NTNU – Trondheim
Norwegian University of
Science and Technology

Development of a Predictive Display Interface to assist control of a Robot Arm in a Telepresence System

Eivind Berntsen

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

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU

Department of Engineering Cybernetics

Development of a Predictive display interface to assist control of a robot arm in a telepresence system

Author:
Eivind Berntsen
eivinbe@stud.ntnu.no

Supervisor:
Assoc. Prof. Amund Skavhaug
Amund.Skavhaug@itk.ntnu.no

June 4, 2012

Dedication

I'd like dedicate this report to my grandmother whom I loved and have many fond memories of.

Abstract

Operation and maintenance O&M is an expensive but necessary part of almost any industrial endeavor. This becomes even more expensive when the object in need of O&M is located in remote places. This will be the case if you have a wind farm at sea. To get the cost of this down research into using a robot or possibly a robotic arm to do remote O&M has been initiated. This Thesis deals with some of the challenges when it comes to remote presence or telepresence and the use of a robotic arm.

Telepresence systems have some unique challenges when it comes to control. The time delay and limits when it comes to field of view has led many to advocate a solution called a Predictive display.

Predictive displays lets the user see an immediate response to control input. It does this by simulating the system that is being controlled and then in some way synthesize an image that would show a simulated response.

This Thesis is concerned with the development of such a display. The solution proposed here leads to the use of a real time simulator of a robotic arm. In addition to this I have worked with a possible solution for synthesizing the view of the robot by using a game engine to render the robot and its environment.

The resulting application as far as the predictive display is concerned leads me to believe that the tools I have used can be used for the purposes described in this thesis.

The predictive display method seams like the best alternative when it comes to providing the operator with information about the robot's surroundings. And thus increasing the operator's situational awareness.

The report shows how the application can be created and provides you with information on how the solution can be extended.

Further, the application created should make it possible to carry out further experiments and help in future developments within this project.

Sammendrag

Drift og vedlikehold O&M er en dyr men nødvendig del av nesten alle industrielle foretak. Dette blir enda dyrere når objektet som som trenger O&M ligger langt unna. Dette vil være tilfelle hvis du skal drifte en vind farm som ligger på havet. For å se på mulighetene for få kostnadene ned når det kommer til O&M forskning på muligheten for å bruke en robot til dette er blitt satt i gang. Denne master oppgaven tar for seg noen av utfordringene et system bestående av en fjern styrt robot arm med vil ha.

Telepresence systemer har noen unike utfordringer når det kommer til kontroll. Tids forsinkelser og begrensninger når det kommer til et begrenset synsfelt har ført til at mange har foreslått en løsning som kalles "Predictive display".

Det prediktive displayet lar brukeren se en umiddelbar respons på kontroll input. Det gjør dette ved å simulere systemet som kontrolleres i sanntid og så på en eller annen måte lage et bilde som vises umiddelbart til brukeren.

Denne master oppgaven handler om å konstruere et slikt system. Løsningen som er foreslått her fører til bruk av en sanntids simulering av en robot arm. I tillegg til dette har jeg jobbet med en løsning for å syntetisere et bilde av roboten og miljøet den beveger seg i.

Den resulterende applikasjonen når det kommer til det prediktive displayet leder meg til å mene at de verktøyene jeg har brukt akn brukes til å lage det ønskede resultatet.

Metoden med det prediktive displayet er den beste løsningen når det kommer til å gi operatøren av systemet informasjon om miljøet han styrer roboten i. Og for å øke hans situasjons forståelse.

Videre bør applikasjonen som er laget kunne brukes til å utføre eksperimenter og tester og det bør kunne hjelpe med videre utvikling innen dette prosjektet.

Acknowledgments

I'd like to first and foremost thank my supervisor Amund Skavhaug for both his guidance and the freedom I have had in this project. It is good to finally set the agenda for what needs to be done and take things in my own phase.

I must also thank Øyvind Netland the Ph.d. student working on this project, for his input and discussions.

I'd also like to thank Hung Bui who are working on the same project and whom I share offices with.

I must also direct a huge thanks to my family especially my sister Kjersti and her husband Jan Petter Morten who have helped me with some of the editorial work on the report. And also my nieces and nephews who provide great inspiration and diversions.

And finally thanks Mum!

Contents

Chapter 1

Introduction

As the threats of global warming and depletion of our oil and gas resources are looming, the search for alternative energy resources are becoming more and more important. One alternative is harnessing the powers of the wind with wind turbines.

Wind farms require large areas with good wind conditions to be effective. The sea has many large areas where wind conditions are good. Exploring the possibilities this resource has, is something that lately has gained a lot of interest.

There are however many challenges when it comes to building and maintaining wind turbines at sea, one challenge is operations & maintenance (O&M) which according to estimates will account for around 20-25% of the total income [24]. In the preproject [6] leading up to this master thesis, I looked into the possibilities remote presence coupled with a robotic arm has at helping with O&M in offshore wind farms, and some of the challenges involved in this solution. This master thesis is in many ways a continuation of this work and an effort to develop some of the ideas that I came up with during the proses of writing the previous report.

The work done has given me a good understanding of what is needed, both in therms of skills and the time needed should one decide to pursue the solutions I have chosen to work on.

1.1 Scope of work

In my pre-project I investigated some of the challenges that would be involved with using a robotic arm in an environment such as the nacelle[1]. There are mainly four things that need to be addressed before an arm is constructed. First one need to define what tasks the arm should perform. Secondly find a way to make certain that the arm can be collision free both with itself, and the more challenging task of

[1]Nacelle is the name of the part of a wind turbine where the gears, generator, etc., is located.

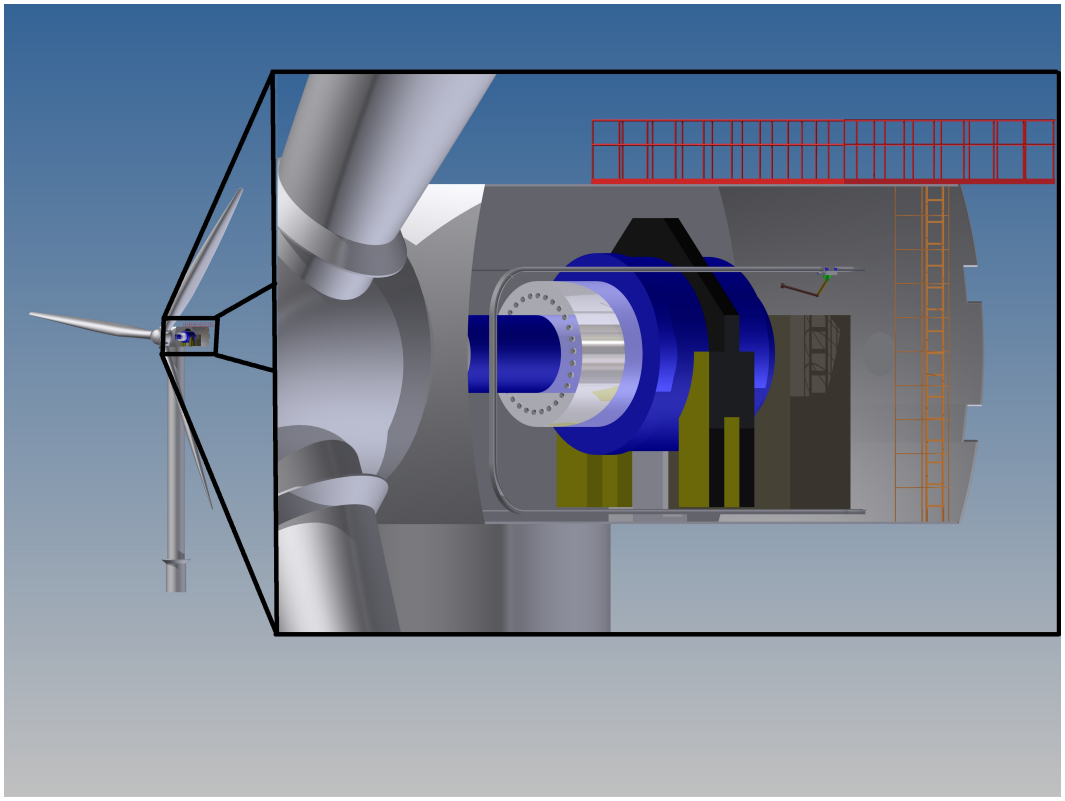

Figure 1.1: Rendering from Autodesk Inventor to illustrate the system

guaranteeing no collisions with it's environment. The third thing to address is how large the arm needs to be and how many links are needed and what configuration they should have. And lastly we need to address how the operators should control the arm and what support the operators need trough the control interface.

All of these topics have been touched upon in my previous work. The question of what tasks has to be performed are hard to answer fully. I think the correct approach here would be to not go to far when it comes to attaching different tools and to start with restrict this to a simple setup.

In addition it is difficult to say anything as to what size and form the arm should have, without knowing more of the environment the robot will be operating in. This question is also strongly linked with what task the arm should perform.

The two questions that can more easily be answered are how to keep the arm collision free and how the user interface for the arm should be. However this report deals mainly with the last question. The aim is to investigate some of the research on this subject and to start the work on a solution for this problem, based on this study. And hopefully make a good contribution to the project.

The report will go into some detail on something that in the terminology of the

telepresence community has been coined as “Predictive display”. A term that will be explained later in the report. We will see that the solution that I have been working on fits rather neatly into this category of solutions.

1.1.1 Outline of chapters

- Chapter two gives an overview on operation and maintenance and also past contributions to the project.
- In Chapter three I discus some of the changes in system architecture. This is however not the final word on the matter as this will be discussed further in chapter nine.
- Telepresence systems have many unique challenges and there has been done many studies in this field, and how it effects robotic systems and control. This is the subject of chapter four.
- In chapter five I present the predictive display as a solution to many of the challenges that face us in a telepresence system. I also discus some of the possible ways in which a user interface can be constructed.
- To be able to create the predictive display we need a dynamic model of a robotic arm this is the subject of chapter six.
- In chapter seven I look at possible ways of creating a real time simulation of the arm with Simulink.
- 3D models and a system for rendering them is discussed in chapter eight.
- Chapter nine deals with how all of these technologies can be put together and used.
- In the tenth chapter We take a look at how some of the difernt parts can be combined in to one system.
- Chapter eleven is the final chapter and concludes the report and deals with what should be done next.

1.2 The work flow in this project

I had the ideas for what to do in this project while working on the pre-project. The work done here has involved CAD modeling, visualization, Human computer interfaces, real time simulation, and to some extent hardware considerations. In

addition it has been necessary to look in to some literature to see what has been done and what can be done. And what methods yield good results.

When planing this project I realized that the first thing to do was to find out If I would be able to use Panda3D [30] for the things I had planed to do with it. I had looked in to the possibility of using it in the last semester.

I started out reading parts of the manual for Panda3D. I wanted to get to the point where I was able to use it to render the same 3D models in two display regions of the app. When I had manged to do this with some of the models shipped with the Panda3D SDK, I felt ready to move on to finding a way to take 3D models created with solid modeling CAD tools in to Panda3D.

Once I reached this milestone it was time to look in to modeling a robot. I did come quite far with this however during this work I discovered that somebody had created a lot of what I would need and published the work done.

Using this model I then proceeded with exploring the possibility of creating a real time simulator of the robot arm.

It has also been a goal to provide good documentation of the methods i have used. This is part of what this report is about. Hopefully somebody can take what I have created and continue the work.

Chapter 2

Background and motivation

2.1 Maintenance

Maintenance is the task of making certain that a system is operational. In some cases you are not allowed to have any down time, this makes for both an expensive system and calls for hard demands on the maintenance routines employed. In our case the constraints are not so strict, as some downtime can be allowed. The driving factors here are economical. From an economical point of view you don't want to do more maintenance than strictly necessarily to keep the wind turbine operational. Obviously when the wind turbine is not working and there are good wind conditions for producing power, the owners are losing money. To make good economical projections you usually want to plan when the system is down for maintenance. To achieve this one usually employs a maintenance strategy called planned maintenance. This can involve cleaning and refurbishing and a general inspection of different parts. If you or the company producing a specific component, have past data for that component, you can also make predictions on the Mean Time To Failure (MTTF) based upon this. Using this data you can make plans for when you want to replace that component. However in some cases it can be more economical and practical to run a component until the end of its lifetime or until it fails. Such a maintenance strategy is called Corrective maintenance. If there are no data to be used for estimating the component's MTTF, and no measurements can be performed, or such a measurement would be to expensive compared to the potential benefits of doing it. In such cases the corrective maintenance strategy can be the one to choose.

A different and more advanced strategy to determine maintenance schedules and the replacement of parts, is to employ a system to record and monitor different data in the system. These sensors give information about the health of the system or specific critical components of the system. This is called Condition monitoring.

This can potentially allow you to predict more precisely when a component will fail and thus you don't have to replace it prematurely or for that matter allow the component to go beyond its point of failure. Such a system can also be used to prevent a failure to happen. This can be achieved by letting a safety system, or possibly an operator, shut down the system before a component has completely degraded. You can then potentially prevent a cascade of failures to components that would otherwise be damaged.

Condition monitoring allows you to go from scheduled or corrective maintenance to predictive maintenance or condition-based maintenance. Condition monitoring however can't catch every kind of failure, not every thing can be effectively measured and the measuring devices can themselves fail. Condition monitoring systems have been employed in some wind-turbines and some experience has been gained with regard to this [9, 24, 36].

In [24] some of the measurements you can use in condition monitoring systems are outlined and discussed:

- Using accelerometers to measure vibration in the gear, shaft and bearings
- Torque measurement to measure the rotor load
- Oil/Debris Analysis to check the health of bearings
- Temperature of bearings
- Acoustic Emission to check the health of bearing and gear
- Stator Current/Power

For a more comprehensive study into these types of measurements take a look at [24].

