycles on a floating-point multiplication than a fixed-point. This is regarded as a positive result, since the lack of FPU would suggest that the floating-point performance could be even worse. The Pentium 4 is considerable faster than the AVR32, as expected. It also used the same time for both fixed- and floating-point multiplications, since it has a built in FPU.

3.4 Preliminary test conclusion

The preliminary tests should be able to reveal if the selected platform is able to perform the task it is intended for, and if there are any weaknesses that is important to be aware of. The AVR32 platform is able to run RTW generated code, and generate accurate timers. The only problem is the poor floating-point performance, so it should be avoided if effective code is important.

4 AVR32 Real-Time Target

4.1 Custom RTW target

As described in 2.3, a custom RTW target can be created based on an existing one. It can be modified so it will generate code and build an application with the required properties.

The three RTW system targets described in 2.3 can be used as the basis for a custom target. AVR32 used the GRT target as a basis, since 3.1 showed that it is possible to run GRT code on AV32, and it is more widely available than ERT. The GRT malloc was not tested, but should offer no advantages over the normal GRT for this purpose.

If GRT code is not running on the selected platform, or if the goal is to create production code, then the ERT target should be considered. The ERT is not too different from GRT, so the rest of this paper should still be relevant.

AVR32 real-time target is different from the GRT in two ways. It builds the application for the AVR32 architecture, and it implements timers, for real-time execution. It consists of the three files described below, that are placed in the folder `MATLAB_ROOT/rtw/c/avr32`.

4.2 `avr32.tlc`

This is the system target file, based on `grc.tlc`, and it controls how code is generated. The following changes were done:

- `avr32.tmf` was set as Makefile template.
- The build directory suffix was set to `_avr32_rtw`.

4.3 `avr32.tmf`

This is the Makefile template, which is based on `grt_unix.tmf`, the standard Makefile template for GRT on UNIX/Linux platforms. The following changes were done to compile the code for AVR32:

- `avr32.tlc` was set as system target file.
- `avr32-linux-gcc` was set as compiler.
- The `-m32` compiler flag was removed, since it is not recognized by the compiler.
- `-lpthread` was added to linking options, to enable the pthread library.
- `avr32_main.c` was selected as the applications main file.
- Updated the folder of the AVR32 target files.

4.4 `avr32_main.c`

When RTW generates code, it does not create a main function, which is needed for a C application. The main function is defined by a RTW system target specific file, and modifications to this file will affect all applications built using that target. `avr32_main.c` is based on `grt_main.c`, with the addition of code for periodic execution of the time steps. Only one periodic task was needed, so threads were not used. When all the code for one period is executed, the program will wait until the start of the next period. It is indicated in `grt_main.c` where code should be added for this purpose.

Code for logging calculation time for each time step, and other debug information can also be added to this file where appropriate. This can be used for evaluating the performance of the control system.

5 S-functions for I/O

5.1 Type of S-function

To create code for I/O operations, S-functions were created, which allows I/O channels to be represented by blocks in the Simulink model. The S-function only contain user-space code, so if a device driver is needed, it must be loaded beforehand. The S-function is supposed to be inactive during a Simulink simulation, which means that an fully inlined S-function is the best choice since the functionality of a simulation and RTW code can be different.

How many I/O channels one S-function should represent is a design question. Either one S-function for each channel, one S-function for all inputs and another for all outputs, or anything between. It should be avoided that one S-function have both input and output channels, since inputs should be executed in the start of a time step, while outputs should be in the end. This will not be possible if they are in the same blocks, because RTW code executes one block at the time.

5.2 S-function code file

The S-function code file decide the functionality of the S-function during Simulink simulations. During simulation, the I/O blocks are supposed to stay inactive, so the basic `sfuntmpl_basic.c` template was used. The only modifications were to define the number of inputs, outputs and parameters of the block, which is defined in this file no matter what type of S-function it is.

5.3 Target block file

The target block file specify how RTW should generate code for the S-function, and is written in the TLC language. To make S-functions for an I/O card two TLC functions were used. The function `InitializeCondition` generates code that runs when the application is started, and code needed to initialize the I/O card should be added here. The function `Outputs` generates code that runs once every time step, and should be used for calculating the S-function outputs, often based on the inputs.

If one S-function generates code for more than one I/O channel, it is important that code only is generated for channels that are connected in Simulink. This is because I/O operations are time consuming, and performing unnecessary operations is ineffective.

6 Demonstration Experiment

To verify that the solution is working, a small test was performed with the use of an AVR ATMega128 acting as an I/O card, with a SPI interface to the AVR32. To enable SPI communication in Linux a device driver [7] was developed.

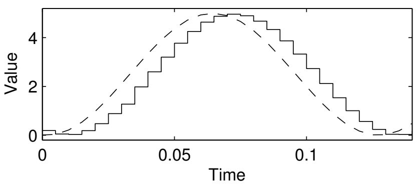


Figure 1. Result of sine test

The purpose of the test was to send a sine signal from an output to an input of the I/O card, with a time step of 5ms. Fig. 1 shows the original sine signal together with the signal sent through the I/O card. The second signal is updated every period (5ms), with a 5ms old value from the original signal.

The time used for accessing the different channels on this I/O-card was also tested. An input channel took about 300μs, while an output took about 250μs.

7 Discussion

The creation of an AVR32 real-time target and S-functions for the I/O card, makes it easy to build a real-time control system application for an AVR32 processor from a Simulink model. When selecting this target in the RTW option page, the whole process is automated by clicking the build button. This is a quick method of creating an application without the possibility of human coding errors. It allows for efficient testing and debugging the Simulink model on a physical system.

The preliminary test 3.3 confirmed that floating-point performance is a weakness for the AT32AP7000 CPU, since it does not have a FPU. This suggest that the number of floating-point operations should be minimized. To avoid that RTW uses floating-point operations, the Simulink model must be converted to fixed-point. This will make the AVR32s calculations more effective, but will require extra work, limit the possibilities in Simulink, and possible increase overhead.

A small experiment was set up, and it demonstrated that a simple system with an I/O card worked as defined in the Simulink model. A signal was sent from an output channel, and the correct signal was received on the input channel connected to this output. It was only delayed by one period. When a value is changed by the output, it is not possible for the input channel to see the change before the next time period, so one period delay is the best possible result.

The tests of the I/O card showed that each I/O operations took between 250μs and 300μs. At least one input and one output is needed for a working control system, and such a system will use more than 500μs each period on just I/O operations. This confirms that it is no purpose for having timers that are faster than 1ms when using the AVR32 and this I/O card. It is possible to use timer period down to 1ms, but the 10ms period that was tested in 3.2 will be a better choice for most applications.

The AVR32 RTW system target was based on the GRT target that comes with RTW, but it could also been based on the ERT target. The main reason for using the GRT is that it is available for everybody that have RTW, while ERT will require the additional Embedded Coder add-on. Since the AVR32 is a 32 bit CPU running on Linux, the platform is close enough to a x86 Linux system that the GRT code is possible to use. For other platforms this might not be the case, and the ERT is an alternative, since it provides more detailed control over the code generaton. It is also the preferred choice for code that will be used for final products.

All the necessary steps for implementing a similar solution for other embedded platforms are described in 3, 4 and 5. Below is a short list over questions that must be answered when following these steps, based on the experience from AVR32.

- Are the hardware architecture and the operating system compatible with RTW generated code?
- What are the platforms performance limitations?
- Can it create timers of sufficient resolution and accuracy for its intended use?
- Which RTW system target should be used as a template?
- What must be changed in the Makefile template for the program to compile properly?
- What code must be added to the main file to make it execute the code periodically?
- Which S-functions must be created for interfacing the I/O card?
- Will the I/O card require development of kernel drivers?

8 Conclusion

Real-Time Workshop has been adopted to build a real-time control system application for the AVR32 platform based on a Simulink model, using a modified generic real-time target. The same approach should be possible for other unsupported embedded systems, by using either the generic real-time target or the embedded real-time target from the Embedded Coder add-on. The ERT target should be considered for production code, since it is optimized for that purpose.

Creating code with RTW is an automated code generating process, that will be time efficient and prevent human coding errors. These are both important economical concerns, since it will allow for faster and cheaper development of control systems, and the complete product will be more reliable. It can also be used for educational purposes, and make it possible for students to quickly test their Simulink models on a physical system.

For the AVR32 and other CPUs without FPU, the Simulink model can be converted from using floating-point numbers to fixed-point for the most effective code execution. It is advised to consider if fixed-point should be used before creating the Simulink model, since it can make the conversion process easier.

References

[1] Atmel Corporation. *AT32AP7000 Preliminary*.

[2] Atmel Corporation. *AVR32 Architecture Manual*.

[3] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages*. Pearson, 3 edition, 2001.

[4] Z. Chen, X. Luo, and Z. Zhang. Research reform on embedded linux's hard real-time capability in application. In *Proc. Int. Conf. Embedded Software and Systems Symposia ICESS Symposia '08*, pages 146–151, 2008.

[5] B. Chou and T. Erkkinen. Converting models from floating point to fixed point for production code generation. *MATLAB Digest*, 17(6), 11 2008.

[6] W.-S. Gan, Y.-K. Chong, W. Gong, and W.-T. Tan. Rapid prototyping system for teaching real-time digital signal processing. *Education, IEEE Transactions on*, 43(1):19–24, feb 2000.

[7] A. R. Greg Kroah-Hartman, Jonathan Corbet. *Linux Device Drivers*. O'Reilly, 3rd edition, 2005.

[8] R. Love. Lowering latency in linux: Introducing a preemptible kernel. *Linux Journal*, 97, May 2002.

[9] K. H. Low, H. Wang, and M. Y. Wang. On the development of a real time control system by using xPC target: solution to robotic system control. In *Proc. IEEE Int Automation Science and Engineering Conf*, pages 345–350, 2005.

[10] P. S. Shiakolas and D. Piyabongkarn. On the development of a real-time digital control system using xPC-target and a magnetic levitation device. In *Proc. 40th IEEE Conf. Decision and Control*, volume 2, pages 1348–1353, 2001.

[11] A. Skavhaug, T. Lundheim, B. Vik, and T. I. Fossen. A decade of rapid software development for control system experiments.. lessons learned. *Proceedings of the 15th IFAC World Congress 2002*, 2002.

[12] The MathWorks Inc. *Real-Time Workshop 7 Embedded Coder Users Guide*.

[13] The MathWorks Inc. *Real-Time Workshop 7 Users Guide*, 2010a edition, 04 2010.

Paper B Prototyping and Evaluation of a Telerobot for Remote Inspection of Offshore Wind Farms

Published in the *2nd International Conference on Applied Robotics for the Power Industry* (CARPI 2012) in Zurich, Switzerland, September 2012.

Prototyping and Evaluation of a Telerobot for Remote Inspection of Offshore Wind Farms

Øyvind Netland and Amund Skavhaug,

***Abstract*—A telerobot can be used for inspection at a location far from where its user is, i.e. remote inspection. This paper presents the design and implementation of a lab for evaluation of remote inspection, consisting of a telerobot prototype and an environment for it to operate in. Remote inspection can be used in many industries, but we argue that it is especially suitable for offshore wind turbines, since these are large, unmanned and complicated machines at difficult to reach locations. Remote inspection of offshore wind turbines has the potential to reduce the maintenance cost, increase the knowledge of the turbines' condition and the predictability of maintenance. The typical user of remote inspection is expected to have expertise in maintenance, not robotics. Therefore we suggest evaluating remote inspection using usability tests, which consider how easy it is to learn, and use a system for its intended purpose.**

I. Introduction

Wind energy is expected to be an important addition to fossil fuels in the future. A large scale development of wind energy is driving the industry offshore where there are large available areas with suitable wind conditions. Offshore wind turbines can be built larger and have fewer problems with visual and audible noise, since they are located far from land and habitation. In 2011, the European Wind Energy Association (EWEA) [1] were able to identify over 130 GW of offshore wind energy projects in various stages of planning. These are ambitious plans, when there are currently just above 8 GW in operation or under construction.

A wind turbine is a large, complicated and unmanned machine that is expected to have approximately 97% availability [2]. Turbines are normally inspected and maintained on a regular basis to ensure reliable operation and avoid down time. Although there are little available operational data from offshore wind farms, the cost of operation and maintenance is expected to be significantly higher than on land, due to the increased cost of transportation, expensive equipment for heavy lifting and delays caused by harsh weather. Estimations presented in [3] indicates that between 25% and 30% of the total energy cost for offshore wind energy will be from O&M, while it is only between 10% and 15% on land. Lowering the cost of O&M is important to make offshore wind energy more economically viable, and to realise the ambitions plans.