Wind turbines are quite complex systems and you can't necessarily look at one measurement and determine whether or not that reading is unusual. You have to view the data in correlation with other data to determine whether or not you are in a safe state.

The condition monitoring systems are usually quite good at finding out that a failure has occurred and that there is a fault in the system. Indeed some times it can detect that a failure is about to happen. However it does not necessarily tell you precisely what has happened and specifically what component has failed.

2.1.1 Inspection

Inspection is an important part of maintenance. It can allow you to find parts that are about to fail before they do. Also during an inspection you can do minor repairs. This can help you to keep the wind turbine operational longer.

An important part of this is that you can make corrections on components that otherwise could fail and cause bigger problems in terms of cascading damage from one failure.

When it comes to wind turbines one important part of this is blade inspection. This is usually a visual inspection either performed with personnel doing a rappel down along the blade when the turbine is not operational, or by a camera mounted on a remote controlled mini helicopter. However the German Fraunhofer Institute for Factory Operation and Automation has developed a robot (Figure 2.1) that is capable of doing this inspection automatically. The robot is equipped with infrared thermography sensors, ultra sound sensors and a camera. This means it can provide you with more data on the health of the blades [16].

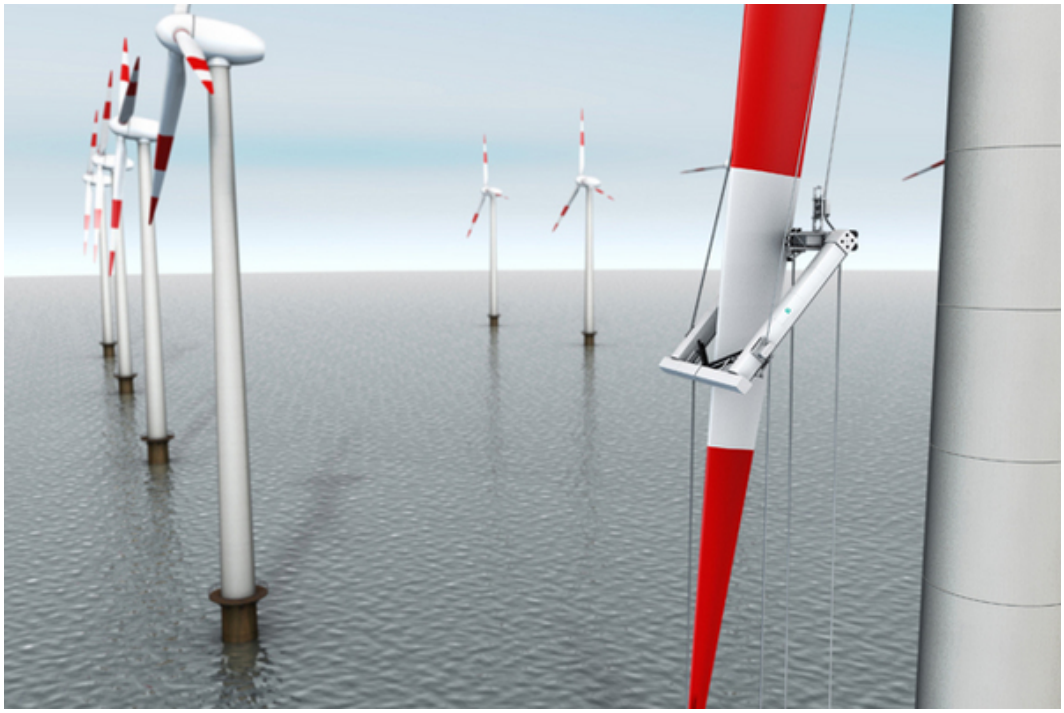

Figure 2.1: The blade inspection robot

There are also rail mounted robots in use for inspection. This is being used in pipes and in nuclear reactors.

It should be clear that a lot of money is being spent at keeping these investments running, and so any savings either from heightening reliability and up time or cheaper O&M should be welcome. It is in these areas we believe our system can be of benefit.

2.2 Previous work

To monitor or do inspection inside a nacelle, previous work in this project has argued that a system consisting of a rail and one or more carts is the best option. Indeed if you want to extend the system capabilities to that of maintenance this design decision makes even more sense, as this system should easily allow you to bring the tools you need to where they are needed with out to much fuss.

2.2.1 Rails

Figure 2.2: Rails with carts mounted

A set of rails and carts where developed by Viktor Fidje and the details of the design can be found in [17]. As can be seen in Figure 2.2 the rails consist of two aluminum tubes running in parallel, connected by C shaped crossbeams. The design is relatively inexpensive and the tools used for bending equally so. However,

the bending process used has deformed the cross sectional shape of the pipes so that they are no longer circular, but elliptical. This and other factors mean that the wheels of the carts don't have good surface contact. It also seems somewhat difficult to design a good wheel configuration to accommodate this design. In particular with respect to the demand for longevity of the cart and rails it seems hard to make a good suspension system for the rail.

When Torgeir Welo a professor in mechanical engineering was asked to design a rail profile for this kind of system he came up with a wholly different design using a monorail. This design looks more promising as far as stability goes, and the forces working on the axles of the wheels will be more or less parallel to the axis of the wheels.

The design that has been implemented does however easily beat the monorail design when it comes to ease of construction and the limits posed by the budget. It is also a good start as far as prototyping goes, and it may suffice to test new ideas. However not everything built for this rail can be directly ported to the final system.

2.2.2 Carts

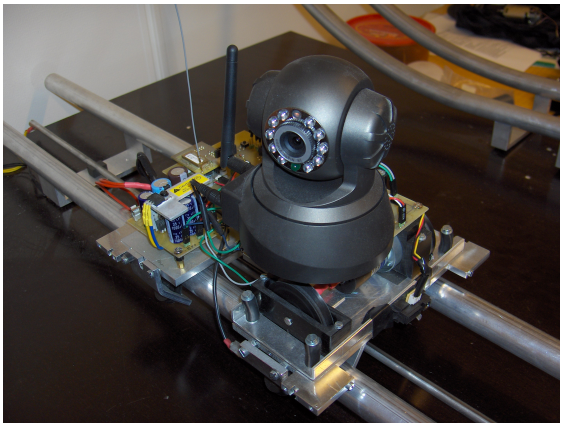

Figure 2.3: First cart

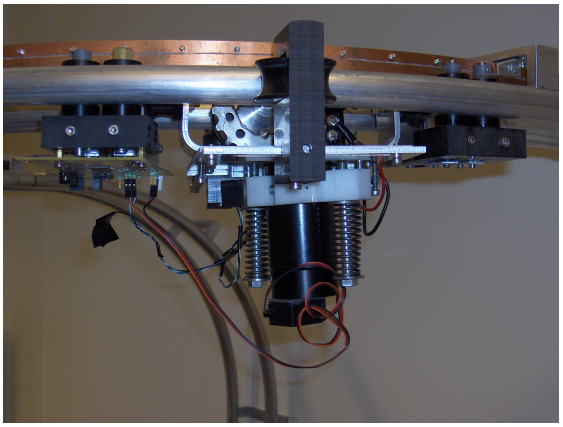

Figure 2.4: Second cart

The first cart created (Figure 2.3) is the one that has the camera installed. It consists of two parts linked together by a joint that only allows for movement in the horizontal plane. This means that it can't ascend or descend along the vertical bends of the rail. To move this cart along the rails the large wheel on the topside has been motorized. It was however found that this did not provide enough friction to move the cart when there was a small inclination of the rails relative to the horizontal plane. This is true even though the wheel has been fitted

with a rubber O-ring to offer better traction on the rail. This also has the effect of making it impossible for the cart to handle the bends in the rails, as the link between the two sections is quite stiff and the wheel gets less friction in the bends of the rails. It is thus limited to a straight rail with only small inclinations.

For these reasons a second cart has been developed (Figure 2.4). This cart has a cogwheel that interacts with the bicycle chain to provide locomotion for the cart. The cart consists of three sections, two that provides stability and some room for electronics and potentially camera and sensors, and one in the middle that provides a platform for the motor. The sections are connected with a flexible joint that provides freedom to move in any desired direction along the track. This has however not been tested as the rails are not complete yet.

A big problem with the second design is that it vibrates quite heavily while moving. This is due to the interaction between the cogwheel and the bicycle chain. This makes it unsuited for our purposes as the vibrations are likely to reduce the lifetime of the whole system. The vibrations will probably also propagate to the camera and/or the arm. This can be detrimental to the quality of the visual feed to the operator and control of the arm. These problems has been considered too difficult to tackle when it comes to testing how a remote presence system might work in the an inspection situation.

The cart and rail system developed by the professor can be seen in Figure 2.5. This system has obvious advantages over the previously built systems when it comes to providing a stable platform for a robotic arm. All the wheel axis are parallel to the surface the wheels are in contact with. The drive system proposed will be made from plastic. Which should prevent large vibrations.

2.3 Last semester

The last semester there were four people working on the project. Me, Hung Bui, Jeremias Moragues, and Øyvind Netland.

Hung Bui was working on the electronics on the robot. He was investigating the use of a more powerful micro computer, the Pandaboard [31], that could be used with the robot. The new system does not need a dedicated server inside the wind turbine. The previous robots used Wi-Fi to communicate with the camera where as commands to move the robot were sent over a separate RF connection. This is not necessary with the Pandaboard.

Jeremias Moragues was working with using a Joystick to control the cart. He also did some work on the possibility of using a model train in a test of the principal idea of the system. In addition he was looking at the anatomy of different nacelles and what maintenance tasks are being performed.

Øyvind Netland was gathering information doing duty work and taking a course

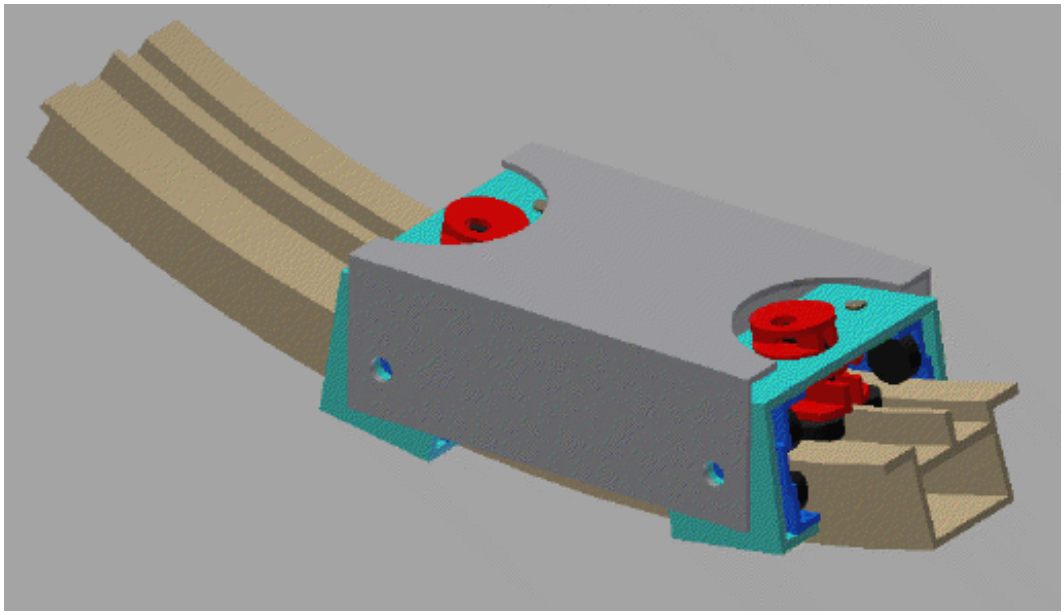

Figure 2.5: Basic Rail and cart design

on human machine interfaces, all needed for his work on his PhD. He was also assisting us, and answering questions we might have.

2.4 My previous work on this project

In my pre-project I deal largely with how a solution with a robotic arm, will fit into the system. It further deals with the potential benefits of using an arm, or indeed the necessity of such a solution. I also look into the challenges one will encounter with extending the system with an arm.

The main arguments for the arm is that it will extend the capabilities of the system. If you attach a camera and other sensors, that are dependent upon the angle and relative placement in its environment to give good data, the system can give an inspector more information to base his or her conclusions upon. Indeed there may be viewpoints that you can not reach remotely in any other way. That is without sending people to the wind turbine. Something you want to avoider as this is more expensive.

If it is later found that a system designed for inspection is too limited and that you need more tools to perform different tasks, then you will have a good platform to stand on to further extend the system. This should be apparent as having designed the first system and made controllers and a user interface for this you have a lot of the tools ready. Also the project will have an accumulated experience

base to draw upon to make the development time shorter.

In the report I conclude that constructing the arm might not be the right thing to do at the moment. It may be more appropriate to handle the problems inherent in the fact that the system will be a telepresence system, with a potentially difficult control task for the operator. This led me to propose a solution where the operator is given a virtual 3D representation where the input from the operator is displayed immediately using a real time simulation. This I discovered this year can be called a predictive display. The main body of work has gone into this.

2.5 Current work on the project

This year a second prototype cart has been developed by Netland. This cart has got the Pandaboard on it. And it can go along the whole rail. This should make it possible to do tests and verify the soundness of the design.

Hung Bui has continued work with the Pandaboard but, the work has focused on producing a robot for telecommunication. Although this work has not been performed directly with use in the same context some of the work done by him might be of interest.