Ribrant et al [4] have surveyed failure data from about 600 Swedish onshore wind turbines over 5 years. We have extracted the frequency of different types or failures and the average downtime caused by these, and created a bar-graph shown in figure 1. The figure shows that except for the blades and pitch, the most frequent failures originates inside the generator room on top of the tower, also called the *nacelle*.

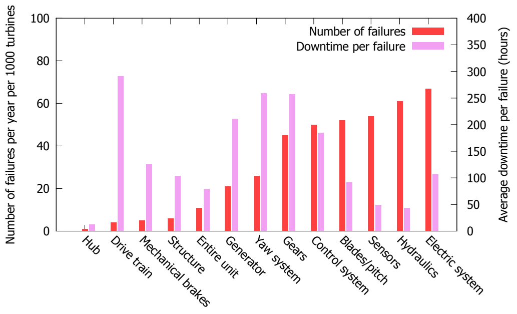


Fig. 1. Failure frequency and average downtime per failure for different parts of wind turbines

It is reasonable to expect that offshore turbines will have similar failure rates to what is described in figure 1. The downtime is, however, expected to be higher, due to the extra time needed to get available transportation, waiting for acceptable weather conditions and the transportation itself. This makes all failures more serious, but especially the frequent failures that can be repaired relatively quickly on land, will cause significantly more downtime for offshore turbines.

This article will first describe how inspection and condition monitoring of wind turbines are done traditionally (section II), followed by how telerobots can be used for remote inspections (section III). We have built a prototype of a telerobot for remote inspection of a wind turbine nacelle, which is described in section IV. The article ends with section V, describing different approaches to evaluating a remote inspection system with usability tests

II. Inspection of Wind Turbines

There are different strategies for performing maintenance, the simplest being corrective maintenance which is only performed after the machine has failed. Wind turbines are expected to have high availability and repairing failures can be expensive, so corrective maintenance alone is not a viable alternative. Preventive maintenance takes action before a failure, and can be scheduled based on time or accumulated use. Both time and use based maintenance depend on estimates for deciding when maintenance is necessary. Incorrect estimates can result in unnecessary, expensive preventive maintenance, or more seriously that a failure happens before preventive maintenance is performed.

Condition based maintenance schedule maintenance using what is known about the condition of the wind turbine, either from inspections or from automatic condition monitoring systems, to optimize the maintenance. The cost of maintenance can often be reduced without increasing the risk of failures.

A. Manual Inspection

Wind turbines are typically inspected once or twice a year together with preventive maintenance. Most of this work is performed inside the nacelle, which contain all the equipment for electricity generation and most auxiliary systems. Some equipment can be found in the hub, the pitch controller being the most important. Many turbines require climbing on the outside of the nacelle to access the hub, making it a safety concern. To inspect the blades is even more difficult.

Manual inspections of offshore wind turbines require transportation by boat or helicopter. Helicopter is a fast an expensive alternative, while boats are less expensive and slow. Regardless of the transportation method, it will be significantly more expensive and time consuming than transportation to a turbine on land. Whether access to a wind turbine is possible, depends on the wave and wind conditions. A wind turbine in the North Sea is expected to be inaccessible most of the time during the winter months.

During an inspection a technician search for signs of wear or damage to the equipment in the turbine and problems like oil leaks, loose cables etc. In some cases abnormal sound or vibrations can be detected. A commonly used tool is a thermal camera to look for areas with increased temperature, which can indicate increased friction or an electrical problem. For safety reasons, a turbine is often stopped when accessed by people. This limits what manual inspection can find, as heat, sound and vibrations are best to observe when the turbine is running.

B. Condition Monitoring

Condition monitoring analyses a large amount of sensor information from the wind turbine to estimate its current condition. It is an automated, continuous process requiring little human intervention. The condition monitoring system's diagnosis of the current health of the turbine and its prediction of future condition can be used to plan maintenance. Condition monitoring is considered to become standard equipment on future offshore wind turbines, but the number and types of sensors used will differ. There have been several research projects looking into the use of condition monitoring for offshore wind turbines [3], [5]. Unfortunately it is difficult to document the effect of condition monitoring, as it requires observation of many wind turbines for several years.

A condition monitoring system is assumed to be superior to a human for analysing sensor measurements for abnormal signals predicting failures, but is limited to its available sensors, which can't cover all unforeseen situations. When a potential problem is identified, the condition monitoring is not a suitable system for studying the problem in detail to decide which actions to take. There are also some problems that are easier for personnel in the turbine to detect visually, e.g. visible wear on parts, loose cables etc.

III. Remote Inspection of Wind Turbines

A telerobot is a programmed mechanical device that is controlled from a distance, usually with a computer interface. Telerobots can be used to inspect locations that are dangerous or impossible for humans to work in, e.g. nuclear power plants [6], the inside of large machinery [7] or inside pipes [8]. The motivation for using telerobots for inspection can also be to reduce cost, rather than doing something that is otherwise impossible. Remote inspection of the nacelle of an offshore wind turbine as described in this article is an example of this. Other parts of a wind turbine can also be inspected using robotics, e.g. the blades [9], the hub or the tower structure.

Both condition monitoring and remote inspection have the same goal; to reduce the need for manual inspections of wind turbines, but they are not mutually exclusive solutions. Condition monitoring provides a continuous monitoring of the turbine, using pre-defined sensors and diagnosis algorithms. Remote inspection benefits from the problem solving abilities of a human user, and can be used to investigate problems identified by condition monitoring in detail [10], examine unforeseen situation, in depth inspections and for planning maintenance operations.

To examine different parts of the nacelle the robot must be able to move around, and have sensors that approximate the senses a person would use during an inspection, e.g. cameras for vision and microphones for listening. Since a robot can observe the turbine while it is running, a thermal camera can potentially be even more useful than during manual inspections, when the turbine is usually stopped. However, it is not certain that these expensive cameras would be cost-effective for remote inspection, when one is required in each turbine. The robot can also have sensors for measuring temperature, vibration, voltage, current etc.

A. Usability of Remote Inspection

Usability is a term from human-computer interaction [11] that refers to how easy and enjoyable a computer system is to use, and how well it can be used to solve the task it's intended for. It's also an important concept in robotics, within human-robot interaction (HRI), and usability tests [12] are used for evaluating robots in several research projects. Among the most noteworthy is the development and usability evaluation of a robot control interface for search and rescue robots at University Massachusetts Lowell [13], and several large scale usability tests performed at Idaho National Laboratory [14], [15]. We have been unable to find any usability tests evaluating the use of telerobots for remote inspection or any experiments that compares a telerobot and a human performing the same task.

Usability is broken down into usability goals, but different sources do this differently. In the ISO 9241-11 standard [16], the usability goals of effectiveness, efficiency and satisfaction are used. In HRI the situation awareness [17] is often also considered a usability goal, because the ability to know where the robot is and be aware of its situation is so important when controlling a telerobot you are not able to see directly [18]. For the purpose of evaluating the usability of remote inspection, we define the following usability goals:

1) Effectiveness: The effectiveness of an inspection is how accurately the results from the inspection reflect the inspected system's true condition. The scheduling of maintenance is decided based on the information gained from inspections, thus effectiveness is considered the most important of the usability goals when judging inspections.

2) Efficiency: The efficiency of an inspection is the total amount of resources consumed during the inspection, most importantly the time and cost. For offshore wind turbines, we can claim that remote inspection will always be more efficient than manual inspections. Any small variations in the inspection time will be insignificant compared to the additional time and cost required to plan manual inspections, and transportation to the turbine.

3) Satisfaction: Satisfaction describes whether the inspection task feels enjoyable or frustrating. Satisfaction does not directly influence the results or cost of an inspection, but are important for a good working environment.

4) Situation Awareness: Situation awareness is defined in [18] defined as ”the perception of the robots location, surroundings, and status; the comprehension of their meaning; and the projection of how the robot will behave in the near future”. It is important for successful operation of a telerobot, as it is difficult to get the robot to do what you need, if you don't have a good understanding of its situation. Situation awareness is linked to the other usability goals, since low SA can lead to incorrect results (decreased effectiveness), longer completion time (decreased efficiency) and user frustration (decreased satisfaction).

IV. Prototype

A. Mechanical Construction

Our prototype is a semi-mobile robot moving on a rail, shown in figure 2. It grips to the rail similar to a roller-coaster, making it able to hang upside down. It is equipped with a camera on a pan and tilt mechanism, but no other sensors, as it is intended for evaluation based around visual inspection. The design was created with simplicity and low cost in mind and built at the mechanical workshop at our department.

The cost is an important factor, since wind energy is a low margin industry where large additional costs are unacceptable. The prototype has an estimated part cost of 2.000 euros, and a basic commercial version based on the concept is expected to cost 10-20 times as much. An expensive thermal camera will add to this.

In table I two methods for movement is compared. A robot able to move freely around in a wind turbine, with little floor area and many obstacles, is considered to be unnecessarily complex and expensive for our application. Movement on rail is considered to be a better alternative. The rail can be customized for the individual nacelle so the robot is able to reach all points of interest, and it can easily reach both high and low locations. The possibility of powering the robot through the rails, and not depend on batteries, is also important. A fully mobile robot must use batteries, which increase weight, reduce lifespan, and introduces the risk of completely disabling the robot by running out of batteries without reaching a charging station.

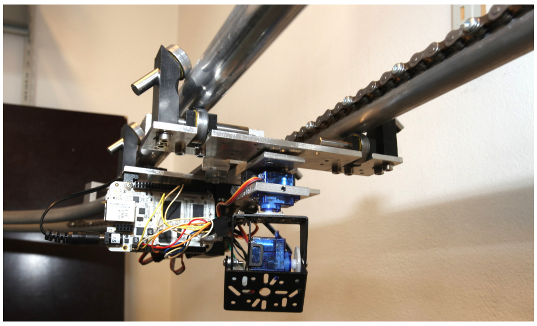


Fig. 2. Our prototype without cover and camera, hanging from a section of the rail.

TABLE I
Comparison between movement on rail and full mobility

	Advantages	Disadvantages
Movement on rail	• Simple and reliable. • Can be powered through rails. • Easy to know exact position. • Not dependent on the environment to move.	• Can only reach where the rail goes. • Require installation of rail. • Where the robot can move is decided when installing the rail.
Full mobility	• Can potentially reach everywhere.	• Obstacles limit movement. • Reach higher ground (climbing) is difficult. • Needs batteries for power. • Can get stuck.

The rail used by the prototype consists of two aluminium pipes as shown in figure 2, where one of the pipes has a bicycle chain attached to its side. This solution was chosen for the low cost and availability of materials, and works well for prototyping. A commercial product based on this concept must use a rail that is more suitable for mass production, e.g. a monorail extruded in aluminium. The bicycle chain is a simple rack and pinion implementation, a technique often used for railways with steep gradients. With this solution, the prototype is able to move on a vertical rail.

B. Control System

The robot is controlled by a small, single board ARM computer, called *Beaglebone*, running a Linux operating system. A client program controls the robot and displays information about itself and from its sensors. It is implemented in Java, and runs on a normal desktop computer. The connectivity between

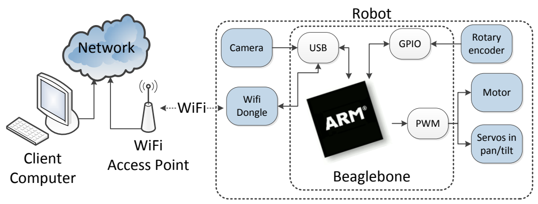


Fig. 3. Control system design

the Beaglebone and client is through a network, as shown in figure 3. The wireless connection between the robot and the network is encrypted. The Beaglebone board is able to communicate directly with all the sensors and actuators on the robot, using USB, GPIO and PWM signals, reducing the need for additional electronics.

Since the robot is moving on a rail, it can only move in two directions, forward and backwards. The default control is direct teleoperation, where the desired speed is set in the user interface, and the robot is moved forward or backward with keyboard or mouse clicks. The robot's speed and direction can also be controlled with an analog joystick on a gamepad, which is considered a more intuitive method with more accurate control. The speed of the robot is derived from an optical encoder, and used by a PI controller to set the PWM signal controlling the motor. The operator can also control the position instead of the speed of the robot. Position control can also be used for autonomous inspection, where the robot can autonomously move around in the turbine and collect data, e.g. take photos or video. Images of interesting locations taken at regular intervals make it possible to observe change or wear over time, which can be a useful tool for understanding the current condition.