Chapter 3

The system

As has been shown the system now consists of a cart with a camera, that can pan and tilt and moves along a rail. The cart previously has communicated with a server that is connected to a network inside the nacelle. This server then communicates with the operator station. Lately, work has gone into removing the server. Instead an embedded computer on the robot simply communicates over a wireless network with a router. This router only has to relay the messages to the operator station. The main blocks of the system can be seen in figure 3.1

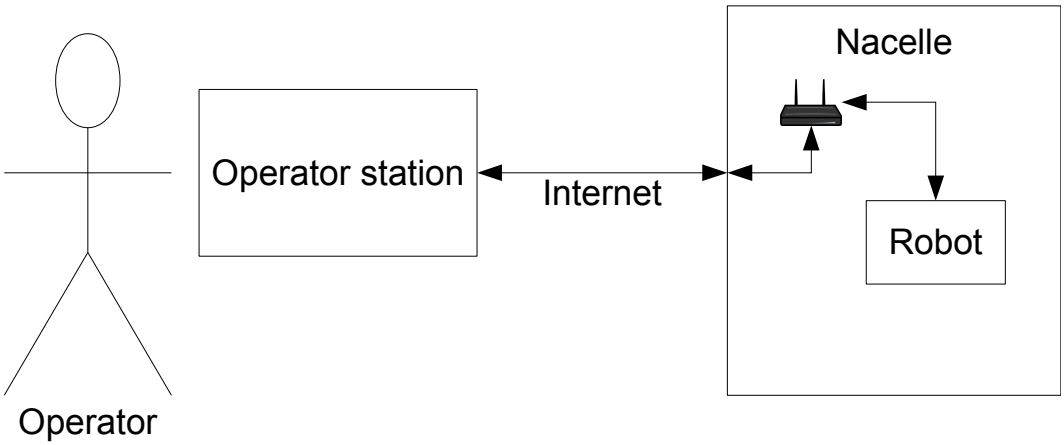

Figure 3.1: Principal diagram of the system.

In the master thesis of Thor Mælum Karlsen [20] the use of a robotic arm was discussed as an addition to the system. This can potentially lead to significant increases of the capabilities of the system. Not only when it comes to what you will be able to see with it, if you attach a camera to the end of it, but also the possibilities to attach appropriate tools and sensors for measuring vibration, sound

and temperature etc. This has been described in more detail in my project work [6].

The system does not have to change with respect to what is shown in 3.1 when we introduce an arm to the system. Naturally there will be some changes in the electronics of the robot, although it is probably not necessary to use a different embedded computer than what has been suggested by Hung Bui in his project work [8], however you would have to create electronics to interface the card with the signals coming from the servos in the arm and other sensors, as well as an interface to the power electronics.

There will also be some new requirements to the operator station with the solution proposed in this report. The virtual 3D environment will most likely consist of many geometric shapes concentrated in one place and as such you will need a powerful graphics card.

The simulations of the arm will as we will see later require a lot of computations. If you want to run both the simulations and the rendering of the virtual world on the same computer this will most likely require a multi core CPU. The simulations should run on a separate core. While the game engine runs on one core, it might also be desirable to run a collision avoidance algorithm on a separate core.

The current version of the rendering software proposed here supports only running on one core, The next release of the software used for rendering, supports running on multiple CPU's through a scheme called pipe lining. If this is utilized it may become harder to schedule the system to run well. These things will be discussed further after having looked at the game engine and the chapter concerning the simulation of the arm.

Chapter 4

Previous findings on telepresence

As mentioned, in the last semester I did a prestudy into the applicability of using a robotic arm in an O&M telepresence system. operating inside a nacelle [6].

In that report I argue that to inspect or perform maintenance in side the nacelle it might be necessary to employ a robotic arm to get to the spots inside the nacelle that are of interest. However I also outline some of the challenges we face in employing such a system.

There are many challenges inherent in a telepresence system. These challenges I discussed in the previous report but they are so fundamental to the continued work on this project that they are wort restating and discussed here.

4.1 Telepresence

Telepresence leads to many challenges when it comes to controlling robots and it is an expanding research field. In [14] telepresence is defined "as the perception of being present in a remote environment". The quality of this sensation is often measured subjectively trough questionnaires such as the one presented in [37].

Telepresence systems are employed in many areas and industries. It is an important subject when it comes to robot control in areas where humans can't go, such as in areas of high nuclear radiation (nuclear reactors, dump sites) [26]. It is also an important part of some sub sea exploration, and in sub sea mine clearing. It also enters in to space exploration and combat drones.

An important concept in telepresence is that of situational awareness. This term is a qualitative measure of how well the teleoperator is able to understand how the robot is situated in its remote environment, and what consequences actions performed will have towards get him closer to realizing his goals. Most research papers on telepresence in some way address this subject.

The article that I have found to best describe the challenges and solutions that

has been employed in the field of telepresence is a summary of more than 150 articles on the performance issues involved in telepresence and interface design in robotics up until the year 2007 [11]. The article systematically goes trough the limitations and problems encountered in telepresence systems and some of the solutions to counter these.

4.1.1 Limited field of view

The article starts with the problem of a Limited field of view also called the "keyhole" effect. The essence of this problem is that the operator is unable to perceive the peripheral surroundings of the robot and must rely upon his mental model for the surroundings to navigate that space. Cameras don't provide the operator with the same degree of peripheral vision as you get with the human eye. You can extend the cameras field of view and also employ many different cameras to give more information. This doesn't entirely counter this effect though. Unlike the human eye, the cameras can seldom change its viewpoint as fast, and with the same precession. Also the peripheral vision in humans is perhaps more than a matter of just field of view, as research strongly suggest that the visual impressions we get from our peripheral vision is processed differently by our brain than the central field of view [2].

The limited field of view also means you will have to create a mental model of where the robot you are trying to control is in its environment. This is less of a problem when it comes to a robot you can observe in its environment. This problem has been observed many times in rescue robots where the operators are unable to navigate in the ruins of the buildings they are searching trough. They often get stuck without knowing exactly why [10].

4.1.2 Problems with multiple cameras

A problem has also been found when it comes to using multiple cameras and screens, that is that the operator start to change his focus between them and instead of this helping the operator it leads to confusion. The article suggests that you can in some cases use auditory alerts and visual momentum[1] techniques [38].

4.1.3 Depth perception

Good depth perception is something that a camera has problems with giving the operator. This makes it harder for the operator to estimate distances.

[1]Visual momentum is a qualitative measure of a users ability to extract relevant information across displays or windows/widgets

When humans navigate in a certain space they can rely upon stereo vision to help in judging distances. This is however not the only means with which we make these estimations, we also use the size of reference objects, and are helped by the parallax effect[2].

Some solutions to this involves using stereo vision to give operators with better depth perception in the remote environment. This may not be an appropriate solution in this case though, as you would have to spend a lot of time developing touch a system. And it might end up being quite expensive.

4.1.4 Frame rate

Low frame rate can lead to a degraded sense of motion for the operator. This in turn leads to the operator using a drive then stop strategy for navigating the remote environment. In the article [11] several articles are cited as to the minimal frame rate for maintaining the sense of motion, the article concludes that this minimum lies around 10 Hz.

4.1.5 Time delay

In general time delay is detrimental for good control of a robot. It appears that for different applications and environments, the maximum allowable time delay, before the operator has problems dealing with this, waries significantly. In some cases one second is cited as critical for the operators navigation strategy. In other articles, they can see significant performance degradation at lower latencies, ranging from one second to somewhere in the order of 100 ms.

Something that I had not found in any of the articles I studied in the pre-project, is that a varying delay such that you often have in communicating over the Internet, can degrade the performance of the operators even further. Indeed, studies have shown that a small but time varying delay can lead to worse operator performance than a larger but stable delay.

One solution to the variable time delay could be to buffer up parts of the video feed so that you are able to give a continuous feed to the operator. However, such a solution would led to a rather large delay.

One solution I thought of while doing the prestudy was to show the operator a simulation in a virtual 3D environment. In the article [11], this is referred to as Predictive display. This solution is one of the the subjects in the following chapter.

[2]The parallax effect in this case refers to how objects that are close to you will appear to shift more compared to the objects that are far away as you change position

Chapter 5

User interface

Designing a user interface or operator interface embodies more than just how the layout of the system should be in this case, but also the use of the predictive display technique to assist the operator.

A well designed user interface will help immensely when it comes to how well the system as a whole will be rated by the ones who use the system. Having solid hardware is of little value if the human machine interface is under par for the tasks the operator need to perform.

5.1 Defining the interaction between operator and the system

To some extent, I will concentrate this discussion on the task of inspection, and using this system for that purpose.

In doing an inspection you are often working with a checklist. This means that the inspector can, if the list is well constructed, simply move from one area of the turbine to the next and systematically go trough the checklist.

Øyvind Netland, a PhD student working on this project, has suggested that this can be done easily by simply letting the robot run a predetermined program, where the path the robot follow let the inspector simply observe what the camera can show. The robot can also be made to then stop at certain points of interest, and wait for confirmation from the operator to make certain that he has observed the things that are necessary for him to complete the inspection satisfactorily.

Such a solution would not require much in the form of collision avoidance. The interface during such an operation would probably not have to be to complicated either. The most important feature would be to show the video feed. It might also be useful to give some data to the operator about the things that are in the view of the camera, and possibly also a map. This can be done in different ways.

Potentially you could have a separate area of the screen that displays this data. This would however require that the operator shifts his attention from the video to the data display area.

One problem with using different areas of the screen for different view points is that this can lead to the same type of confusion as using multiple cameras as described in the previous chapter. The operator has to use some time to change his attention from one area of the screen to a different area.

Netland has mentioned that it might be desirable in the future to extend the system with image recognition to help the operator during his use of the system. Netland wants to make it possible for the operator to click on certain objects observable in the video feed and then let that initiate an action to show data on that object. This type of interaction would make it possible to display the data on top of the video feed without the operator having to shift his attention from the video feed. Also this would allow the video feed to occupy a larger area of the screen.

To always have the robot follow a predetermined path might not be the best solution. If the Inspector discovers something he considers irregular he might want to investigate this further either with a closer look or viewing it from a different angle. It might also be the case that a sensor has given an irregular reading that must be investigated. In such cases you must employ a different strategy to get the robot into the right spot at the right time.

There are two control strategies that might be applicable in such a situation. You could let the operator select on a 3D map of the nacelle the position and angle he wants the camera to be in next. Then the system can calculate a path, if one exists, that would allow the robot to get to that position without colliding. This would make it relatively simple from the operator to control the robot. This strategy does however seem slightly more difficult to utilize if the operator only wants to make a small change to the position and the operator is not entirely certain as to how far that would be, to get the view you want. It might then be better to allow the operator manual control of the robot.

This should make the system more flexible, and you don't have to do much pre programing of movements for this to work. In addition it seems easier to extend the system later should you want an arm to perform more advanced tasks.

While the first solution will require a good algorithm and interface to be possible to use successfully, the second solution on the other hand inherits more of the problems discussed when it comes to making a good telepresence system. If the operator is to navigate the arm using only the video feed he will experience the effect of delay, limited field of view, and limited depth perception.

The second alternative seems the easiest one to implement despite the challenges involved in telepresence. It allows the operator to move the arm relatively

freely, and it seams like the easiest way to inspect the nacelle. But a question that I asked myself while I was doing the pre-project was, how do we handle all the difficulties with telepresence. The answer I came up with was to present the operator with a view of a virtual arm that simulates the movement that would be performed by the real arm.

5.2 Predictive display

The predictive display works by providing a virtual representation of the robot under question and its environment. The kinematics of the robot is then simulated in real time in such a way that the operator is able to instantly see the effect of input to control the robot.

The pioneering work when it comes to predictive graphic displays was according to [22] made by Noyes and Sheridan in an article [29] published in 1984. According to the same article there had been very little progress in the field of telepresence since the 50s. Predictive displays were later in use in a remote controlled space arm in the year 1990 [5].

In [11] using a predictive display is strongly suggested as a solution to avert the problems of time delay; "Time delays as short as 170 ms affected driving performance. If these delays cannot be engineered out of the system, it is suggested that predictive displays or other decision support be provided to the operator". However the predictive display can potentially do more than that. As you are operating in a virtual world you are not limited to only providing the viewpoint of the camera but you can show the whole arm. Thus the operator does not have to maintain a mental model of the arm configuration and the environment behind the camera of the arm.

In a situation where you have manual control of the robot, you can be fairly certain that time delays can have a significant impact on the human in loop control task. The predictive display technique should be able to counteract the problems of time delay.

In [12] they concurrently build a coarse 3D model of the environment the robot is to operate in, and then use this model to visualize the effect of input from a Phantom omni controller, figure 5.1. This input is also transmitted to the real robot. They find that the use of the 3D model with real time simulation significantly enhances the performance of the operator in the loop. Indeed the performances are not far from that of using the controller in real time with the video signal. They report that the use of the predictive display improves performance time with 40% over that of the video feed with delay.

Predictive displays have also been used for other purposes. In [35] they set up a navigation task in two dimensions (figure 5.2) where participants of the study

Figure 5.1: Phantom OMNI controller [32]

navigated a boat using a predictive motion display. The idea here is to control the acceleration of the boat, not the speed. They then use simulation to show a predicted position of the object at some predetermined time ahead with the current acceleration. This has also been proven to be an effective way to use predictive displays.

There are also many articles in which they present predictive displays where they use images from the camera to create new images to use as the predicted response to input from the user [12, 19]. There has also been predictive displays using laser range finders and stereo vision apparatus to create the virtual environment for the predictive displays. The Kinect [23] might also be a candidate for such applications.

5.3 Developing a predictive display

In our case there are many options available to create a virtual model of the environment. All of the methods mentioned above are strictly speaking possible. One that has not been mentioned so far is also available, that is using cad models of the environment as a foundation for the virtual world representation.

A nacelle and the components within will most likely all have a 3D cad model

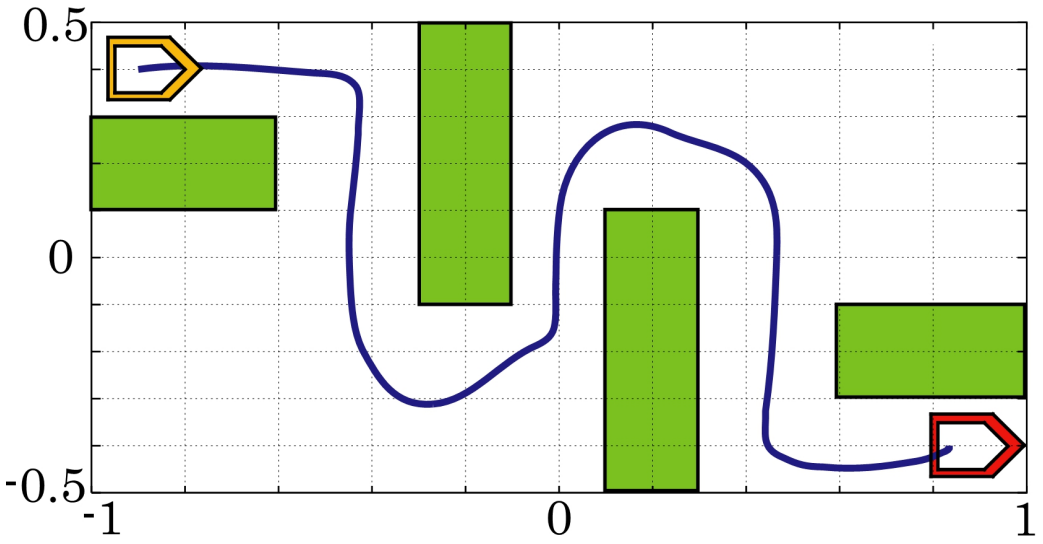

Figure 5.2: Test system used in [35] to test predictive display

available as a foundation for construction. Many subcontractors delivering components to the industry will provide simplified models that show the outer dimensions of components. these can be put into your model, so that you are able to make certain that they fit with the rest of your components. In cases where no such model is given, the company making the construction will often make a mockup of the part.

It should then be possible to use this information to construct the virtual environment needed for safe and efficient navigation of the nacelle. There are some potential drawbacks though with relying upon this. You will have to get the models from the company owning the rights to them, and you, have yourself the right to use these models for the purposes mentioned here. This might become a problem. In the case that you are not granted the right to use all models you should still be able to create and use mockups in their place. However at some point where if you don't get enough cad models this strategy might not become a fruitful strategy.

There might also be a problem with accuracy between the models and the constructed nacelle. In some cases constructors might diverge from the blueprints they are given, if there are no reasons why this should be a problem, and there is a problem with following the blueprint. It then becomes a matter of whether this is later updated and reflected in the blueprint or cad models.

This method does however allow you to spend less time in developing new technologies, you only need to use "old" technology in a new way.

I'm going to concentrate my efforts into outlining and making a system where

the images are renderings of a virtual world and not product of older photos. This seems like the best option as in the article where they use the image based technique to show renderings of a robotic arm, the camera is not situated on the end of the arm but rather at some point where the camera can see the arm. As the images from the camera is used to create the predictive display view this would be suboptimal for our use. If you were to use such a technique you would have to install one or more different cameras to provide this view. It could prove to be hard to place these cameras to give an optimal viewpoint for all actions the operator might want to perform. You could of course also just simply opt not to show the arm. However then you would not be able to counteract the problems described in the last chapter of the limited field of view.

It seems a better option to build a virtual world and show a simulated arm, in what you in the gaming world would call a third person view[1].

5.4 Layout of the user interface

My main Idea when I started out was to first and foremost create a user display that shows a map like view of the robot in the environment, the 3D predictive display, and the video feed. Netland also mentioned that It might be nice to have an area on the screen where you can display data on parts inside the nacelle. The basic layout would be as shown in figure 5.3.

As can be seen the largest part of the screen is dedicated to the video feed, when it comes to doing inspections this is the most important source for visual information. The predictive display part and the map is left a bit to the side and are not given as much space on the screen. The same is true for the data part of the screen.

The intention is that the map is something the operator only glance at when he is uncertain of where the robot is located inside the nacelle. The predictive display in addition to showing a simulation of the arm in real time would also give the operator a view of the whole arm while the video feed shows the video from the camera on the end of the arm.

This display might however, not be the best choice. If the arm is following a predetermined path inside the nacelle it isn't necessary for the operator to see the predictive display. While at times it may be convenient for the operator to see on the map where the robot is located, it isn't necessary for him to see this all the time.

[1] In the gaming world a third person view is one in which you are not seeing the world from the eyes of the protagonist of the story, but rather from some camera angle, usually behind the protagonist. In the case of the robot arm you would not see through the camera but rather as if you were a ghost floating behind it

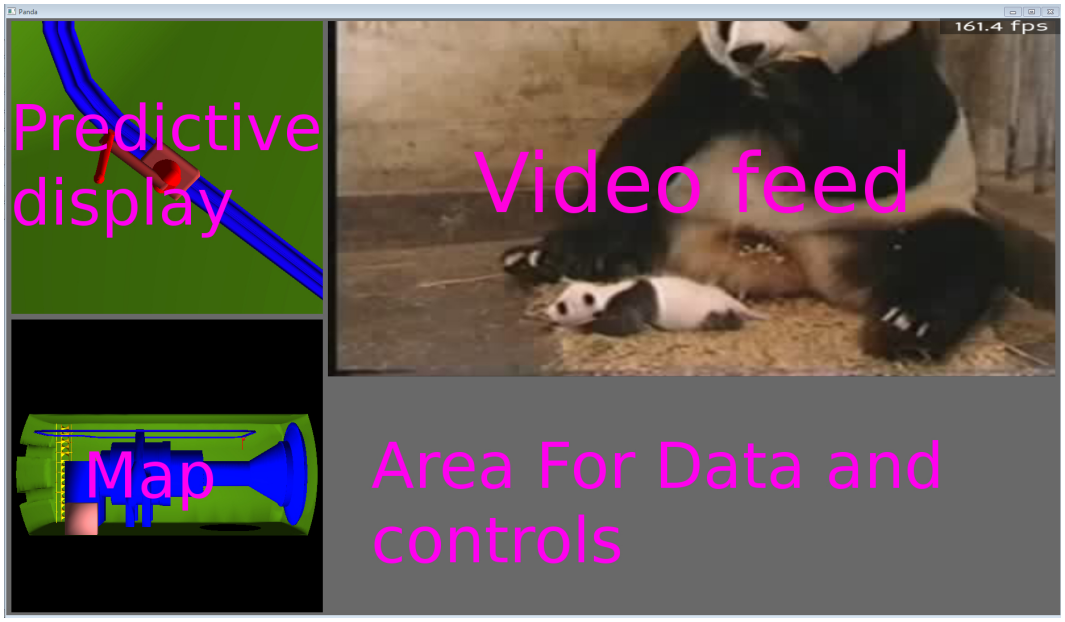

Figure 5.3: Basic Layout of the User Interface

Having the possibility of displaying data and some control options might be nice, but maybe this can be solved in another way perhaps by letting the operator get this as pop up displays on top of the video feed.

It might be best to have two different display modes one when doing the regular "guided" inspection and one mode for when the operator need manual control. At the point where the operator is performing manual control operations.

These things I considered rather late in the project so not much effort has gone into realizing something like this in the software. In any case these things are something Netland will have to consider and possibly something he is able to give better answers to. (As he has taken a course on human machine interfaces.)

In any case I think the predictive display is needed. To create the parts we need for the predictive display we mainly need three things. We need some way of rendering the 3D models, the 3D models themselves, and we need a real time simulator for the kinematics of the robot. These things and how the arm should be are the subjects of the next tree chapters.

Chapter 6

Arm model

There are many ways in which an arm can be constructed and many different configurations that may be useful. Tings to consider are placements of servos how many links it should have. However the questions of the length of the arm and the amount of links needed Is the first question I will concentrate on in this chapter.

6.1 Physical dimensions of the arm

The arm should reach any point of interest in the nacelle however this can be quite hard to do. And without knowing the layout environment, that is how the nacelle is built this is difficult to make any conclusions about.

As mentioned we want to reach every point of interest in the nacelle, or at least enough places that you can within reason lower the frequency of human on site inspection. You might in the future like the arm to be able to do harder maintenance tasks, however in the first place the goal and expectations should probably be lower for the first prototype robot.

You are also constrained when it comes to where you can put the rails. One thing is that it has to be physically possible to place the rails where you want them, another is that they must not severely interfere with other interest such as doing a regular inspection, different maintenance tasks. They should also not severely interfere with changing old gearboxes as these are expected to be changed once in the life of a nacelle.

I did consider some solutions to this problem during the pre-project. However I thought of another this year. To reach as many places in the nacelle you could attach the arm to a beam extending out from the cart and then place the rails more or less in the roof. An illustration of this is shown in figure 6.1.

This solution can be good if what you need to reach also lies in a path that is cleared so that people can walk there and there are no obstructions there. If these

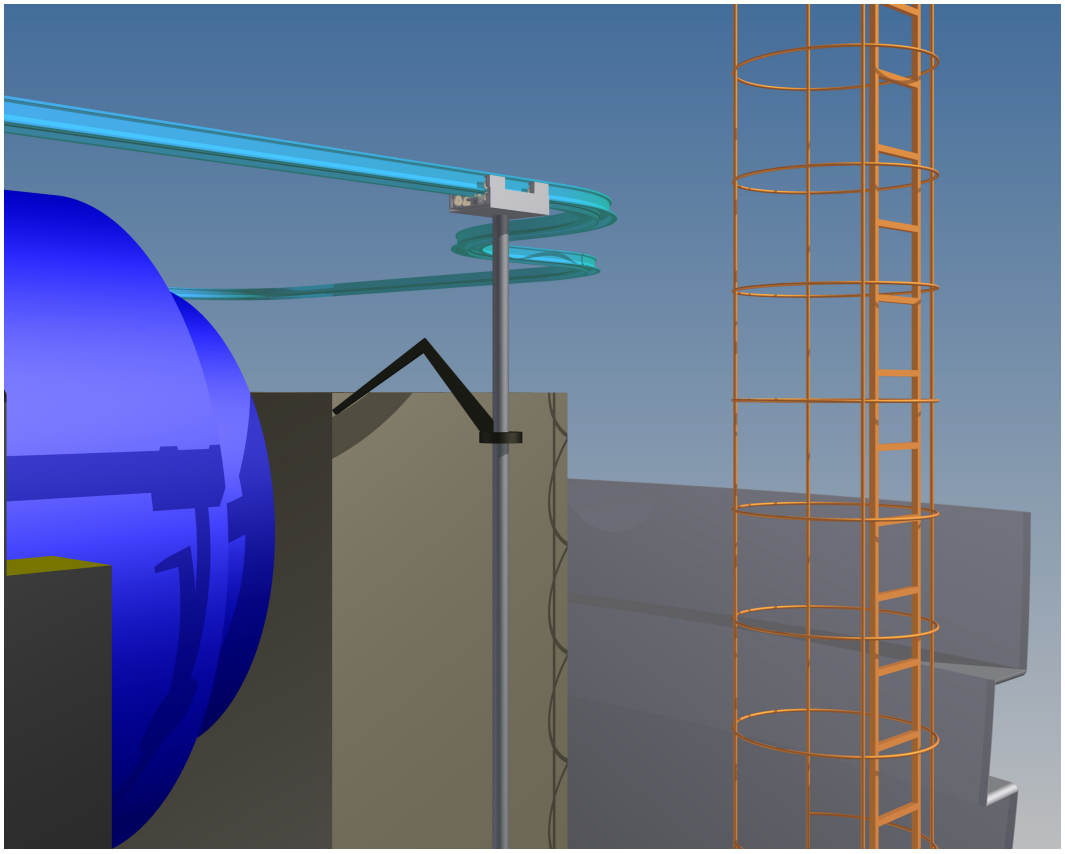

Figure 6.1: Robot attached to beam

conditions are met then this could be a good solution. There is no need to have the cart be able to handle any downward bends as anything below the cart can be reached trough lowering the arm along the beam.

Despite this I will concentrate on a solution that resembles that shown in the illustrating in figure 6.2. The advantage of this design is that it does not require as much clear space where it goes and that it can more easily be led vertically along the rails. The disadvantage is that to cover a larger area is that you will need more rails and that the rails will have to be closer to the points of interest and therefor stand a chance of being in the way.

The robot can be represented in the Denavit-Hartenberg Convention as shown in figure 6.3. It has one degree of freedom in the base allowing the arm to swivel around, one joint giving a second degree of freedom that stands close to the base and one joint that resembles an elbow joint. The end effector has three joints giving the arm a total of 6 degrees of freedom. If you then add the freedom of movement along the tracks you will have to ad more degrees of freedom.