V. Approach for Evaluating the Usability of Remote Inspection

Remote inspection of offshore wind turbines is a potential alternative to manual inspections. We don't, however, know how remote inspections would work in practice, as it is not commonly used, nor have we found any experimental results. This section describes three different variants of experiments, shown in table II, for evaluating the usability of remote inspection and compare it to manual inspections.

A pilot experiment is only expected to give preliminary usability results, but will provide important experience before performing other experiments. Quantitative experiments should have a large number of participants to get statistical significant quantitative results, like [19]. It is suitable to test whether a system is as usable as planned, to compare remote and manual inspections, or to compare different remote inspection systems. Qualitative experiments focus on observation and comments from the participants, and are used to explore how different solutions works and to get new ideas for improvements.

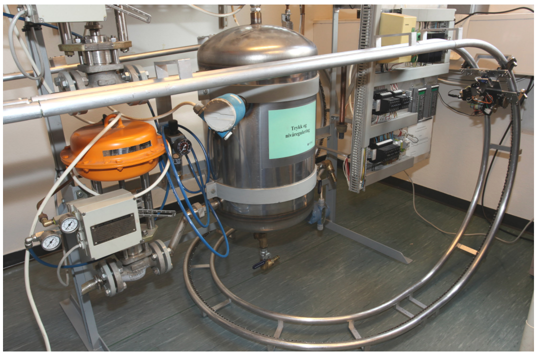


Fig. 4. A section of the lab for evaluation of remote inspection

A. Lab Environment

To test the usability of inspection methods, it is necessary to have something to inspect. One alternative is to perform the experiment in an actual industrial system, like a wind turbine. To get access to such a location is difficult, and even if access is possible it will be restricted. Being unable to change the environment as needed will make it difficult to perform a controlled experiment. There are also safety concerns, like high voltage and danger of falling, which would interfere with the experiment. In the development of a remote inspection product, it would be necessary to perform tests in a realistic environment, but it is not appropriate to do this before a prototype have been thoroughly tested elsewhere.

A better solution for prototype evaluation is in a lab that represents an industrial system. We have built such a lab, which consists of non-functional industrial equipment, as shown in figure 4. The intention is not to have an exact representation of a specific system, but to have an environment where we can introduce error markers that represent wear, early stages of failures and other problems that should be detected during an inspection. We have defined several such markers in our lab, which each can easily be switched between an error and normal state. It should be challenging to identify the markers in error state, but not so challenging that the participants aren't able to find any during inspections. Figure 5 shows two examples of markers, where the top images show normal states and the bottom ones show error state. It is important to perform a pilot experiment to test how challenging the markers are. The ones that are found to be too easy or too challenging to find will not provide useful results in a quantitative experiment. In contrast to an industrial location, we have full control over our lab, and can change it depending on what we want to test. The lab can represent pristine condition, with all markers in normal state, or worn condition, with one or more marker in error state.

B. Participants

Usability tests are supposed to have potential end users as participants, which for remote inspection would be mainte-

TABLE II
DESCRIPTIONS OF USABILITY TEST VARIANTS FOR EVALUATION OF REMOTE INSPECTION

Experiment	Purpose	Defining Characteristics
Pilot experiment	• Evaluate the lab and the experiment procedure. • Determine which error markers that should be used in future experiments. • Identify problems with the robot.	• Few participants, no special requirements. • Provide only preliminary results. • An evaluation of the experiment itself, not the telerobot.
Quantitative experiment	• Compare the effectiveness of remote and manual inspections (ratio of markers found) to determine whether remote inspection is a viable alternative. • Can also compare different variations of remote inspection.	• Many participants (20+), not realistic that they should have experience from maintenance. • Each participant performs a number of inspections using both remote and manual inspection. • To get the best quantitative results, the conditions for each inspection should be as identical as possible. – Same time limit per inspection. – Balance the use of markers over the participants and inspection methods. – Participants should not be interacted with or asked to think out loud.
Qualitative experiment	• Find the prototypes usability problems. • Get ideas for improvement of the prototype.	• Few participants, preferably with maintenance experience • The participants opinions, ideas and frustrations are more important than their performance. • Remote inspection can be tested alone or compared with manual inspection. • An accurate measure of effectiveness is not important so the experiment can be performed less stringent than a quantitative experiment.

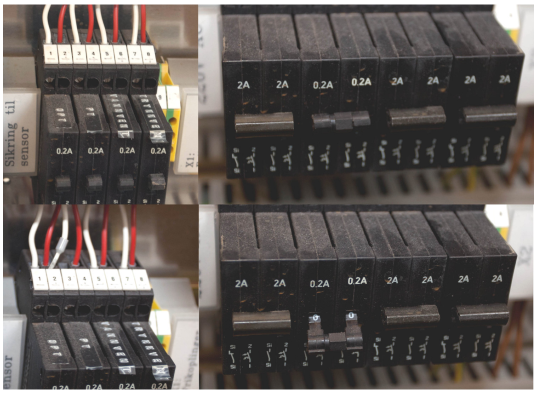


Fig. 5. Examples of error markers: normal cable (top left), loose cable (bottom left), normal fuse (top right) and tripped fuse (bottom right)

nance personnel. It is considered difficult to recruit these as participants for usability tests, especially in large numbers. If a few, or even one, of the participants in a qualitative experiment have such a background, it will significantly increase the value of the results. The opinions, criticism and ideas from people with real experience in the task the robot is supposed to do will be valuable for further development. In HRI, students or visitors to a museum or event where the experiment takes place is often used as participants. This is the most realistic method for recruiting a large group of participants, which we want for quantitative experiments.

C. Task

Before starting an experiment, the participants should be given the opportunity to look at the lab in pristine condition, meaning markers in error state. Knowing this, corresponds to knowing the original condition of equipment before inspecting it. The main task given to the participants is to inspect the lab one or more times with different error markers, using either remote or manual inspection. For quantitative experiments it is important that the use of markers and the order remote and manual inspection is performed at is balanced over the participants, to reduce the effect learning will have on the results. The time available for each inspection should be the same, to keep as many variables constant as possible. Quantitative and pilot experiments focus on observations and comments, and are less affected by parameters of the experiments.

It is expected that doing just an inspection task will be too easy for the participants. To make it more challenging, a secondary task can be introduced. Our suggestion for such a task is to search for a unknown number of items of a certain type. The participants should be told that this task is less important than the inspection task. In addition to making the inspection more challenging, the secondary task gives us a second quantitative measurement.

D. Measurements

During an experiment we can measure the usability using some or all of the following usability measures:

1) Effectiveness: The ratio of error markers that was found is a quantitative measure of the effectiveness of the inspection. If something is incorrectly identified as an error state, it will reduce the effectiveness.

2) Efficiency: We do not have any specific measurements for efficiency, as remote inspection is considered to be inherently more efficient.

3) Satisfaction: How satisfied the participants were with an inspection can be evaluated by asking them to fill out a questionnaire directly after each inspection. For qualitative experiments, the participants should be asked to think out loud while inspecting, to give the experimenters a better understanding of how the participants think and experience the system. Think out loud should not be used during quantitative experiments, as it might affect the results.

4) Situation Awareness: How well the secondary task is performed can be used as an indicator or an implicit measure [18] of situation awareness. The more the participants are aware of the robot's surroundings, the more likely is it that objects will be noticed. Similarly to satisfaction, the subjective situation awareness can be found based on questionnaires and from observing participants that thinks out loud [20].

VI. Conclusion

We have built a lab for evaluating remote inspection, which consists of a prototype inspection robot moving on a rail in an environment that represents an industrial system. The lab can be inspected either remotely using the robot or manually, which makes it possible to compare these two types of inspection directly.

In this article, we argue that the usability of a telerobot is equally important as its capabilities for use in remote inspection. We have therefore defined three variants of usability tests that evaluate different aspects of the lab and the remote inspection concept. The intention of pilot experiments is to evaluate the lab and the experimental procedure, but they are not expected to give any conclusive results. Qualitative experiments can provide valuable opinions about the system and ideas for improvement, while quantitative experiments of remote inspection can test whether it is equally effective as manual inspection.

Offshore wind energy is expected to be an important source of energy in the future. It is, however, known that operation and maintenance will be a challenge, due to the remote location of offshore wind turbines. Our ongoing experiments will indicate whether remote inspection is as effective as manual inspection and therefore a viable alternative. The results also give directions for future field studies of inside actual wind turbines, and the evaluation of the potential for cost savings with remote inspection.

Acknowledgment

This work is performed in cooperation and with funding from Norwegian Research Centre for Offshore Wind Technology (NOWITECH). The authors will like to thank the master students that have been involved in the research, especially Viktor Fidje and Tor Karlsen, for their work with the early prototypes of our robot. Norsk Automatisering AS should also be thanked, for their financing of the early prototypes.

References

[1] EWEA, "Wind in our Sails," Tech. Rep., 2011.

[2] A. Henderson, "Offshore Wind Energy Ready to Power a Sustainable Europe," Concerted Action on Offshore Wind Energy in Europe, Tech. Rep. December, 2001.

[3] E. Wiggelinkhuizen, L. Rademakers, T. Verbruggen, S. Watson, J. Xiang, G. Giebel, E. Norton, M. Tipluica, A. Christensen, and E. Becker, "CONMOW Final Report," Energy research Centre of the Netherlands, Tech. Rep., 2007.

[4] J. Ribrant and L. Bertling, "Survey of failures in wind power systems with focus on Swedish wind power plants during 19972005," *Energy Conversion, IEEE Transactions on*, vol. 22, no. 1, pp. 167–173, Mar. 2007.

[5] G. Giebel, A. Juhl, K. Hansen, J. Giebhardt, T. Pahlke, H. Waldl, M. Rebbeck, O. Brady, R. Ruffle, M. Donovan, and Others, "CleverFarm-A SuperSCADA system for wind farms," RisøNational Laboratory, Tech. Rep. August, 2004.

[6] B. Luk, K. Liu, A. Collie, D. Cooke, and S. Chen, "Tele-operated climbing and mobile service robots for remote inspection and maintenance in nuclear industry," *Industrial Robot: An International Journal*, vol. 33, no. 3, pp. 194–204, 2006.

[7] W. Fischer, G. Caprari, R. Siegwart, I. Thommen, W. Zesch, and R. Moser, "Foldable magnetic wheeled climbing robot for the inspection of gas turbines and similar environments with very narrow access holes," *Industrial Robot: An International Journal*, vol. 37, no. 3, pp. 244–249, 2010.

[8] F. Tâche, W. Fischer, G. Caprari, R. Siegwart, R. Moser, and F. Mondada, "Magnebike: A magnetic wheeled robot with high mobility for inspecting complex-shaped structures," *Journal of Field Robotics*, vol. 26, no. 5, pp. 453–476, 2009.

[9] N. Elkmann and T. Felsch, "Robot for rotor blade inspection," *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*, 2010.

[10] Ø. Netland and A. Skavhaug, "Extending Condition Monitoring of Offshore Wind Farms with Remote Inspection," in *24th International Congress on Condition Monitoring and Diagnostics Engineering Management*, 2011.

[11] J. Preece, Y. Rogers, and H. Sharp, *Interaction design: Beyond human-computer interaction*, 3rd ed. John Wiley & Sons, 2011.

[12] J. Dumas and J. Fox, "Usability Testing: Current Practice and Future Directions," *Human-Computer Interaction: Development Process*, p. 231, 2009.

[13] B. Keyes, M. Micire, J. Drury, and H. Yanco, "Improving human-robot interaction through interface evolution," in *Human-Robot Interaction*, 2010, no. February, pp. 183–202.

[14] D. Few, C. Roman, D. Bruemmer, and W. Smart, "What Does it Do?: HRI Studies with the General Public," in *Robot and Human interactive Communication, 2007. The 16th IEEE International Symposium on.* IEEE, 2007, pp. 744–749.

[15] C. Nielsen, D. Gertman, and D. Bruemmer, "Evaluating robot technologies as tools to explore radiological and other hazardous environments," *Response, and Robotics*, 2008.

[16] ISO, "ISO 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Part 11: Guidance on Usability," Tech. Rep. November, 1998.

[17] M. Endsley, "Design and evaluation for situation awareness enhancement," in *Human Factors and Ergonomics Society Annual Meeting Proceedings*, vol. 32, no. 2. Human Factors and Ergonomics Society, 1988, pp. 97–101.