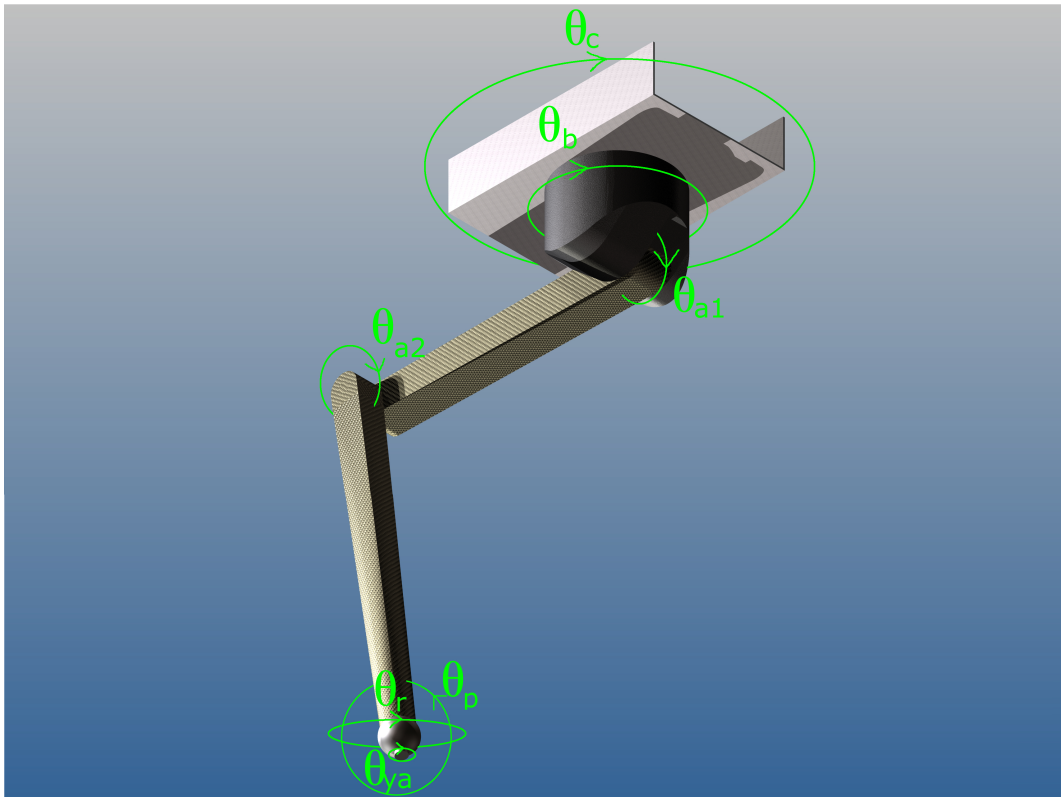

Figure 6.2: Robot used in 3D simulation

If the rails are set up in such a way that they only allow the robot to move in the horizontal plane you will have two degrees of freedom that describes translation and one degree of freedom that describes rotation about the central vertical axis of the cart. If on the other hand the rails are allowed to bend downward you get one more degree of freedom that describes rotation and one more to describe translation. If you then also allow the rails to twist you get one extra degree of freedom that describe the and the twisting of the cart.

In the 3D models I have restricted the rails to move the cart only in the horizontal plane.

6.2 Arm model

To be able to simulate and control the arm we must have a mathematical model of the arm. I initially wanted to develop thees models by myself. However after some time doing this I realized that this was somewhat of a large task. This despite the fact that I was using Maple [27] to help me derive the equations.

I used the method called the Euler-Lagrange formulation as found in [21], to develop the equations of motion. It was my intention to first develop a model with translation in the horizontal plane and one degree of freedom added to the cart to describe rotation caused by the bends in the rails. I also decided not to model the degrees of freedom in the end effector and rather add this later. My efforts has been added in appendix C, and the maple scripts can be found on the DVD. This work is however not complete and for reasons explained later, probably not of much future use.

As mentioned during the work of deriving the equations of motion I realized that It would be a large job to develop the equations I was also not certain of how to go from the equations in maple to something that would run in MATLAB [28]. For this reason I decided to do a quick search through some of the previous master thesis reports having been done at NTNU.

I found that indeed somebody has done this before me. Herman Høyfødt did his master thesis [18] in modeling the dynamics of the IRB 140 robot arm from ABB in the fall of 2011. Where I was attempting this using the Euler-Lagrange formulation Høyfødt used the Newton-Euler formulation.

The IRB 140 is a six degree of freedom robot, not unlike what I had in mind of modeling myself. This model does however not contain translation but the end effector has been fully modeled. This should make it suited to doing the simulations and also the size of the problem is equal to what I had in mind of developing my self.

One thing I hadn't expected to find, was the fact that trough the Newton-Euler formulation they where able to decrease the relative computation time, when using the variable step method Dormand-Prince[1]. This method is according to [15] the standard method used in MATLAB for integrating initial value problems. When compared to using a model derived by the Euler-Lagrange formulation the Newton-Euler formulation yielded far better results in terms of computation time.

A PhD student. S. Pchelkin working at NTNU derived a dynamic model for the robot using the Euler-Lagrange formulation so that the two models could be compared. Høyfødt simulated the two systems in both open loop and with a PD controller. The two systems are simulated in ten seconds using the variable step method in open loop the computation time for the open loop system was about 6 seconds for the Newton-Euler system and 7 minutes for the Euler-Lagrange system. The difference in the calculation times is quite significant. The Euler-Lagrange system takes 70 times longer to simulate in this case. When the system is simulated in in closed loop the calculation time increases but the ratio between the two is relatively similar. The Euler-Lagrange simulations took about 80 times longer to complete.

[1]Dormand-Prince is named ode45 in Simulink

As the model fit very nicely with what I need, and also the attachment in the master thesis [18] provided both a maple script for deriving the equations of motion and producing some of the MATLAB code needed, and also a Simulink model of the system with PD controller I decided to use this model in my work.

I did make some changes in the maple script. I changed the vector of gravity and copied the MATLAB code produced, into the script used in Simulink. I did however not change the masses, the link lengths or their position in the script. As these things would only be approximated in any case, I decided not to use any time on this. These values would probably have to be changed at some point in any case.

The results when it comes to computation time suggest strongly that using the Newton-Euler formulation is the correct way to go, and not to use the Euler-Lagrange formulation that I where working on. The computation time has some bearing on the possibility of creating a real time simulation for the system. Creating a real time simulator for the system is the topic of the next chapter.

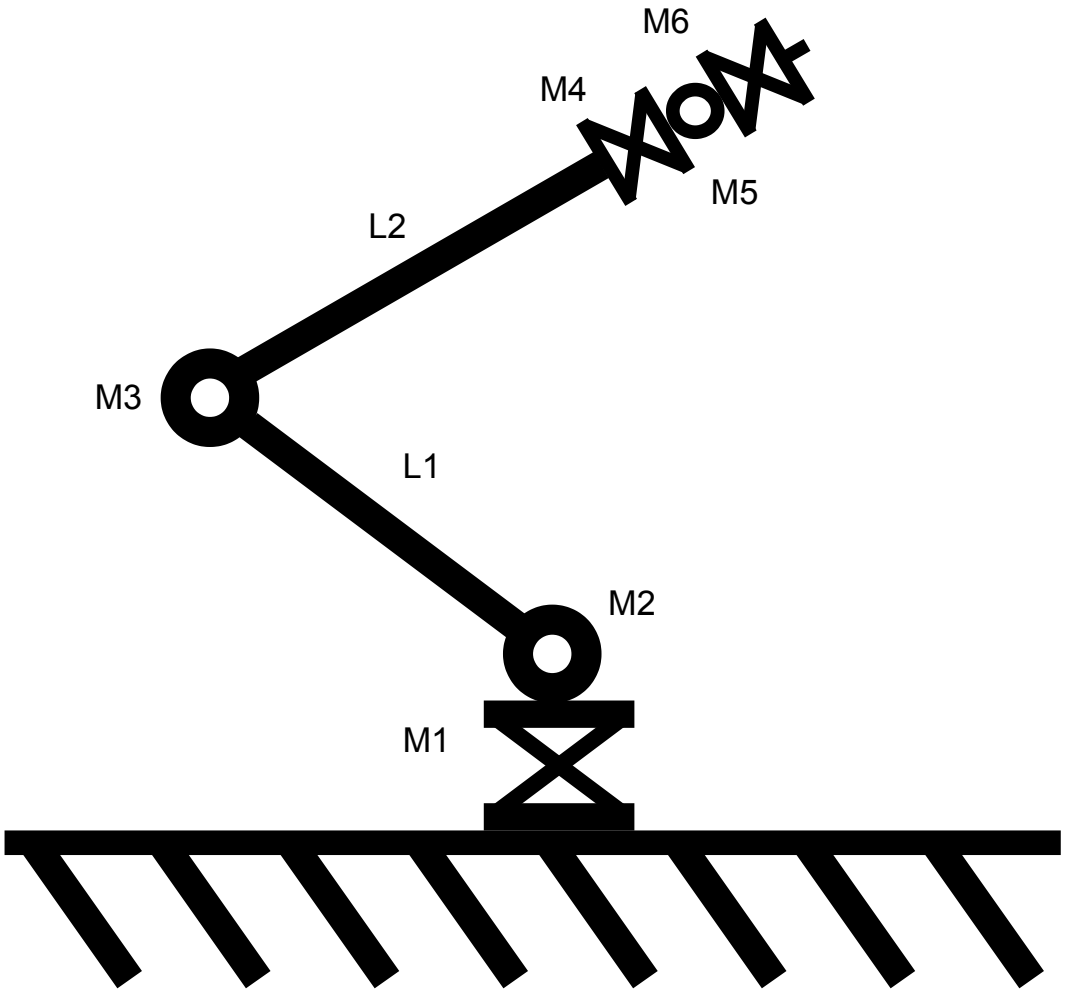

Figure 6.3: Free Body Diagram of the robotic arm. (Denavit-Hartenberg Convention)

Chapter 7

Simulation

Simulations are an important tool when it comes to designing controllers, and in many other areas doing a simulation is preferable to having many prototypes. To simulate the system in our case we want a real time simulator. This means that there are special constraints to the simulations. We need the simulation to be able to guarantee an upper bound on how much calculations need to be performed in each step, and that the simulation can run periodically with a predetermined period. To achieve this we need a fixed step integral solver.

In [25], something very similar to what I am doing in this project. They are making a real time simulator of a robotic arm being visualized in 3D. They use a fixed step solver to simulate the dynamics of the arm. They have however not done the arm modeling in Simulink or MATLAB, but instead used Dymola for these purposes. The arm they are simulating is however somewhat similar in that it has six degrees of freedom. The size of the physical problem is somewhat similar. In addition to the dynamics of the arm they have also simulated friction, power electronics, and the motors and gears.

While they have used Dymola in [13], they use Simulink to create real time simulations of the Puma 560 robot. This suggests that doing so here should be a viable solution.

There are several solvers to chose from, [15] lists some of them. I wanted to see what could be done with MATLAB when it comes to this, so the options do get somewhat limited by this.

When it comes to choosing a solver there are several concerns that need to be addressed. The solver need to be stable for the system being simulated. The step length can't be to short, as the shorter the step length is the more steps you get each second and thus the more computations need to be performed. Also, you want the solver to be accurate.

Before we get into the realm of fixed step solvers we needs to take a look at how we can create code in Simulink to do the simulations for us.

7.1 Code generation in Simulink

As mentioned in the previous chapter Høyfødt has made a Simulink model of the arm. This model I wanted to use further in the project to investigate if it would be possible to get a real time simulator for the arm.

NTNU buys licenses of MATLAB and some of the toolboxes. The license available to all students of NTNU does however not include one important toolbox, namely the Simulink code generation toolbox. Luckily the IME faculty has a license for MATLAB that does include that package. This makes it possible to convert a Simulink model into efficient C or C++ code, and also to create an executable. The code created is however not real time code but simply a fast execution of the model for however long you want to run.

To be able to do this you would in addition to Simulink, and the code generation toolbox also have one additional toolbox. Either the embedded toolbox or the Windows real time simulation toolbox. With this you should be able to produce a real time simulation directly from a Simulink Model. These packages are however not available for students only the staff.

That the code generator didn't produce code for real time execution was something I did not know until after having worked with the code generation toolbox. I did however see a possible solution for using the code generated by the code generation toolbox in a real time simulator. However before I get into this I want to explain some of the things that were necessary to do with the Simulink model created by Høyfødt to create code and some of the findings I made during this work.

7.1.1 Preparing The model for code generation

I found during my work that the diagnostic tools and error messages produced by Simulink to be very helpful when it came to the work done in this section. A lot of what I learned in this section came from trying to generate code and compile it with the code generation functionality in Simulink, and then following the directions and suggestions given upon some kind of failure.

To be able to perform the things done in this section there are some requirements to the MATLAB configuration that need to be made. You first of all need to have MATLAB with Simulink. Further you need the code generation toolbox and also a C or C++ complier that is set up to work with MATLAB. The latter you can setup in MATLAB by typing "mex-setup" in the command window of MATLAB. In the installation of MATLAB that is provided by IME a compiler was made available. However you can install different compilers if you so desire. Microsoft's Visual studio compiler is one alternative and so is the Borland compiler. Once these things are in place you should be good to go.

The model that Høyfødt has created can be seen in figure 7.1. There are several things that need to be changed in order for this model to be ready for code generation.

First of all the Level-2 M-file S-function block that implements the equations of motion for the arm model is not particularly suited for code generation. If you do want to use this block you have to write a TLC file that basically implements all of the model, or a TLC file that redirects the code generator to a C-MEX file that implements all the necessary code in the Level-2 M-file. I thought for a while that I would have to do this but with further research I found that the code being run in that block could be transfered with minor changes to a MATLAB Function block that supported direct code generation from the M-files.

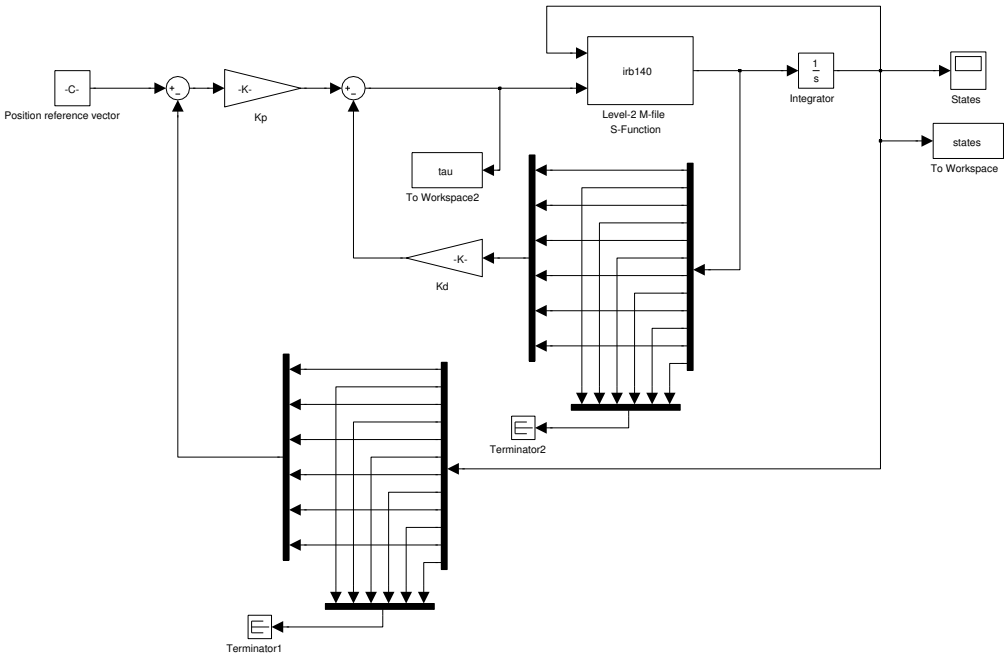

Figure 7.1: Simulink model developed by Høyfødt. This simulates the arm with a PD regulator controlling the position of each of the joints

After I had replaced the Level-2 M-file S-function block, the code generator let me proceed with code generation however it gave one error message and one warning.

The error was that the model contained one or more algebraic loops. An algebraic loop can best be explained by looking at the diagram in figure 7.2. For us it is relatively easy to see that the output here is a sum of the input minus half of the output, and that this will led to some kind of decaying output if the input

goes to zero. However, to produce code that can be simulated, the model has to be discretized and then it becomes somewhat more difficult to see what you are subtracting in the summation block. One way to deal with this is to discretisize the model before we generate code, and insert a unit delay in the loop as shown in figure 7.3. In this case it becomes easier for the coder to generate code as the sum block here is substracting the input to the gain block with the half of the previous input to the gain block.

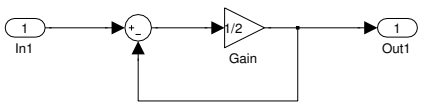

Figure 7.2: Example of algebraic loop

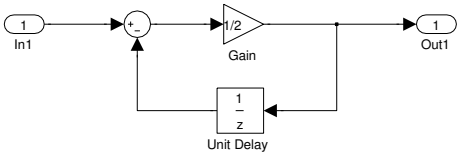

Figure 7.3: Solving the algebraic loop problem

The ordinary simulator in Simulink has special function to handle this however the coder does not have any way of handling this and so the loop has to be removed manually.

In the case of Høyfødts Simulink model there are three such algebraic loops one in the proportional feed back loop, one from the integrator back to the Level-2 M-file S-function block and one in the derivative feedback loop.

The algebraic loop in the derivative feedback loop can be dealt with by inserting a unit delay as has been shown in the solution example (figure 7.3).

The two others are solved in a different way that is related to the warning I described earlier. The coder gives a warning that it is best to discretisize the model before code generation. The only thing needed to do that is to replace the generic integrator block with a discrete integrator block. This block is part of the ordinary Simulink toolbox, and it should be no problem using it.

Using this integrator block reduces the number of fixed step integrator methods you can use. It actually only offers you three different methods. This will however not likely turn out to be a problem.

When the discrete integrator block is in place you have not only made the model discrete but also removed the last two algebraic loops as the integrator block can take care of this for you. The whole system can be seen in figure ...

7.1.2 Simulations in Simulink

Once you have gotten the model this far you have the option of using the accelerator option when performing simulations. This is can be very useful I did some

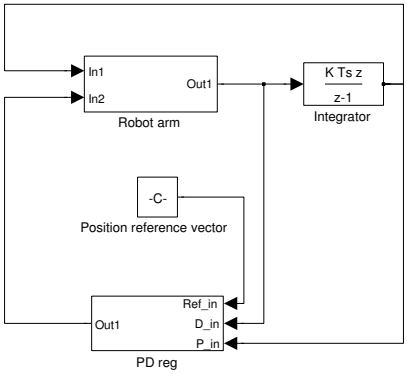

Figure 7.4: Simulink diagram of the system

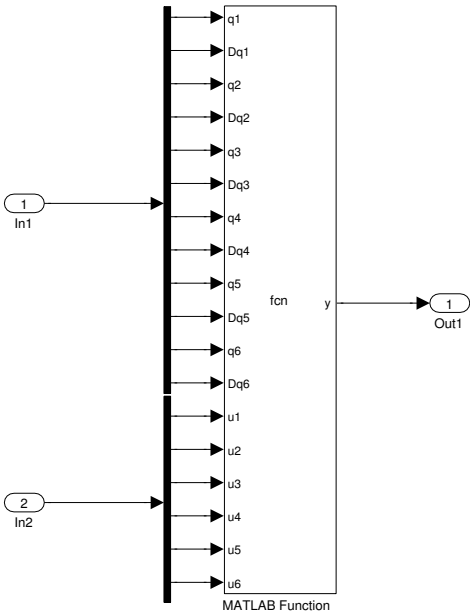

Figure 7.5: Simulink diagram of the robot arm sub system

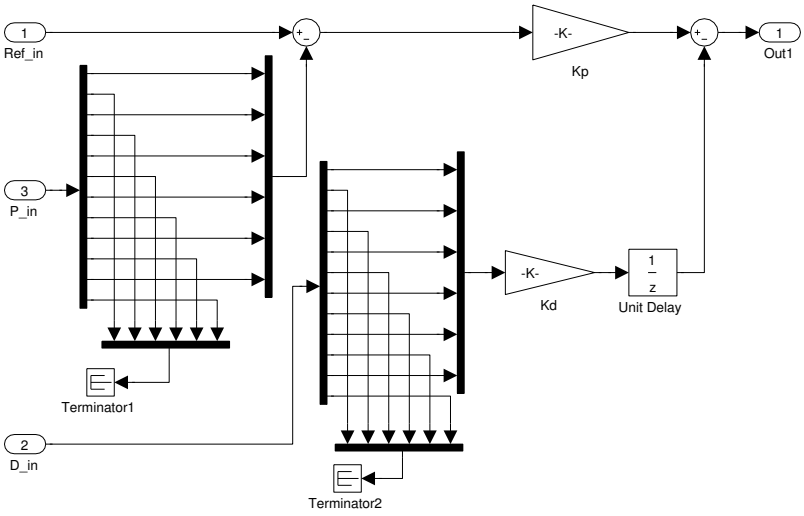

Figure 7.6: Simulink model of the PD regulator sub system

experimenting with this option and found that the runtime of simulations were very similar when using the accelerator option when simulating in Simulink and

when running the executable. This comes from the fact that when running the accelerator parts of the model is translated into efficient C code. This is very useful as this gives a good indication of whether or not a solver will be efficient enough, as you want the simulation to take slightly less time than the time span you are simulating. This is so you are able to see If it would be possible to run the simulation on the hardware you are on or not.

If you are not going to run the simulator on the same hardware the requirement that the simulation time be lower than the time span you are simulating does not appear however it can still give you valuable information.

There are some other reasons for running the simulations, you need to see that the simulation with the discrete one step method does not diverge to much from simulating the model with the standard variable step method Dormand-Prince (ODE45 in MATLAB). This has to do with the accuracy of the solver.

Another reason is that you want the simulation to be stable. A model that is stable under the variable step method is not necessarily stable with a fixed step method. This is dependent on the step length and the eigenvalues of the system. in this case these are time varying. Stability will be discussed in more detail later.

7.1.3 Real time simulation versus the variable step method

To find out if it would be possible to create the real time simulator I simulated the system created by Høyfødt for 5 seconds with the PD regulator regulating the position of the joints in the arm. And then I simulated the different fixed step methods available in Simulink.

When the model has been discretized, the number of integration methods available decreases to three. The discrete integrator block allows you to only use Forward Euler, Backward Euler and the Trapezoidal rule. When testing the different integration methods I was looking for two things stability and computation time.

The simulations has been performed on a relatively new Intel core i 5 processor. There are faster processors than this but it is a relatively good processor at the time of writing this report.

During the tests I found that the Backward Euler and Trapezoidal integration methods was stable at far lower step sizes than the Euler methods. However The computation time of the methods was so long that there seemed little point in going further with them. At a step size of 0.0005s the system was stable under both of those methods. The simulation time was set to 5 seconds but the computation time was far above the double of that. I tried doubling the step size of the methods as that should make the computation time half as long, but then the system became unstable. The simulations was tested using the Accelerator option in Simulink.

I also tested The Forward Euler method. This is the simplest integration method. It computes one step using only the past values of the system and the

derivative. The equation for this method is as follows:

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{7.1}$$

h is the step size in 7.1. This is an explicit Euler method where as the two former ones are implicit methods. Using this method I was able to get the system

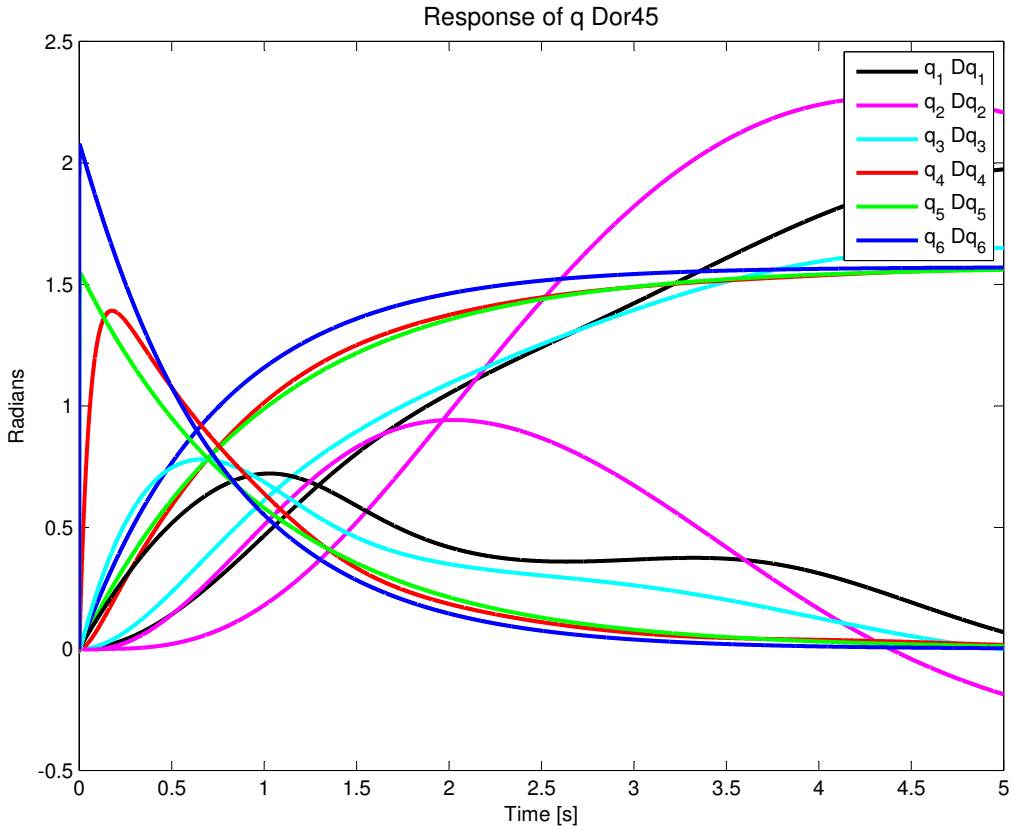

Figure 7.7: Plot of joints of the arm going from 0 degrees to about 90 degrees using the ODE45 integration method.

to behave well. And although I had to use quite a short step size (0.0001s), the simulation run time was below the time used in the simulation.

In figure 7.7 You can see the system being simulated using the standard integration method. As this method is a variable step method and the method is set to have a low error tolerance this simulation can act as a benchmark for how the optimal simulation should be.

The lines in figures 7.7 and 7.8 that approach $\pi/2$ or 90 degrees are the lines representing the angular position of the joints. The lines approaching zero represent the angular speed of the joints.