[18] H. Yanco and J. Drury, "Where am I? Acquiring situation awareness using a remote robot platform," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 3. IEEE, 2004, pp. 2835–2840.

[19] D. Bruemmer, R. Boring, D. Few, J. Marble, and M. Walton, "I call shotgun!: an evaluation of mixed-initiative control for novice users of a search and rescue robot," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 3. IEEE, 2003, pp. 2847–2852.

[20] J. Drury, B. Keyes, and H. Yanco, "Lassoing hri: analyzing situation awareness in map-centric and video-centric interfaces," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction.* ACM, 2007, pp. 279–286.

Paper C Two Pilot Experiments on the Feasibility of Telerobotic Inspection of Offshore Wind Turbines

Published in the *Conference Proceedings of the 2012 Mediterranean Conference on Embedded Computing* (MECO 2013) in Budva, Montenegro, June 2013.

Two Pilot Experiments on the Feasibility of Telerobotic Inspection of Offshore Wind Turbines

Øyvind Netland
Department of Engineering Cybernetics
Norwegian University of Science and Technology
Trondheim, Norway
Email: oyvind.netland@itk.ntnu.no

Amund Skavhaug
Department of Engineering Cybernetics
Norwegian University of Science and Technology
Trondheim, Norway
Email: amund.skavhaug@itk.ntnu.no

***Abstract*—With cyber-physical systems, it is not necessary to be physically present at a location to perform work there. Inspection of offshore wind farms is a task that would be beneficial to do remotely, due to the time and high cost required for accessing the turbines for manned inspections. Such remote inspections must be equally effective at finding errors in the turbines, since errors that aren't found can cause expensive failures. This paper describes a remote inspection robot prototype, and how it was used to compare participants' ability to identify errors using remote and manned inspections in two experiments. The results demonstrated that errors with both known and unknown symptoms were successfully identified using remote inspections, although not as effectively as manned. This is considered promising for remote inspections, and what we have learned in these experiments is used in the planning of a larger experiment, and in the development of an improved prototype.**

Keywords—Robotic inspection, Human-robot interaction, Wind energy, Cyber-physical systems

I. Introduction

There are many challenges for installing and operating wind turbines in offshore areas, causing problems for the ambitious plans for offshore wind. Especially operation and maintenance is difficult, as access is expensive and unpredictable. There are limited operational data available from offshore wind turbines, but it has been estimated that between 25% and 30% of the total energy cost of offshore wind energy will be from operation and maintenance (O&M), compared to only 10% to 15% on land [1].

Operation of wind turbines on land typically relies on information from manned inspections for planning maintenance. This is possible because of the relatively easy and inexpensive access, while frequent manned inspections of offshore wind turbines would be prohibitively expensive. Offshore wind turbines are located in areas with high average wind speeds for maximum energy production. Since the turbines are inaccessible when the wind speeds and wave heights exceeds a certain threshold, there can often be long periods where manned inspections, and maintenance operations, are impossible to be performed.

In this paper we investigate whether remote inspections with a cyber-physical system is a feasible alternative to manned inspections. A remotely controlled robot, or telerobot, can be equipped with sensors for inspecting the equipment inside the turbine. The use of robotics for inspections has typically been to bring an expensive robot to the site and have it access an area that are impossible or dangerous for humans to access, e.g. inside generators [2] and for examining the blades of wind turbines [3].

The main motivation for using remote inspection for offshore wind turbines is to reduce the need to visit the turbine, thus it would be counterproductive to bring the robot to and between turbines. One or more robots should be permanently installed inside the nacelle of each wind turbine. This is the room where all the equipment used for electricity production and auxiliary systems are located, thus most failures that can be detected during inspections originate from here. Since each turbine requires its own robot, the cost of the robot must be low for it to be economically viable for the low margin wind energy industry. The robot must also be able to operate unattended for long periods, so it should be highly reliable.

Before robotics can be introduced on an offshore wind farm or similar, the concept should be evaluated in a laboratory environment. Two experiments have been performed with a remote inspection robot prototype as the first part of such an evaluation. The purpose of these was to determine whether remote inspection is feasible as an alternative to manned.

II. Equipment

A. The Robot Prototype

A robot system that is permanently installed in a wind turbine must be low cost and highly reliable, thus we want to build it as simple as possible. A prototype of such a robot for laboratory testing has been designed and built at our department (figure 1). It moves on a rail, because we consider this advantageous when doing inspection tasks in an enclosed area, like a wind turbine nacelle, which are packed with equipment. It is a simple way to get the robot up from the floor, and closer to the equipment that is being inspected. A freely moving robot would need to climb to achieve the same, which increases the cost and complexity, while the reliability will be reduced. Because the robot grips to the rail, similarly as a roller coaster, it can't fall off the rail and cause damage to itself or nearby equipment. This is considered an important safety feature. The rail also makes it easy to know the robot's position and the robot can be powered through the rail, both simplifying the robot and reduce its cost.

The prototype is equipped with a 1080p USB camera from Creative, on a pan and tilt mechanism. It faces forward by

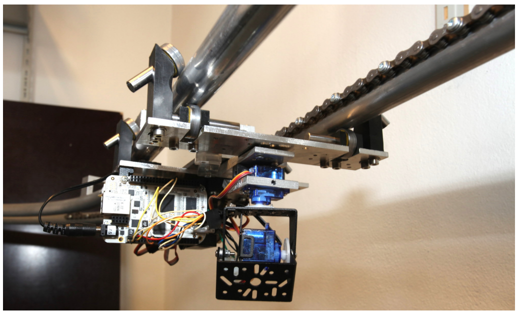


Fig. 1. The robot prototype (without camera attached).

default, and can turn approximately 90 degrees to each side as well as up and down. Since this is the only sensor available on the prototype, the evaluation will be limited to visual inspection. Other sensors will be added in future versions of the robot, including thermographic camera, microphone, temperature and vibration sensors. Especially theromigraphic cameras are expected to be useful for inspections of actual wind turbines, as it is a common tool during manned inspections.

The robot is controlled by a Beaglebone development card, with an ARM processor. It uses an Angstrom Linux distribution, which is intended for embedded applications. With many GPIO and PWM pins, the Beaglebone can connect to the motor encoder, the motor driver and the servo motors controlling the pan and tilt, with only a few additional passive electronic components. The control system is implemented in C, and communicates with a client using UDP. The camera video is streamed to the client using a small open source program called mjpg-streamer.

The robot is controlled from a desktop computer using a keyboard, mouse or a gamepad. The gamepad is considered the best control interface, and was used in the experiments. The user interface is a Java application running on a 24-inch monitor with a 1920x1200 resolution. As seen in figure 2, the interface is a typical telerobot control interface, with a large video display and a control panel on the side. Since it can be difficult for the user to be aware of the direction the camera faces, this was indicated on the screen as green lines overlaid on the video stream [4].

An inspection robot would be used by personnel with experience from inspections, who will not necessarily have expertise in controlling robots. Thus, it is important that the robot is easy to use and suitable for doing inspections, i.e. high usability [5]. One of the goals of the experiments is to improve the usability of the robot and the control interface through user centered design [6].

B. The Laboratory

To evaluate inspections, there must be something to inspect. For this purpose, we have created a laboratory environment [7], shown in figure 3. It is intended to be a mock-up of visually similar equipment as one might find in an industrial system that can be observed during inspections. However, it is not considered a replication of a wind turbine. Around this equipment, a rail for the remote inspection prototype is installed. It consists of an upper part, lower part and a transition between these. Only the larger upper part is used during the experiments.

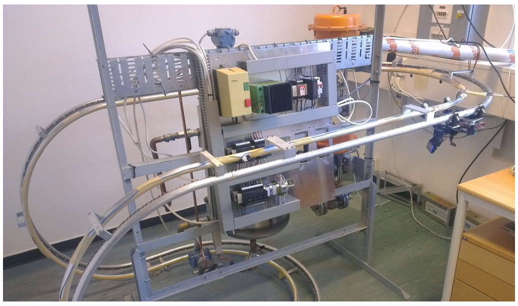


Fig. 3. The laboratory environment

The purpose of inspections is usually to identify wear, damage and other problems before they cause more serious failures. We divide these problems into two groups; errors that can be identified by recognizing known symptoms or patterns, and errors that have unexpected symptoms or symptoms unknown to the inspector.

To evaluate the participant's ability to find errors with known symptoms, a number of paper clips were positioned in the laboratory before each of the inspections. Paper clips were used since they are easily recognizable, but small enough to be hard to find. They are also easy to attach to most objects. The number of paper clips that were found during an inspection was used as a measure for the effectiveness for finding errors with known symptoms.

Evaluation of the ability to find errors with unknown symptoms is more complicated. We have defined eight error makers that represent wear, damage or other errors in the equipment, based on information from actual inspection procedures and interviews with maintenance personnel. Each of these was unique, as finding one should not give an advantage when looking for the others. The markers were designed to be recognizable as error conditions for the untrained participants in our experiments, but were not intended to represent authentic errors. Even though actual errors often would be more subtle and difficult to identify, we consider untrained participants' ability to identify our error markers to be a reasonable approximation to trained inspectors' ability to find actual errors. Thus the number of error markers that were found was used as a measure of the effectiveness for finding errors with unknown symptoms.

Two groups (A and B) of errors were created, with four paper clip locations and four error markers in each group. They were divided with the intention that the two groups should have various types of errors at various locations, but that the combined difficulty of each group should be as similar as possible.

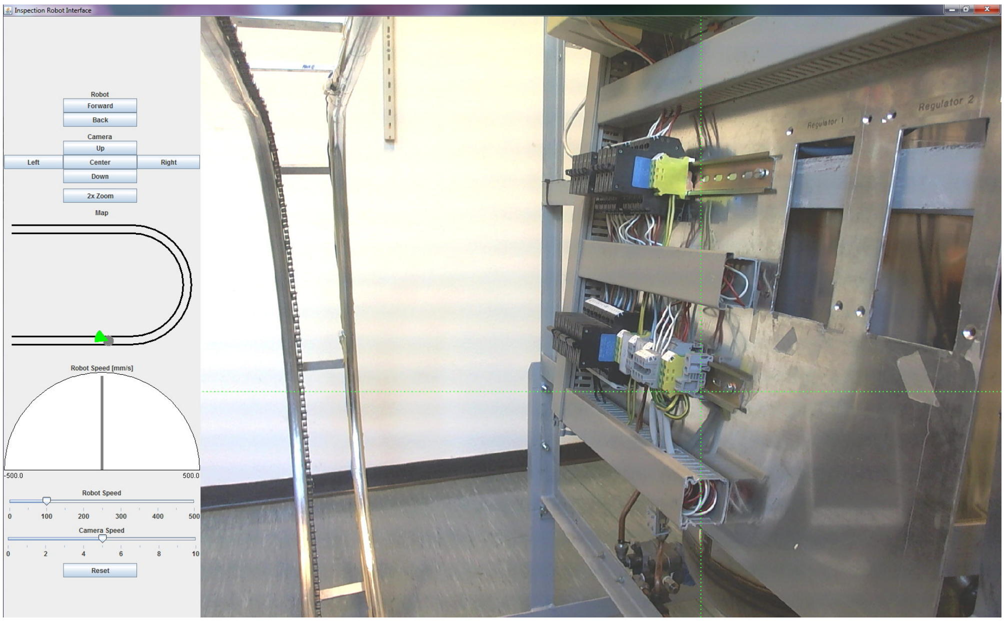


Fig. 2. The user interface

III. METHODS

A. *First Experiment*

There were four participants in the experiment, three PhD-students and one post.doc. recruited from the Department of Engineering Cybernetics were the experiment took place. None of the participants have been involved in the development of the robot and the laboratory, or used the robot before. The participants were first given 2 minutes to look and familiarize themselves with the equipment without any visible error markers. Personnel doing inspections is expected to know the original condition of the equipment. This was performed without using the robot.

Before the inspections, the participants were told that they had two tasks. The primary task was to look for signs of wear, damage or other conditions that would require maintenance. The secondary task was to look for paper clips that were hidden in the laboratory. Each participants performed two inspections, one manned and one remote. For manned inspections the participants could move freely in the room with the equipment to look for error markers and paper clips. With remote inspections the same task was performed by controlling the robot from an adjoining room. Both inspections lasted four minutes.

For each inspection the paper clips and error markers from one of the two groups were shown. Two participants had group A during their remote inspection while the other two had that group during their manned inspections. The participants did not know how many items they were supposed to find. Two of the participants performed manned inspection first, and the other did remote first, so the learning effect would skew the results as little as possible.

If there was a technical problem with the robot during remote inspections, the system would be restarted and the inspection continued with an additional 10 second time to compensate for the loss of concentration.

After both inspections were performed, the participants were given the opportunity to comment on the experience and suggest improvements in a short informal interview.

B. *Second Experiment*

The second experiment was performed with four new participants, three master students and one PhD student. As in the first experiment, all were from the department were the experiment took place, but none of them had been involved in the development of the robot and the laboratory, or used the robot before. This experiment was performed as the first one, except:

- Before inspecting, each participant were given 2 minutes to look at the equipment both with and without the robot, instead of just without the robot.
- Only the primary task of searching for error markers was given to the participants, the secondary task of finding paper-clips was not used.

IV. RESULTS

The results, sorted for remote and manned inspections, are shown in figure 4. There is a significant difference between the number of found error markers in the first experiment ($t = -2.45$, $df = 6$, $p < 0.05$), but not for paper clips in the first experiment ($t = -1.94$, $df = 3.5$, $p = 0.074$) or error markers in the second ($t = -0.655$, $df = 5.9$, $p = 0.27$).

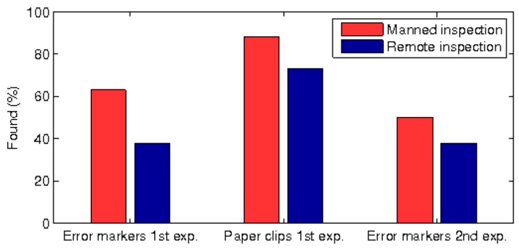


Fig. 4. Results comparing remote and manned inspections

V. Discussion

A. Discussion of the Experiments

Due to the low number of participants, the results can only be considered preliminary and not conclusive. But there is a trend, especially in the first experiment that remote inspection with our early prototype was less effective than manned. The participants of the first experiment complained that they would have performed better if they were allowed to test the robot before starting the inspection. It was observed that they wasted time in the start of the inspections to learn how to control the robot. Some of the reason for the low effectiveness of remote inspection in the first experiment can be attributed to this.

When the participants were able to test the robot beforehand, there was a smaller difference in the number or error markers. This is also realistic, as the operator of such a system would have, at least, basic training in its use.

The results demonstrate that the error markers were of suitable difficulty. All markers were identified by at least one participant, while none were found by all. The comments by the participants, both during and after the inspections, indicated that the error markers represented unknown symptoms the participants did not expect. While when the markers were noticed, the participants did understand that they were indications of errors. Thus we consider them to be suitable replacements for real errors in this application. Real errors might be more subtle in their nature and more difficult to identify, but the personnel doing real inspections would also be more experienced than the participants in these experiments.

The task of looking for paper-clips where removed from the second experiment because it was observed that some participants prioritized finding paper clips and ignored the errors with unknown symptoms. A likely reason for this was that the participants found the task of looking for unknown symptoms frustrating, and it was easier to focus on the more clearly defined paper clip task. In the second experiment, it was observed that when getting frustrated, the participants tended to give up. This is not the effect we intended to accomplish by removing the paper-clips, thus we advise keeping the paper-clips in future experiments. If nothing else, the paper-clip task keeps the participants from giving up.

B. Usability Issues

Based on the results from these experiments and comments from the participants, we have identified the following usability issues in our prototype:

- The camera could only be turned approximately 180 degrees, which limited its use.
- The robot movement are controlled by moving the left joystick forward or backward. When the camera looked to the side, this forward and backward movement of the joystick would create a sideways motion from the user's perspective, which was reported to be confusing.
- The map and controls on the side of the user interface use a large portion of the screen size, which instead could be used to show a larger video display.

VI. Conclusions

Inspection of offshore wind turbines performed remotely with a cyber-physical system can be less expensive and more predictable than traditional manned inspections. There is a large potential economic benefit of this, especially because of the high cost of transportation offshore. The experiment presented in this paper is the first of a series of planned experiments for determining how effective remote inspection, using an inexpensive telerobot, can be compared to manned inspections.

The laboratory used for the two experiments is not a realistic representation of an offshore wind turbine, but the inspection task given to the participants is considered to be an adequately realistic inspection task. Although the results show that remote inspection is less effective than manned inspections, we consider the results promising for inexpensive remote inspection of offshore wind turbines. The comments from the participants will inspire improvements in the prototype that will be evaluated again in a new and larger experiment currently being designed. The larger number of participants will give results of higher precision, so the difference in effectiveness between the methods, if any, can be determined with more confidence.

References

[1] E. Wiggelinkhuizen, L. Rademakers, T. Verbruggen, S. Watson, J. Xiang, G. Giebel, E. Norton, M. Tipluica, A. Christensen, and E. Becker, "CONMOW Final Report," Energy research Centre of the Netherlands, Tech. Rep., 2007.

[2] G. Caprari, A. Breitenmoser, W. Fischer, C. Hürzeler, F. Tâche, R. Siegwart, P. Schoeneich, F. Rochat, F. Mondada, and R. Moser, "Highly compact robots for inspection of power plants," in *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*. IEEE, 2010.

[3] N. Elkmann and T. Felsch, "Robot for rotor blade inspection," *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*, 2010.

[4] M. Baker, R. Casey, B. Keyes, and H. A. Yanco, "Improved interfaces for human-robot interaction in urban search and rescue," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 3, 2004, pp. 0–5.

[5] ISO, "ISO 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Part 11: Guidance on Usability," Tech. Rep. November, 1998.

[6] J. A. Adams, "Critical considerations for human-robot interface development," in *Proceedings of 2002 AAAI Fall Symposium*, 2002, pp. 1–8.

[7] Ø. Netland and A. Skavhaug, "Prototyping and Evaluation of a Telerobot for Remote Inspection of Offshore Wind Farms," in *Applied Robotics for the Power Industry (CARPI), 2012 2nd International Conference on*, 2012.

Paper D Software Module Real-Time Target: Improving Development of Embedded Control System by Including Simulink Generated Code into Existing Code

Published in the *Conference Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications* (SEAA 2013) at Santander, Spain, September 2013.

Software Module Real-Time Target: Improving Development of Embedded Control System by Including Simulink Generated Code into Existing Code

Øyvind Netland, Amund Skavhaug
Department of Engineering Cybernetics
Norwegian University of Science and Technology
Trondheim, Norway
oyvind.netland@itk.ntnu.no

***Abstract*—The method presented in this paper, *Software module real-time target* (SMRT), aim to make code generated by Simulink Coder easy to include into a real-time embedded code project. It is intended to give developers experienced in embedded programming better control and flexibility when using Simulink Coder, without having to understand the code generation process. A general methodology, SMRT, has been defined, and a library for using SMRT has been implemented for Xenomai Linux. The library can be modified for other operating systems. This solution has been tested on a Beaglebone development board and used in the development of a robot prototype.**

***Keywords*-Embedded Programming; Simulink Coder; Rapid Prototyping; Real-Time; Xenomai; Linux**

I. Introduction

MathWorks Simulink is a de facto standard for modeling and simulating control systems. Simulink Coder [1] (formerly known as Real-Time Workshop) generates C and C++ code, which can be used to implement a controller based on the Simulink model. Some changes are typically needed in the code depending on the embedded target computer platform and the required functionality of the control system. Several solutions have been developed to make it easy for control engineers to use Simulink Coder without expertise in embedded programming [2], [3], [4]. In practice it has, however, been shown that custom code for drivers, hardware interfaces and complex functions often are needed [5].

In this paper we describe a method called *Software Module Real-time Target* (SMRT). It uses the generated code as a module within a larger software project, instead of attaching bits of custom code to the generated code. This is shown in figure 1. The intention was not to make a ready to use solution for control engineers that does not require any embedded programming, but instead to assume that some custom programming will be necessary and tailor a solution for this. The intended users are embedded programmers that can specify platform specific code and custom functionality in languages and tools they are familiar with. There is no need to have specific knowledge of how Simulink Coder works.

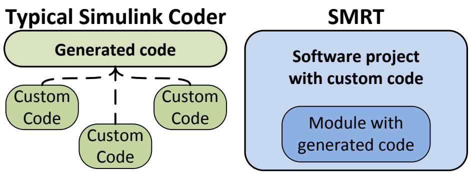


Figure 1. The relationship between generated and custom code in SMRT compared to a typical use of Simulink Coder.

II. Software Module Real-Time Target

A. System Target File

A system target file defines how Simulink Coder generates and builds code. *Generic Real-time Target* (GRT) is included in the Simulink Coder toolbox, and generates generic C-code that will run the simulation steps of the Simulink model. With some modifications this code can run on an embedded system, operate in real-time and interact with its environment with I/O signals. Alternatives to GRT include *embedded real-time target* (ERT), which provides more control over the code generation process and is intended for resource constrained embedded hardware. From our experience with GRT for embedded Linux [6] we found that the code generated by GRT was suitable for the SMRT concept. Thus it was preferred over ERT, which require the additional *Embedded Coder* toolbox, at an additional cost.

In its current form, SMRT use the GRT system target file for code generation. A possible future change would be to define a system target file specifically for SMRT, possibly making it easier to use.

B. Integration with Eclipse Project

With Eclipse, we have convenient access to all (generated and written) code and build instructions in the same IDE. The generated source files and necessary files from the Matlab installation path were easily accessible as shortcuts in the Eclipse project. A makefile based project is also possible, especially for Linux hosts where this is the norm. Both these options, together with SMRT, have the advantage that configuration options are stored within the project files

Table I
WHERE CONFIGURATIONS ARE STORED IN GRT AND SMRT.

Configuration	GRT	SMRT
Options for Makefile	Simulink config	NA
Build instructions	Makefile template	Eclipse project
Build options for external code	Makefile template	NA
Cross-compilation	Makefile template	Eclipse project
Compiler and linker flags	Makefile template	Eclipse project
Periodic execution	`main()` file	SMRT Library
Implementation of multitasking	`main()` file	SMRT Library
Calls to custom code	`main()` file and S-functions	Eclipse project
I/O and hardware interfacing	S-functions and external code	Eclipse project
Aperiodic events	External code	Eclipse project

or the SMRT library as shown in table I. If the GRT generated code was used directly, the same configuration options would be found in different locations throughout the system, making it more difficult to use without experience with Simulink Coder.

C. SMRT Library for Xenomai

GRT provides a main C-file, which creates a program that initializes and runs the code generated from the Simulink model. As this file is unchanged when code is generated, the user can modify it to enable custom functionality. The SMRT library replaces this main file by providing a library of functions for controlling the execution of the generated code. This allows the generated code to be a part of a larger program.

We have implemented an SMRT library for Xenomai Linux, and tested it on a Beaglebone ARM development board. The library provides one function for initializing the Simulink model and another that will wait as long as the model is executing, possibly indefinitely.

A Simulink model consists of blocks, where each does a specific operation. Most blocks have one sample time, which defines the time between each execution of the block, i.e. its period. Some blocks can also have several sample times, but this is less common. When initializing the Simulink model, the SMRT library starts one periodic thread for each of its sample times. The priorities of these threads are specified at compile time depending on their periods, shorter periods means higher priorities. This is according to the principle of rate monotonic scheduling [7], [8].

D. S-functions

S-functions are programmable Simulink blocks. They are useful for implementing custom functionality, communication with I/O etc. The most flexible and efficient S-functions for code generation are called *fully inlined S-functions*, and all S-functions mentioned in this article are of this type. When generating code, each block creates code that implements its functionality. For fully inlined S-functions this is defined with the TLC programming language, which is specific to Matlab and Simulink.

We have developed three S-functions for SMRT. These were implemented with TLC, but knowledge of TLC is not needed to use them. This is considered an advantage, as most developers will not know TLC. The first of the developed S-functions is an alternative to the standard *rate transition* block developed for Xenomai. It uses a Xenomai mutex to ensure that only one thread can access the resource stored by the block at the same time. It also implements priority inheritance [9] to prevent priority inversion. This block can be modified to use mutex functions of other real-time systems if needed.

Two other S-functions were created for communication between the Simulink model and the rest of the program. These are called input and output S-functions. The input S-function lets the user specify a function defined outside Simulink and uses the value this function returns as a signal in the Simulink model. The output S-function lets the user specify a function that takes a Simulink signal as one of its parameters. Exactly what these functions will do is entirely up to the programmer. In figure 2 it is shown how the generated code can interact with the written code and I/O. Different parts of the program can be implemented entirely in Simulink, as written code or a combination of these.