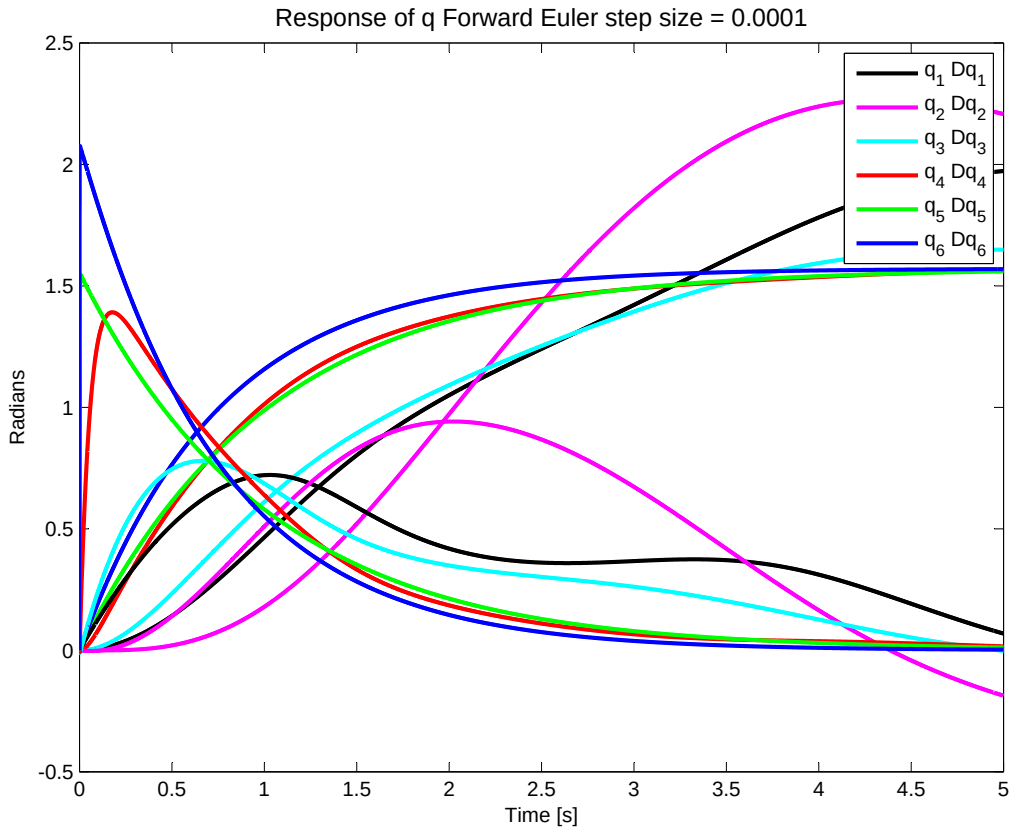

Figure 7.8: Plot of the joints of the arm going from 0 degrees to about 90 degrees using the Forward Euler method

The system is set up with all the initial values of the joint angles set to zero degrees. The reference angle for all the joints is sett to 90 degrees. As can be seen the regulator is not finely tuned. With some joints over shooting and others steeling in a bit below the reference value.

Fine tuning of the regulator has however not been an objective. But there is one thing that needs to be mentioned when it comes to the regulator. Høyfødt had set the values of the K_p and K_d vector relatively high compared to what I have used in my simulations (about 10 times as high). The system is stable when using Høyfødts regulation parameters when integrating the system with the ODE45 method. When running on a fixed step method this is however not the case.

7.1.4 Stability of the real time simulator

The reason why I lowered these values is that the stability of the integration method depends on how fast the dynamics of the system is. With a higher value on K_p the regulator becomes more aggressive and displays faster dynamics. The regulator parameters I have used is the same for all the plots.

As can be seen the three joints close to the end effector q_4, q_5, and q_6 has quite fast dynamics especially q_6 and q_5. Looking at the graph of the angular speed of q_6 we can see that it is an almost vertical line going from zero to about 2.1 radians per second. This is by far the fastest dynamics in the system and the reason why you need such a short step size to integrate the system.

If we simply look at what the Euler method does to solve the system, we can get a sense of why it can fail and become unstable. The method uses the derivative multiplied with the step size to get to the next point. If the step size is to long and the value of the derivative is to big this leads to instability. The line going from one point to the next will over reach the correct value.

An interesting point when it comes to the Euler method is that it will actually add energy to the system. The easiest way to illustrate this is with a figure 7.9. In The figure the red lines represent successive steps of the integration method, while the blue line is the correct value. As the Euler method uses the derivative at the point of the curve it is on it will diverge from the real solution and pick up more and more energy.

The reason why you don't see much of this behavior when it comes to the robot arm is that the step size is large enough for this not to be visible and also because the regulator will force the system to behave well. If on the other hand you had simulated this system in open loop, with the Euler method, you would expect the system to pick up speed and for this to grow larger and larger as there is noting in the open loop system to dissipate this energy.

This brings us to another point. The simulation presented here does not simulate an entirely realistic system. Aside from the simplifications in the inertia matrix, you would expect there to be quite a lot of friction in the gears of the real robot. Also there is a limit to how much torque any given motor can deliver with any given power electronics.

When it comes to friction and motor models [25] reports success when it comes to creating a real time simulator of an six DoF arm that contains both a motor model power electronics and a friction model.

The friction model they use th LuGre model [15] this is a continuous model. This fact is probably of importance as large discontinuities could cause the derivative of the model to become very large and thus it could cause the integrator to fail.

I suspect that if you add a friction model to each of the joints in the robot

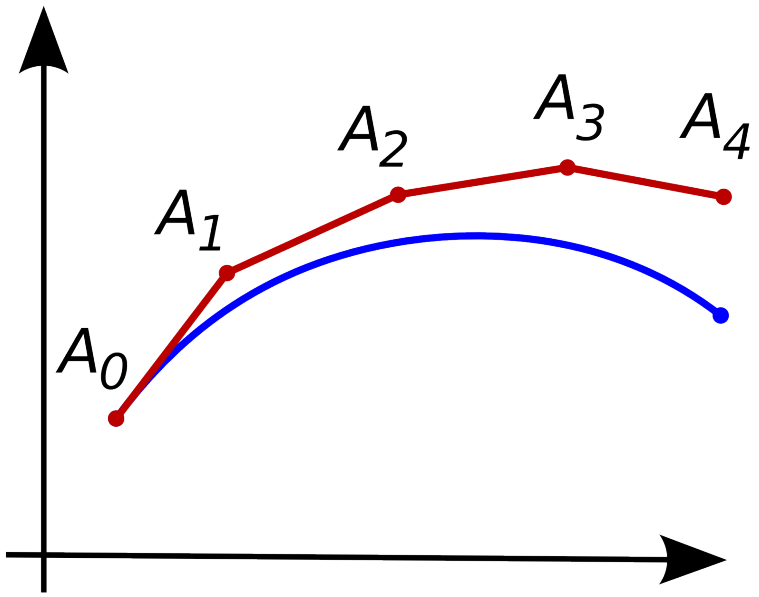

Figure 7.9: Illustration of Euler method accumulating energy (Picture taken from Wikimedia commons)

some of the faster dynamics of the system might be dampened out. Also an upper limit to the torque delivered to each of the system might remove some of the fast dynamics of the system. If these things were modeled they might enable you to increase the step size. Thus despite the fact that the models has more computations in each step the total system might be more effectively simulated.

This might be important if it is necessary to extend the model with the additional degrees of freedom as the equations of motion will become larger if this is added.

If the equations of motion turned out to be to big if you try to model them all, you could try to model the translation of the cart and its rotation and the three first joints of the robot. The mass of the three last joints might be so small that movement in these joints does not effect the system to much. You might expect that the real regulator on the robot can compensate for any disturbance that movement in the last joints can cause. Thus you don't get such a large discrepancies between the simulated model and the real robot. The last joints might then be modeled separately from the rest of the robot.

7.2 Creating real time simulations from the code generated

I had initially considered simply generating the code and then inserting a sleep statement in the code that would enable the simulator to execute the simulation in real time. I found where in the code the reference values were used in the code. This reference value can be replaced with a command input from the user.

However the sleep functionality does not really give you proper real time performance as it does not guaranty that it will wake up on time. You could get better results with a spin lock mechanism but this is a somewhat clumsy method to use.

A better method would be to run the code under Linux with the RTAI [33] extension. This should be possible as according to the help files the code generated should be portable between Windows and Linux.

There is also another alternative, MATLAB boast of being able to give real time capabilities to windows trough their Real-Time Windows Target toolbox. It boast of being able to simulate systems with sampling frequencies approaching 20KHz. As the simulation being run here requires around 10kHz this should be within the limits of what can be done. It should be noted that the real time capabilities relies upon a kernel level extension (provided by MATLAB in the toolbox) to be able to do this. It would not be possible to run real time code on windows with out this extension.

The simulations presented in this section does not guarantee that the simulation will stay stable. Indeed B-stability or Algebraic stability, which are conditions for proving stability of nonlinear contracting systems can not be proven for explicit Runge-Kutta methods such as the Euler method this can however be proven for the backward Euler and the trapezoidal rule [15].

To handle this it would be desirable to monitor the some of the states of the system such as speed or acceleration. You could do a check to see that these states don't go beyond a certain level and then throw an exception and handle this appropriately

Chapter 8

3D models and rendering

I started to work on rendering the models before the work on the simulation. It was important to find a good platform for rendering the models and possibly also to make the GUI.

When it comes to the 3D models one might like to use in this project it would be highly beneficial if one can use the 3D data you would normally have as building documentation/specification for a specific wind turbine. However it is important to remember that there are different requirements and manners of modeling between the models used in documentation and models used in games.

When it comes to games you have to worry about how many polygons your model has. If your models consist of to many polygons the models becomes slow to render onto the screen and you can quickly experience lag. To make environments and actors look rich in detail and "biological", games rely upon texturing[1] and anti aliasing[2].

In making a 3D model for construction purposes you are in general not concerned with how many polygons your model has. Indeed how many polygons are used in the representation of the model you are making is of no other concern than how fast your computer is at rendering the object when you are working on it. And in many cases the number of polygons used for representing an object can be set dynamically by the rendering engine. The modeler are only concerned with the exact mathematical representation of the model. An example of this cold be a cylindrical shaft. The modeler sets the diameter and height of the shaft, and once that is done how many polygons are used to render this is arbitrary to you, so long as it looks okay and the rendering proses is not to slow to imped further modeling and manipulation of the model.

[1] Textures are images laid upon the 3D model to give color or patterns to the model and also to avoid the need to increase the number of polygons in a model

[2] Anti aliasing is an effect that helps in removing sharp edges giving an appearance of higher detail and smother lines

In making such a shaft for a game you will at some point have to decide how many polygons are going to be used to represent that geometry. There are algorithms that can do this automatically however this is something that is considered to expensive for the game engine to do on the fly or upon loading a new 3D model.

The two different schools of 3D modeling has allot of different tools, and there are basically one set for the engineering and architecture world and one for the gaming and film industry.

One tool that is commonly used by engineers is Autodesk Inventor. Last semester I used Inventor to create a model of a 5MW wind turbine using the photo shown in 8.1.

Figure 8.1: Photo of 5 MW wind turbine (picture taken from Wikipedia)

Figure 8.2: Wind turbine modeled in Inventor

To get a good understanding of what tools are needed and what challenges one wold encounter in using the 3D models you would normally have when you design and build a wind turbine, I wanted to use the model I had created in Inventor in this project. I also wanted to use tools that where freely available. I ended up

going with a game engine called Panda3D [30] to provide the rendering capabilities. I also wanted to use Blender [7] 3D as a link between Inventor and Panda3D as Panda3D has a community supported script for converting the Blender models into egg files that can be used in Panda3D. I did however know that an alternative to Blender would be to use Autodesk Maya as this is also supported by Panda3D.

I consider it important to convey the experiences and solutions that I have found to work well during the course of the project. Also the steps involved in taking a model from Inventor to Panda3D. For these reasons I will go through some of the tools used in the project and some that have been considered but that I did not find to be fruitful.

I have gone into some detail here when it comes to the operations that need to be performed. This is so that if somebody decides to take the same path as I have taken here, and they have read this they don't have to research the methods used so heavily. I have chosen to explain the things done in Maya and not Inventor as once you get the models most work to get it in to Panda3D will have to be done in Maya.

8.1 Panda3D

To visualize the robot arm and it's environment you need something to render the 3D models you have. Making a system that does this from scratch is a big challenge and something that would take quite a lot of time, making it near to impossible for me to do anything else. For this reason the best option is to look at what is freely available.

I wanted something that would render 3D models. The most obvious alternative for this seems to be to use a game engine to do this. Game engines usually have the capability to render both 3D scenery and models as well as display 2D graphics, they are designed as a software framework to use and provide you with a lot of functionality that you otherwise would spend many hours and a loot of money to redevelop.

The choice to use Panda3D came rather early in the project phase. It has many features that makes it compelling for this project. It is distributed under the Modified BSD License, meaning it can be used for commercial purposes without any extra cost. It boasts of being very easy to learn and get started with. The functionality of the game engine can be accessed through the use of either Python or C++.

The game engine was created by Disney and has been used by them for some of their games. It is now in use as a teaching tool by Carnegie Mellon university (CMU) in Pittsburgh, Pennsylvania. CMU also has people working to further develop the engine and the next stable release will feature multi core capabilities

through pipelining, Bullet physics, LibRocket GUI, and much more.

As this engine is used by students it has become very robust and can handle many errors using many assertion expressions to catch errors early.

My experience with it is that it is indeed not all that difficult to use. Many things are done for you, however some of the features and functions available are not very well documented. This is especially the case when it comes to the Mopath class. This provides functionality to move an object or camera along a nurbs[3] curve. In this case there is a small section on it in the manual with some cursory information on some of it's capabilities, but no examples. And in Panda3D's documentation neither the class itself or any of it's functions or publicly available variables are explained. To get an understanding of how it worked I has to instead look at the code. This critique having been made it is important to note that there is a solution to it namely that you can actually look at the code.

The code I wrote and an explanation of it Panda can be found in Appendix A. I will however touch upon some of the things I have found in using Panda3D here.

8.1.1 Using Panda3D

thing that became apparent when it comes to using Panda3D is that you don't need to use very many lines of code to take advantage of the features it offers. A further benefit of Panda3D is that Names of Functions and Classes are easily understandable with perhaps a few exceptions. However you do have to use some time to research the functions you use.

Quite a lot of the code used in this project is actually code that is run at setup. Loading 3D models setting up the skeleton of the robot arm getting the two display regions and so forth are all things that you only have to setup correctly in the start. After the initialization underlying subroutines takes care of the rest for you.

Panda3D does of course offer you a way of controlling what happens during the runtime of your application. This is done trough two different systems namely tasks and events.

Tasks are functions that are executed between the rendering of every frame of the scene. As such the tasks are not being executed with a specific guaranteed interval. Tasks are used to animate your scene or the actors, move cameras and so forth. New tasks you make are simply functions that you register with the taskMgr object.