Table II
EXECUTION TIME AND PERIODS OF TASKS USED IN SCHEDULING TEST.

Task	Period	Execution time	Priority
A	$40ms$	$10ms$	94
B	$50ms$	$12ms$	93
C	$70ms$	$20ms$	92

III. TESTING OF SMRT ON XENOMAI LINUX

We have performed three tests with our SMRT implementation for Xenomai on a Beaglebone development board.

A. Task Scheduling Test

The SMRT library has been implemented to execute its multiple threads according to the rate monotonic principle. We wanted to test that it behaved as expected, and compare it with other uses of generated code. Three S-functions were created that performed busy-wait loops measured to take $10ms$, $12ms$ and $20ms$ of execution time.

In a small test, each of these S-functions was executed periodically, as defined in table III. Their combined utilization is $77.6\% \leq m(2^{\frac{1}{m}} - 1)$ for $m = 3$ [7], meaning that an ideal rate monotonic scheduling would guarantee no deadline misses.

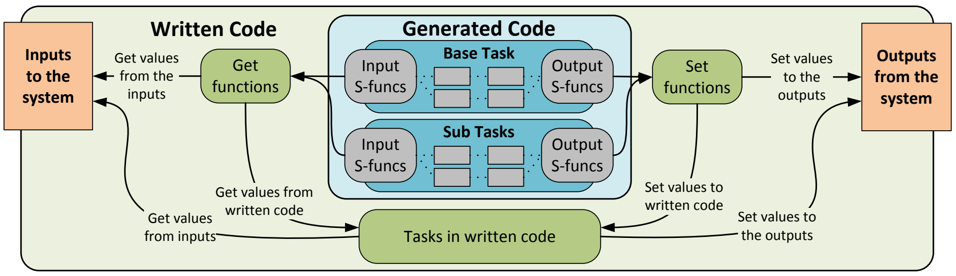


Figure 2. Graphical representation of communication between generated and written code in SMRT.

Table III
RESULTS FROM SIMULATIONS RUNNING FOR 3 HOURS.

	Results	GRT + Xenomai	SMRT + Xenomai	SMRT + Linux
A	Average response time	$10.3ms$	$10.0ms$	$10.3ms$
	Max response time	$10.4ms$	$10.1ms$	$198.5ms$
	Ratio of deadlines missed	0.000%	0.000%	0.003%
B	Average response time	$17.7ms$	$17.0ms$	$17.2ms$
	Max response time	$22.9ms$	$22.2ms$	$230.9ms$
	Ratio of deadlines missed	0.000%	0.000%	0.004%
C	Average response time	$44.7ms$	$41.7ms$	$41.0ms$
	Max response time	$66.5ms$	$64.0ms$	$334.6ms$
	Ratio of deadlines missed	0.000%	0.000%	0.007%

The Simulink model was run with three different configurations. The first used GRT modified to run each Simulink task as a Xenomai real-time thread. The second was SMRT using Xenomai threads, while the last was SMRT using normal Linux threads. The response time was defined as the time from when a period was supposed to start, until its execution was completed. This was measured over a period of three hours, and the results are shown in table III.

Using response time analysis [8] on the set of tasks, we find that the theoretical worst case response time is $10ms$ for task A, $22ms$ for task B and $64ms$ for task C. The worst case results when using Xenomai (both GRT and SMRT) were, in this test, indistinguishable from these theoretical values. This is as expected and demonstrates the hard real-time capabilities of Xenomai. Whether SMRT or GRT was used had no observable effect on the scheduling. Even though our test suggest that SMRT execution steps have a predictable worst case performance, it will not be predictable if its Simulink model contains blocks with unpredictable execution times.

As expected, normal Linux threads result in significantly higher worst case execution times. There were also multiple deadline misses during the 3 hour test. This shows that when SMRT is used for applications that have hard real-time requirements, normal Linux is not suitable.

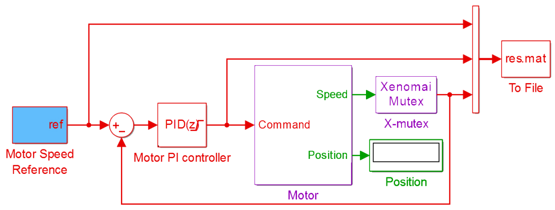


Figure 3. Simulink model used for speed control of a DC motor (Colors indicates sample times).

B. PI Speed Control of DC motor

SMRT was used to control the speed of a DC motor, with the Simulink model shown in figure 3. This demonstrated its applicability for a control task and validated the S-functions that have been created.

A rotary encoder was handled by interrupt routines in the code outside of Simulink. The calculated position was stored in a variable accessed by an input S-function. The motor was controlled from the Simulink model with an output S-function that used a manually written motor driver. This motor driver and other I/O operations used Xenomai real-time drivers (RTDM) that had been developed for the Beaglebone board, not the normal Linux drivers. This was to ensure low worst case access times to I/O.

The Xenomai mutex S-function was tested by sampling the encoder position with a slower sample time than the PI controller runs at. This is indicated by the different colors of the blocks in figure 3. Due to the low resolution of the rotary encoder, a slow sample rate beneficial to avoid a ”noisy” speed signal. It will however reduce the performance of the PI controller, but that is not the main concern of this test.

The actual speed of the motor, when following a square signal reference is shown in figure 4. The results were as expected, when considering the low resolution of the encoder. The behavior of the S-functions was as intended.

C. SMRT in Development of an Embedded System

The DC motor controller was used as a part of a larger project, the prototype of an inspection robot (figure 5)

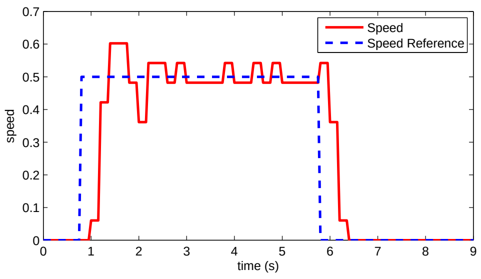


Figure 4. Plot of the speed and its reference.

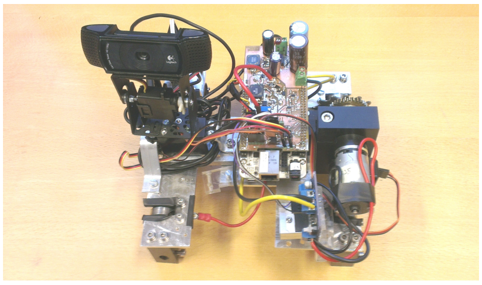


Figure 5. Robot prototype using SMRT

intended for use in offshore wind turbines [10]. Algorithms for controlling the movement and camera motion of the robot were implemented in Simulink. The generated code was integrated into a program that took care of the network communication and other background tasks.

External mode in Simulink creates a connection between Simulink and the generated code running on the embedded target. SMRT allows for this, which makes it possible to observe the signals in a running model from Simulink. Some parameters in the model can also be changed without having to restart the program. The use of external mode shortened the time needed for both implementing the control system and tuning its parameters. The simple interface between written and generated code made it easy to combine these two.

IV. Conclusion

A method for easing the inclusion of Simulink generated code into a project of manually written code has been presented. The method is called software module real-time target (SMRT), since the generated code is used as a software module available for other parts of the project to use. The main advantage of this method is that both the configuration and the written code can be specified with tools an embedded developer is comfortable with, i.e. C-code, Eclipse settings and Makefiles. S-functions have been developed to simplify the interface between generated code and the rest of the program, thus there is no need to develop S-functions for this or learn the TLC-language.

SMRT for Xenomai Linux has been tested on a Beaglebone development board. A test of the scheduling showed that it was seemingly identical with the theoretical rate monotonic scheduling and as good as other uses of Simulink generated code. There was not found any undesirable behavior for hard real-time. In addition, SMRT has successfully been used in the development of an embedded control system, and the created S-functions behaved as intended.

The future plans for SMRT is to further test and document the solution for Xenomai running on Beaglebone, including real-time I/O drivers and interfaces. This is available online [11] together with a guide for porting SMRT to other hardware and software platforms.

References

[1] *Simulink Coder User Guide*, 2013th ed. MathWorks, 2012.

[2] F. Teng, "Real-time control using Matlab Simulink," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 4, 2000, pp. 2697–2702.

[3] W. Gong, W. Gan, and Y. Chong, "Rapid prototyping system for teaching real-time digital signal processing," *Education, IEEE Transactions on*, vol. 43, no. 1, pp. 19–24, 2000.

[4] G. Quaranta and P. Mantegazza, "Using MATLAB-Simulink RTW to Build Real Time Control Applications in User Space with RTAILXRT," in *Realtime Linux Workshop*, 2001.

[5] A. Skavhaug, T. Lundheim, B. r. Vik, and T. I. Fossen, "A decade of rapid software development for control system experiments: Lessons learned," in *Proceedings of the 15th IFAC World Congress*, no. 1999, 2002.

[6] Ø. Netland and A. Skavhaug, "Adaption of MathWorks Real-Time Workshop for an Unsupported Embedded Platform," in *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*. IEEE, 2010, pp. 425–430.

[7] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, no. 1, pp. 46–61, 1973.

[8] A. Burns and A. Wellings, *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Addison Wesley, 2001.

[9] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *Computers, IEEE Transactions on*, vol. 39, no. 9, pp. 1175–1185, 1990.

[10] Ø. Netland and A. Skavhaug, "Two Pilot Experiments on the Feasibility of Telerobotic Inspection of Offshore Wind Turbines," in *Proceedings of he 2nd Mediterranean Conference on Embedded Computing*, 2013.

[11] "Software Module Real-Time Target Project Page." [Online]. Available: http://www.itk.ntnu.no/smrt/index.php

Paper E An Experiment on the Effectiveness of Remote, Robotic Inspection Compared to Manned

Published in the *Conference Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics* (SMC 2013) at Manchester, United Kingdom, October 2013.

An Experiment on the Effectiveness of Remote, Robotic Inspection Compared to Manned

Øyvind Netland
Department of Engineering Cybernetics
Norwegian University of
Science and Technoogy
Trondheim, Norway
Email: oyvind.netland@itk.ntnu.no

Gunnar Jenssen
SINTEF Technology and Society
Trondheim, Norway

Hilde Marie Schade and Amund Skavhaug
Department of Engineering Cybernetics
Norwegian University of
Science and Technoogy
Trondheim, Norway

***Abstract*—This paper evaluates the effectiveness of remote inspections using a robot in a laboratory experiment. The experiment differs from most human-robot interaction experiments in its direct comparison of manned and robotic operations. 21 participants each performed three manned inspections and three inspections with each of the two remote inspection methods; teleoperated and assisted. The effectiveness was measured based on the number of errors they were able to identify. Teleoperated inspections were found to be less effective than manned, although this difference was not statistical significant. Assisted inspections, implemented as an interactive simulation prototype representing a robot with higher autonomy, had similar effectiveness as manned. Because of the time and high cost required for manned inspections of offshore wind turbines, remote inspection can give a large economic benefit. However, this will only be a viable alternative if the robot system is inexpensive and remote inspections are as effective for identifying errors as manned inspections, which the experiment presented here suggests.**

***Index Terms*—Telerobot, Remote Inspection, HRI**

I. Introduction

Inspections are important for the reliability of industrial installations, and are traditionally performed by maintenance personnel on site. Alternatively, a remotely controlled robot on site can perform inspections on behalf of an operator located elsewhere. Such *remote inspections* are especially relevant for sites that are difficult, expensive or dangerous to access. Offshore wind turbines are large unmanned machined located at remote locations with harsh weather conditions. There is a high cost for accessing the turbines, thus regular manned inspections, traditionally used on onshore turbines, will be expensive, possibly prohibitively so. High maintenance cost is one of several obstacles for the current ambitious plans for offshore wind [1].

Inspection robots are typically expensive devices that are brought to a site and used to access areas that are impossible or dangerous for humans to access, e.g. the inside of generators [2], and on the blades of wind turbines [3]. Wind turbine nacelles are a dangerous work environment (high voltage/falling), but the main purpose of our robot is to reduce the need to travel offshore for inspections. To bring a robot between turbines would be counterproductive. The robot is instead intended to be a permanent installation inside the nacelle of the wind turbine that can be controlled remotely. With one or more robots in each turbine, the cost of each individual robot must be kept low. The robots must also be reliable enough to operate unattended for long periods. The system is intended to be installed in the nacelle of wind turbines, because the equipment that causes the most failures, both in frequency and downtime are located there, e.g. most of the electrical system, control system and the drive train [4].

When a remote inspection system has been installed, inspections can be performed inexpensively and quickly compared to manned inspections. More frequent inspections would be affordable, which could lead to an increased probability of detecting problems early.

Usability, used in *Human-Robot Interaction* (HRI), is suitable to evaluate how well non-expert robot users are able to use robots for specific tasks. Operators of remote inspection are expected to be expert in maintenance, not robotics. Usability is normally divided into different usability goals; effectiveness, efficiency, satisfaction [5]. In HRI, situation awareness and operator workload are also considered to be relevant [6].

Usability tests, were participants do relevant tasks with a robot, have shown that operators perform better when they don't control the robot directly (high level of autonomy) [7], [8]. Workload is reduced, making it easier to focus on the main task. How information from the robot is displayed is equally important, as it is easy to overlook information outside the main display [9]. If possible, information from different sources should be *fused* into the same display, e.g. show a video display within a 3-dimensional map [10] or overlay the direction of a pan and tilt camera on a video stream [11].

The purpose of the experiment described in this paper is to evaluate whether remote inspection can be as effective as manned for finding errors in a representation of an industrial system. An inferior inspection method can fail to identify errors early enough and cause expensive failures, which is not acceptable. Two variants of remote inspections, with different levels of autonomy, were compared to manned inspections. Other experiments with direct comparisons between robot and

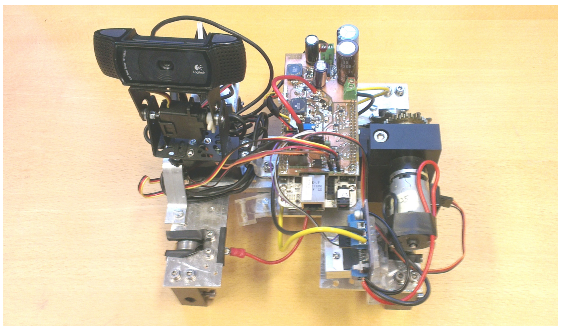


Fig. 1. The robot prototype used in the experiment.

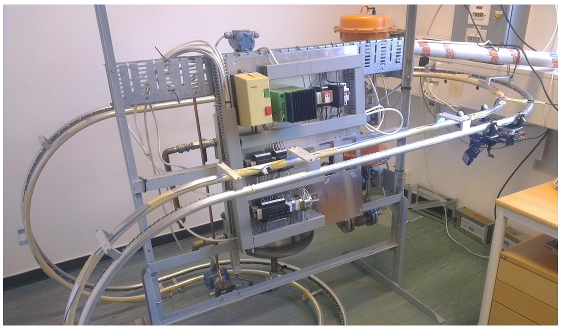


Fig. 2. Part of the laboratory used in the experiment with a mock-up of a nacelle and rail for the inspection robot.

manned operations were not found in the HRI literature. Most experiment compared the usability of different robot systems.

II. Equipment

A. Mechanical Design

The robot prototype used in the experiment has been designed and built at our department, and is shown in figure 1. Although the current prototype is not build for reliability, it moves on a rail, a proven and simple method that makes it possible to make highly reliable future versions. Movement on rail is suitable since the robot is intended to be a permanent installation in a known area. It is also appropriate for our experiments to focus on evaluating the effectiveness of inspections, not on the technical challenges of moving and climbing with a freely moving robot. The rail can be used to supply power, for easy positioning, and ensures that the robot does not fall and damage itself or other machinery. A disadvantage of using rails is the additional cost of engineering and installation of the rail itself, but this is likely less than the increased cost of a freely moving robot.

B. Robot Equipment

The prototype has a 1080p USB camera from Creative on a pan and tilt mechanism. For the experiment the camera was limited to turn approximately 180 degrees in the direction of the laboratory equipment. A robot operating in a real wind turbine, or other installations, would probably benefit from having a camera that can turn 360 degrees, or multiple cameras [12]. The robot's position on the rail is measured with an optical encoder on the cogwheel moving the robot. The position is corrected by infrared line finders when detecting pre-determined markers on the rail.

The current prototype is intended for laboratory experiments, thus it is only equipped with sensors needed for these. Future versions will have thermographic cameras for detection of hot spots from friction and electrical problems, microphones for listing to running machinery and other sensors such as temperature and vibration. The addition of a robotic arm for maintenance tasks is also possible. The rail creates a stable platform, which can support heavier equipment than a freely moving robot.

C. Control System

The control system for the robot runs on a Beaglebone development board [13] with an ARM based processor. This board is inexpensive and only require a few additional electronic components, thus it is a cost-effective solution. We used an Ångström Linux distribution, intended for embedded applications. The software is a combination of code generated by Mathworks Simulink Coder and manually written code [14]. TCP/IP is used for robot-client communication, with a wireless connection between the robot and the network.

D. User Interface

The user interface is a Java application displayed on a 24-inch monitor with 1920x1200 resolution. The user interface can be seen in figure 3. We neither wanted to display the video and map side-by-side, nor to fuse the video into a 3D-map [10]. Instead we prioritized to maximize the video display [11], as it is the operators' primary means for identifying errors. A simple map was shown, non-intrusively, on top of the video display (figure 3.a). The camera position is shown as a green rectangle on top of the video (figure 3.b) [11]. Its position illustrates the camera position, while the size illustrates the zoom.

E. The Laboratory

We have built a laboratory [15] (figure 2) designed especially for these experiments. It is not intended to be a replica of a wind turbine nacelle, but rather a *mock-up* of visually similar equipment. We have defined eight error markers that represent errors with symptoms unknown to the participants performing inspections, e.g. wear and damage. One of the markers is visible in figure 3.c. The markers can easily be added and removed, giving us full control over the laboratory's state. The success rate of finding these is used as a measure of the effectiveness for finding errors with *unknown symptoms*.

In addition to the error markers, we defined twenty locations in the laboratory where we could place paper clips. These represent errors or patterns that are known to an inspector. The success rate of finding paper clips is used as a measure of the effectiveness for finding errors with *known symptoms*.

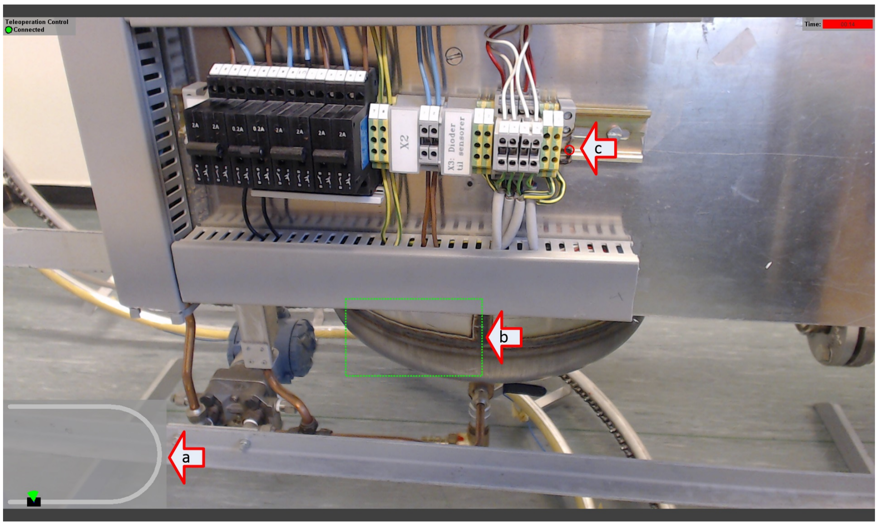


Fig. 3. The actual user interface used for teleoperation of inspection robot, some details are difficult to see due to scale but should not be important. (a: map indicator, b: green square indicating camera direction, c: burnt plastic part, identified by a participant with a red circle)

External validity [16] refers to whether the results obtained in a laboratory setting are transferable to the real-world. Our error markers have been based on information from actual inspection procedures and interviews with maintenance personnel. The laboratory has been tested in two pilot experiments [17], but has not been validated against a real wind turbine.

III. Methods

A. Participants

21 students participated in the experiment, 4 women and 17 men. All were 4th and 5th year engineering students, studying electronics or cybernetics at NTNU, the university where the experiment took place. During recruitment, it was advertised that eight randomly selected participants would receive gift cards from an electronic store. As engineering students they were considered to be more than average proficient with computers and interested in robotics, but none of the participants had previous experience with the robot used in the experiment. All the participants self-reported normal or near normal eyesight with the glasses or lenses they used during the experiment.

B. Inspection Methods

Three inspection methods were compared in the experiment.

1) Manned Inspection: For manned inspections the participants were given 3 minutes to inspect the laboratory in person. They could observe, touch the equipment and move freely around the laboratory.

2) Teleoperated Remote Inspection: During teleoperated inspections, the participants controlled the robot directly with a gamepad controller. The interface was designed to behave similarly as console games with a first person perspective. The camera was controlled with one joystick while movement and camera zoom was controlled with the other.

For teleoperated inspections, the participants were given 4 minutes and 30 seconds to inspect the laboratory. In a real-world scenario, manned inspections would require significant additional time for transportation, thus we argue that this is a fair comparison. We expected teleoperated inspection to be less effective than manned and have a higher mental workload.

3) Assisted Remote Inspection: This is a feature currently under development, where the robot moved autonomously between pre-determined locations based on input from the user. In the experiment an interactive simultation [18] ws used. It did not move the robot, but instead showed the user pre-generated images from the locations the robot would have moved to. From the users' perspective the simulation behaved as the intended future system, except for the inability to manually adjust the robot location or camera direction.

As for teleoperated inspections, the participants were given 4 minutes and 30 seconds for each inspection. We expected the assisted inspection to be an improvement to the teleoperated alternative, with similar effectiveness as manned inspection.

C. Experiment Design

The experiment was conducted over a two week period, where each participant took part in two sessions.

1) First Session: The first session started by reading a script for the participants, with a background story and instructions. They were told that they were working with inspection of wind turbines, which consisted of two tasks. The most important was to find errors with unknown symptoms e.g. wear and damage. The description was kept short on purpose and no example of such symptoms was given, to prevent the participants from understanding exactly what to look for. The second, and less critical, task was to find paper clips attached to the equipment, which represented known symptoms. The participants were told that there would be variable and unknown (to them) numbers of error markers and paper clips during each inspection, and that it could be none.

The participants were given 2 minutes with each inspection method for studying the laboratory without errors or paper clips before starting the experiment. It is expected that maintenance personnel are familiar with both the equipment they inspect and the inspection methods they use.

The first session consisted of one inspection with each of the methods, for a total of three inspections. The participants were divided in three groups, each starting with a different inspection method. For each inspection one to four paper clips were placed at randomly selected pre-defined locations in the laboratory. No error markers were used, as we wanted to establish errors with unknown symptoms as rare events, arguably a likely real-world scenario.

2) Second Session: The second session consisted of six inspections. In each of these, a randomly selected zero to two error markers was present. The number of paper clips was as before. During the nine inspections by a participant, each of the 8 error markers and 20 paper clips was shown once. The different error markers are not expected to be equally difficult to find. To balance the measurements, each error marker was shown the same number of times for each of the three inspection methods.

3) Procedures for Preventing and Handling Problems: When preparing the laboratory for an inspection, the error markers and paper clips from the previous inspection were removed and the new ones were added. An item that was accidentally present when it shouldn't was not registered, in order to not affect the results. We would not be able to know if an item was missing, by mistake, from an inspection after the fact. To prevent this, the presence of all error markers and paper clips were checked twice before starting an inspection.

If there was a technical problem with the robot, 10 seconds would be added to the remaining time after restarting.

D. Measurements

Before starting, the participants completed a form with questions about their experience with computer games.

During remote inspections (teleoperated and assisted) the participants registered their findings by clicking on the screen, i.e. with no interaction with the experimenters. For manned inspections such a solution was not possible, so the participants had to describe their findings to the experimenters that registered it manually. All findings were time stamped. Apart from the registration of findings, there was no interaction with the participants.

After each inspection the participants filled out a NASA-TLX [19] questionnaire on a computer. In addition they were asked to rate how they prioritized looking for the error markers compared to the paper clips. After the last inspection there was also a text field for comments.

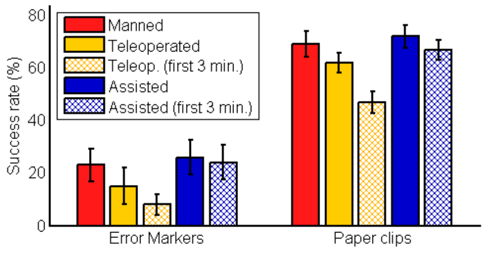


Fig. 4. Participants' success rate when using the three methods (error bars represents standard error).

IV. Results

A. Participants' Performance

The combined number of error markers each of the participants found was used to calculate their success rate with each of the three methods. The same was done for the paper clips. The average success rates are shown in figure 4. The results were analyzed with one-way ANOVA, and there were no significant differences between inspection methods for either error markers ($F(2, 60) = 0.64, p = 0.53$) nor paper clips ($F(2, 60) = 1.49, p = 0.23$).

When we only consider the first 3 minutes of the inspections there was a significant difference between the inspection method for paper clips ($F(2, 60) = 7.99, p < 0.05$) but not for error markers ($F(2, 60) = 2.33, p = 0.11$) A Tukey HSD post hoc test showed the number of paper clips found with teleoperated inspections ($M = 0.47, SD = 0.19$) was significantly different from both manned ($M = 0.69, SD = 0.23$) and assisted ($M = 0.67, SD = 0.17$).

There were not found any strong correlations between the participants' performance with remote inspection and their experience with computer games.

B. NASA-TLX

The average NASA-TLX results for the three different inspection methods are shown in figure 5. The results were analyzed with a one-way ANOVA, and there were found significant differences between the methods for mental ($F(2, 183) = 6.0, p < 0.05$), physical ($F(2, 183) = 92.57, p < 0.05$), temporal ($F(2, 183) = 13.6, p < 0.05$) and effort ($F(2, 183) = 10.43, p < 0.05$). There were no significant differences for performance ($F(2, 183) = 0.7, p = 0.5$) and frustration ($F(2, 183) = 2.71, p = 0.07$). Post hoc comparisons using the Tukey HSD test indicated the following significant differences:

- The mental workload of teleoperated ($M = 40.2, SD = 20.3$) was significantly different from both manned ($M = 32.0, SD = 16.6$) and assisted ($M = 28.9, SD = 19$).
- The physical workload of manned ($M = 45.2, SD = 20.4$) was significantly different from both teleoperated ($M = 13.8, SD = 13.4$) and assisted ($M = 10.3, SD = 12.2$).

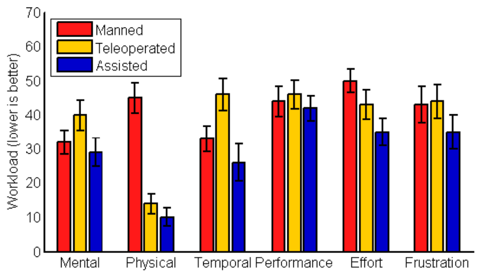


Fig. 5. Average NASA-TLX results from the inspections of the three methods (error bars represents standard error).

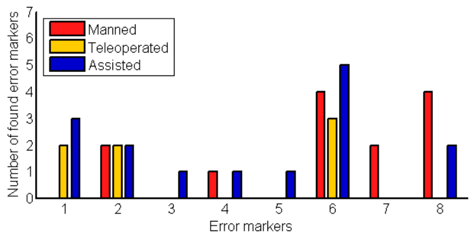


Fig. 6. Number of error markers found by the different inspection methods.

- The temporal workload of teleoperated ($M = 45.9, SD = 21.8$) was significantly different from both manned ($M = 33.3, SD = 17.8$) and assisted ($M = 25.9, SD = 24.6$).
- The effort when using assisted ($M = 34.7, SD = 18.8$) was significantly different from both manned ($M = 49.5, SD = 15.5$) and teleoperated ($M = 43.0, SD = 19.8$).

C. *Error Markers*

Figure 6 show how often each of the eight error markers was found.

D. *Participants' Comments*

Assorted comments from the participants are given below:

- Assisted inspection was the preferred method.
- It was challenging to control the robot directly.
- A method for manually adjusting the robot and camera position would improve the assisted inspection.
- It was difficult to identify the unknown symptoms

V. Discussion

A. *Teleoperated Inspections versus Manned*

There was a large, but not significant, difference in the number of errors found with teleoperated and manned inspections. Combined with the NASA-TLX results and the participants' comments, it is likely that teleoperated inspection was less effective than manned. This was expected, as the participants had to divide their attention between controlling the robot and the inspection task itself. The high temporal workload and the large number of findings towards the end of the inspections suggest that teleoperated inspections also are less efficient.

The robot control system lost power approximately five times during the experiment, which was resolved according to the predetermined plan. This is not expected to have influenced the results.

It is likely that there would be noticeable communication latency between a robot in an offshore turbine and an operator on land. This will reduce the already low usability of teleoperated inspection.

B. *Assisted Inspections versus Manned*

The results indicated a reduction in physical workload and effort for assisted inspections, but no difference in the number of found errors. These two methods are also similarly efficient, indicated by most findings within the first 3 minutes of the assisted inspections and low temporal workload (assumed to be caused by more time available).

Assisted inspection combines the best of teleoperated and manned inspections. It was shown to be as effective and efficient as manned inspection, while it can be performed comfortably, safely, inexpensively and quickly from a control center on land. In a real-world situation, assisted inspections would be more efficient than manned, since the time and cost of transportation is avoided.

Participants' comments confirmed our expectations that the relatively few images available in the interactive simulation was a disadvantage for assisted inspections. A possible improvement is to allow the operator to manually adjust the robot and camera position.

If only considering the error markers that were found relatively often (1, 2, 6 and 8 in figure 6), only the assisted inspection was able to find all these. This could indicate that assisted inspection is the more versatile inspection method.

C. *Evaluation of the Experiment*

Several factors in the experiment contributed to an increased risk of type II errors, i.e. failing to detect a real difference, of the effectiveness for finding errors with unknown symptoms.

- Few error markers divided between multiple inspections.
- The error markers being harder to find than intended.
- Limited number of participants.

The two first contributed to a low number of error markers found in total, thus each finding had a large influence in the result and contributed to the high variance. The participants in this experiment found fewer markers than the participants in a pilot experiment with the same errors. This is expected to be due to a shorter description of the unknown symptoms, and because some participants lost confidence when they didn't find errors during their first inspections (when there weren't any). Five participants didn't find any unknown errors during their nine inspections. Four of these had noticeable declines in how they reported prioritizing the task of looking for error markers, indicating that they had given up on finding any.

To reduce the risk of type II errors in future experiments, it should be considered using more errors divided on fewer inspections, although this will, arguably, be less realistic. The participants should be given a more precise description of *unknown symptoms*, but not so descriptive that they are no longer unknown. A larger number of participants would also, as always for such experiment, be an advantage.

A limitation in the experiment is that the participants were naive to the inspection task. However, recruiting 20 experienced inspectors as participants were not feasible. Since the same task is performed by the same participants using different methods, we expect the relative effectiveness of these methods to be similar regardless of the participants' experience. However, comments from potential end users of the system would benefit the development. To address this, we will, if possible, perform a smaller experiment with maintenance personnel with a strong focus on qualitative results rather than quantitative.

D. External Validity

Our test environment has not been compared with an actual wind turbine, thus the results can only be considered an indication of how a system for remote inspection would function in a real scenario. Testing in a real wind turbine would be expensive, time-consuming and there would be safety concerns to consider. A controlled experiment where errors are added and removed on demand would be impossible. To evaluate and improve the system as much as possible in a laboratory environment, even an imperfect one, would be beneficial before undertaking testing in a real-turbine.

Although the error markers are not authentic errors, they are representative since they are difficult to find, unknown to the participants, but recognizable as errors when detected.

VI. Conclusion

The laboratory experiment presented in this paper has compared two variants of remote inspections with manned inspection. The participants attempted to identify errors with known and unknown symptoms with each of the inspection methods. We have argued that the experiment is a valid comparison between the inspection methods, even if the error markers we used were not necessarily authentic and the participants did not have previous experience from actual inspections.

Of the two remote inspection methods the participants performed best with, and preferred, assisted inspection. It allowed the participants to focus more on observation and inspection and less on controlling the robot. This method performed similarly as the manned inspections, indicating that it is a viable alternative that should be developed further.

The results from the experiment demonstrated that, in a laboratory environment, remote inspection can be as effective as manned inspections. Thus, using it can be economically beneficial for operation and maintenance of offshore wind turbine. We are planning a new experiment, following a similar design, to investigate if improvements to the assisted inspection method can further increase its effectiveness.

Acknowledgment

The authors would like to thank the participants in our experiment, and the Norwegian Research Center for Offshore Wind Technology (NOWITECH) for funding this research.

References

[1] EWEA, "Wind in our Sails," Tech. Rep., 2011.

[2] G. Caprari, A. Breitenmoser, W. Fischer, C. Hürzeler, F. Tâche, R. Siegwart, P. Schoeneich, F. Rochat, F. Mondada, and R. Moser, "Highly compact robots for inspection of power plants," in *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*. IEEE, 2010.

[3] N. Elkmann and T. Felsch, "Robot for rotor blade inspection," *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*, 2010.

[4] J. Ribrant and L. Bertling, "Survey of failures in wind power systems with focus on Swedish wind power plants during 1997-2005," *Energy Conversion, IEEE Transactions on*, vol. 22, no. 1, pp. 167–173, 2007.

[5] ISO, "ISO 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Part 11: Guidance on Usability," Tech. Rep. November, 1998.

[6] J. A. Adams, "Critical considerations for human-robot interface development," in *Proceedings of 2002 AAAI Fall Symposium*, 2002, pp. 1–8.

[7] C. W. Nielsen and D. J. Bruemmer, "Hiding the system from the user: Moving from complex mental models to elegant metaphors," in *Robot and Human interactive Communication, The 16th IEEE International Symposium on*, 2007, pp. 756–761.

[8] D. J. Bruemmer, C. W. Nielsen, and D. I. Gertman, "How training and experience affect the benefits of autonomy in a dirty-bomb experiment," in *Human-Robot Interaction (HRI), 3rd ACM/IEEE International Conference on*. New York, USA: ACM, 2008, pp. 161–168.

[9] C. W. Nielsen and M. A. Goodrich, "Comparing the usefulness of video and map information in navigation tasks," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM, 2006, pp. 95–101.

[10] B. Ricks, C. C. W. Nielsen, and M. A. M. Goodrich, "Ecological displays for robot interaction: A new perspective," in *Intelligent Robots and Systems, 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2855–2860.

[11] M. Baker, R. Casey, B. Keyes, and H. A. Yanco, "Improved interfaces for human-robot interaction in urban search and rescue," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 3, 2004, pp. 0–5.

[12] B. Keyes, R. Casey, H. A. Yanco, B. A. Maxwell, and Y. Georgiev, "Camera Placement and Multi-Camera Fusion for Remote Robot Operation," *Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics*, pp. 22–24, 2006.

[13] "Beaglebone Website." [Online]. Available: http://beagleboard.org/bone

[14] Ø. Netland and A. Skavhaug, "Software Module Real-Time Target: Improving Development of Embedded Control System by Including Simulink Generated Code into Existing Code," in *Software Engineering and Advanced Applications, 39th Euromicro Conference on*, 2013.

[15] ——, "Prototyping and Evaluation of a Telerobot for Remote Inspection of Offshore Wind Farms," in *Applied Robotics for the Power Industry, 2012 2nd International Conference on*, 2012.

[16] O. Shechtman, S. Classen, K. Awadzi, and W. Mann, "Comparison of driving errors between on-the-road and simulated driving assessment: A validation study," *Traffic injury prevention*, vol. 10, no. 4, pp. 379–85, Aug. 2009.

[17] Ø. Netland and A. Skavhaug, "Two Pilot Experiments on the Feasibility of Telerobotic Inspection of Offshore Wind Turbines," in *Embedded Computing (MECO), 2nd Mediterranean Conference on*, 2013.

[18] M. Beaudouin-Lafon and W. Mackay, "Prototyping tools and techniques," *Human-Computer Interaction: Development Process*, p. 121, 2009.

[19] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research," *Advances in Psychology*, vol. 52, pp. 139–183, 1988.

Paper F Evaluation of Remote Inspection of Offshore Wind Turbines with a Tablet Controlled Telerobot

Submitted to *IEEE Transaction on Human-Machine Systems*

Paper G A Review of Experiments Evaluating the Usability of Mobile Telerobots

Submitted to *IEEE Transaction on Human-Machine Systems*