[3]"Non-uniform rational basis spline s a mathematical model commonly used in computer graphics for generating and representing curves and surfaces which offers great flexibility and precision for handling both analytic (surfaces defined by common mathematical formula) and modeled shapes." - Wikipedia

In the code supplied there is one function that is registered with the taskMgr objects. This is the move task.

Events are not executed between every new frame but are akin to interrupts. The ability to register new events or event handlers are given to any class that inherits from Panda3D's DirectObject class or if you inherit a class that itself has an inheritance from DirectObject. Events are useful for handling such things as taking action after the window has been resized, handling mouse clicks either of 3D objects or of 2D objects such as buttons text fields and the like.

In appendix A, some of the code written for use with Panda3D is presented, and to some extent explained.

8.1.2 Useful features in Panda3D

There are numerous features in Panda3D that can be put to good use for our purposes. Here I will present some of these features.

Video support

Panda3D has support for MJPEG through ffdshow. MJPEG has been used in both the old cart and rail system and in the project work done by Hung Buy. This is not the only video format supported by Panda3D it should be able to take advantage of all the formats that the ffdshow decoder can decode.

As far as hoking this up with a stream coming from the Internet this I have not tested. The documentation say that panda3D should handle any movie stream, so it should be possible.

Motion path

As the robot is restricted to follow a rail inside the nacelle it might be nice to only have to specify the position the robot is at along the rail, when any new frame should be drawn. One possibility would be to specify the position using the position of the robot in x, y and z, in addition to the angle. This can however be done far more easily using functionality provided by Panda3D trough their Mopath class.

The motion path is simply a nurbs curve that you create in a tool such as Maya. After having loaded the curve into the application you can restrict objects to follow these curves and specify where on the curve the object should be drawn. You can also specify that the object should be animated with a certain speed along the curve. This functionality it should however not be necessary to use.

Portability

Most programing with Panda3D involves using the classes that Panda3D supplies to you. In addition when programing you program in Python which is not all that different between platforms. While all of this is true you need binaries for your system. This can either be obtained by compiling the source code from scratch, as all that is need for doing so in either Ubuntu, Fedora, Debian and other distros are available for you or downloading a binary for your distro. However they haven't got binary for the latest distros, when it comes to the version I have been working with Panda3D 1.7.2 the latest Ubuntu binary available is for Ubuntu Maverick.

As I have not programed any ting in my program that uses any windows specific resource the code runes fine on Ubuntu without any problems. When Panda3D is installed on the machine.

Looks and appearances

Panda3D does of course support texturing, no modern game engine can live with out it. Doing texturing well is however somewhat of a chore. Work with texturing is mostly something you have to work with in your 3D modeling tool and it's not something I have chosen to concentrate on.

I do however have some ideas when it comes to texturing. You could use different colored or patterned textures to give clues to the operator of exactly where the robot is inside the nacelle and thereby making the map view less important to have.

It is also possible to change textures of parts inside the model. You could for instance let some of the images from the camera be loaded up to the model when the camera has seen that part of the nacelle. The old textures can be darkened before the model is loaded. This way you can see what has been seen, as this wold have a brighter hue, and what parts of the model has not been seen.

Transparency is also an available option to use on textures and models. This could be used to give the operator a visual cue of areas that he is not allowed to move into. This could be useful in high voltage areas.

Handling large and detailed scenes

The number of polygons in a scene is not so big of a problem any longer most graphics cards can handle millions of polygons in one model. However having multiple meshes can easily become a problem. Panda3D does offer some tools to alleviate this. You can use a method called flatten on a node in your sen graph Panda3D will then combine these meshes into one that the graphics card can handle.

There is also a technique called the rigid body combiner this makes it so that the CPU can handle some of the calculations when it comes to the position of different meshes. And like the flatten command the graphics card can consider these objects as on mesh. This technique does allow you to manipulated the objects and treat them as separate objects unlike the flatten command. If you want to know how many meshes are under one node you can use the command exsamplenode.analyze() on that node.

Collision detection

Collision detection is a useful tool that Panda3D offers. I had initially only considered it to be of importance when it comes to handling the camera, so that it does not go trough walls. This may turn out not to be the case though, however this will be further discussed later.

Collision detection is a somewhat complex matter Panda3D offers two different ways of collision detection. One way involves creating extra geometry in the models that is not rendered but used for the collision detection. This method is more effective when it comes to the execution.

The other way just involves using the geometry of the models you are rendering to accomplish the task. This method is faster to implement, at least on simple models. It is however not the optimal solution when it comes to computational efficiency. As the model used in the rendering of the nacelle will likely become quite complex it will probably be necessary to use the first method.

When it comes to using the first method Panda3D offers different shapes that can be used as collision objects. These objects can be divided into from and into objects where the from objects are usually connected to the moving objects in your scene. The into objects are best suited for static objects such as walls and floors and the like.

The different objects that Panda3D offers that may be of use are the CollisionSphere, CollisionTube, CollisionPlane and CollisionPolygon. There are some more however these are probably of less use to us.

The most useful collision object for us when it comes to a collision from object is probably the CollisionSphere. This can be used to detect collisions with the camera so that you can make certain that it does not pas trough walls. It can also be used on several spots on the arm to check for collisions here. The sphere is also the only object that is suited to be both an into and a from object.

The rest of the model will have to consist of CollisionTubes, CollisionPlanes and CollisionPolygons. These objects are suited as into objects.

The CollisonPlane is not suited for use inside the model. This comes from the fact that it stretches infinitely in all directions. It can however be effectively used to box in the robot. It is far better to use than an CollisionPolygon as this is a

somewhat less effective object to use in therms of computations and reliability.

The CollisonTube and the CollisionPolygon will be suited for detecting collisions of objects inside the nacelle. This might however be the most difficult part to model, and model well. The polygon is also somewhat unreliable.

Once you have constructed the collision model you need to program what will happen in the event of a collision. When a collision is detected an event is registered and a Handler for that event has to be set up. There are some event handlers already programed in to Panda3D. The one that is probably the most useful is the CollisionHandlerPusher method this can stop an object from moving in a specific direction or let it slid along a collision object.

User interface functionality

Naturally you can create buttons that you can click on with the mouse, this is functionality that may be useful in the further development of a GUI.

It is also possible for you to make certain 3D object mouse selectable. This can be used to give pop up information to the user. In addition its possible to create menu's that pop up with clicking objects in the 3D environment.

Overlaying text on top of the video is also possible to do. This can be used to give alerts show data and give messages to the operator. (Doing this is not very difficult.)

8.2 Autodesk Inventor and other cad tools

Autodesk is perhaps the largest and most known company devoted to 3D modeling and CAD tools. Their flagship has for quit some time been AutoCAD. The tool was released in 1982 [3] and is used by many professions both in engineering and Architecture. Autodesk inventor was first released in 1999. This tool is meant to be a help in rapid prototyping it allows you to make a 3D model and then explore the soundness of your design be it strength requirements, range of motion and more, before it is decided to make a prototype. In some cases you can go from quite complex designs to the finished product without actually making any prototypes at all.

Autodesk Inventors main competition is SolidWorks, Creo Elements/Pro (formerly Pro/ENGINEER) and CATIA where SolidWorks is by far the most commonly used solid modeling software. That being said, Autodesk AutoCAD which is the dominant engineering tool has recently gotten quite a lot of support for solid modeling. It is in fact hard to see the difference in the tools made available for solid modeling between Inventor and AutoCAD.

To be able to construct the models needed for the virtual world it is most likely necessary to deal with some models that are not created with Inventor or AutoCAD. Luckily Inventor supports importing models from all its major competitors mentioned above [4]. In cases where it does not the STEP or sat, are formats often used for file exchange.

8.3 Blender

Blender is among others a 3D modeling tool it is distributed under the GNU General Public License and as such is free to use. As mentioned there is a script for converting the .blend files of Blender into the .egg format that Panda3D accepts. As the tool is free and also quite advanced, being put to good use in both the film industry and the gaming industry I wanted to use this tool to take the files I had created in Autodesk Inventor.

Blender boasts of being able to import DWG and DXF files trough user supported scripts. I took this to mean that it would be possible to import the models I had already created. I proceeded to learn some of the techniques I would have to be able to use to successfully use Blender together with Panda3D. Among them how to export the .blend files to .egg files. How to construct a skeleton to be used with the robot to be able to properly and easily articulate the robots joints.

A lot of time was spent on learning how to do these things. One thing I found was that many key functions of blender has to be accessed with hotkeys. This meant that I had to learn quite a lot of different key combinations to be able to use the program. While this is probably something that will be valuable for the seasoned Blender user, who is likely to use the same or similar features of the program many times. For me who wanted to pick this program up and simply use it to quickly do what was required there and then, this was not something highly valued.

After some time spent learning the program I felt ready to prosed with importing the Inventor models. This turned out to be more difficult than I had anticipated.

As explained in the previous section Inventor uses Nurbs to describe the shape of the models you have made. While DWG and DXF files are capable of containing Nurbs data, and Blender is capable of understanding and producing models with nurbs curves, the import script was incapable of translating the nurbs data in either the DWG files or The DXF. The import script was in fact only able to successfully import polygon data.

Faced with this I was left with two options eider extend the capabilities of the import script to be able to translate the nurbs data in the DWG files into the acceptable format for Blender, or instead use Autodesk Maya as the link between

the Inventor models and Panda3D.

As I didn't know how hard it would be to rewrite the import script to handle the nurbs data and how long such an endeavor would take i chose instead to Install and use Autodesk Maya.

As such trying to go down the path Of using Blender turned out to be a failure.

8.4 Autodesk Maya

One of the things learned from my experience with using Blender with the intent of exporting the data to Panda3D is that you can't necessarily grab the latest version of the software and expect there to be a ready to use script for converting files from one to the other. In the case of Blender the most complete egger script was written for Blender 2.4x, and not the latest stable release[4]. This turned out to have it's equivalents when it came to using Maya in conjunction with Panda3D. When I was ready to start installing Maya the egger for Maya 2011 had recently been released. As such the 2012 version of Maya was not an option to use.

As Maya is developed and distributed by the same company that also develops Inventor and I knew that file exchange between the the different products Autodesk sells has been a priority for them, I figured that it was likely that the programs would be able to exchange files. This I found to be the case as Maya is capable of directly importing the native files used in Inventor. Maya is also available under a similar student license as Inventor.

Another benefit of using Maya is that this is the modeling program used in the course at CMU that teaches game programing, and is in part responsible for the development of Panda3D.

I found that the interface of Maya was far easier to grasp than that of Blender. You are not forced to immediately learn many hotkeys to use the basic functionality of the program. There where still some things that didn't feel completely natural, however largely Maya proved to be easier for a beginner to learn and quickly be able to do basic things with.

8.4.1 Taking Models From Inventor to Panda3D

The first thing I attempted this time was importing the Inventor models into Maya. This went quite smoothly. The version of Inventor I used to produce them was Autodesk Inventor Professional 2012. This was no problem even though the version of Maya I was importing the models into was the 2011 version. Next I attempted to export these models using the Mayatoegg2011 script provided with the Panda3D distribution. The script exited without any errors however when attempting to use

[4]As of 26.04.12 the latest stable release of Blender is 2.62 for the Windows platform

the newly created egg file in Panda3D noting got rendered. After a little research I found what most game programmers and modelers probably learn pretty early on, game engines don't render objects described by nurbs.

Graphics cards and GPUs[5] are in general designed to render Polygons. If you were to use models described with nurbs curves you would first have to run an algorithm to tessellate the model so that it is represented with polygons and then render this model. Because this would be somewhat costly in terms of the time it takes to do on the games load time, and the models are not guaranteed to be optimized for game performance this is not done.

The next step was then to convert the models from its nurbs representation into polygon meshes. I found that this could be done by going into the Modify menu selecting the convert pane and then selecting Nurbs to polygons. You now get a menu where you have several options when it comes to tessellating the model. I found that having the "Attach multiple output meshes" option on and using the standard fit method on was the best option. The option to Attach multiple output meshes is necessary to prevent different surfaces to be disconnected after the tessellation.

As far as the Standard fit option goes this allows you to select different parameters that influence the tessellation. The objective is to get a model that to some degree accurately represent the model while not creating to many polygons. As far as the parameters you can influence the tessellation with I did not find one setting that suited all the different models so this is something you have to experiment with. A useful tool when it comes to checking this is to go to the "Display" menu select "heads up display" and "Ploy count". This lets you see how many polygons are in the current scene.

If I simply converted the models from its nurbs representation into the polygon mesh I found that you get far more polygons than what you actually need. The models created in Inventor specifies many features of internal design of components that are not of any interest in helping us to navigate the robot. For example the outer walls of the nacelle are of no interest to the controller as he is concerned only in what goes on inside the nacelle.

I thus found that the best option was to remove nurbs surfaces that would not be made visible to the operator. This way the models will take less memory and the GPU will have less work to do.

There was still a problem with the models that became very plain when they where rendered in Panda3D. The meshes often had holes in them that you could see through. To remedy this you have to combine the meshes. This can be done by selecting these meshes and then in the "Polygons" menu select the combine icon[6]

[5]GPU: graphical processing unit

[6]The names of icons and a description can be seen in the bottom left corner of Maya when

after this the meshes has become one Object with a common transform to describe position and rotation however not a combined mesh. To combine them you have to select the “Merge” option in the “Polygon” menu.

There is also another problem when it comes to the polygon count. The game engine calculates the light level of each polygon separately. If you have to few polygons in a mesh this becomes obvious when it comes to the rendered model. In the models used in my application I have used a very low polygon count on the models assuming that you want to import a lot of different objects in the final product. In the application you can see the effect of this on certain areas of the model. To illustrate this you can look at figure 8.3.

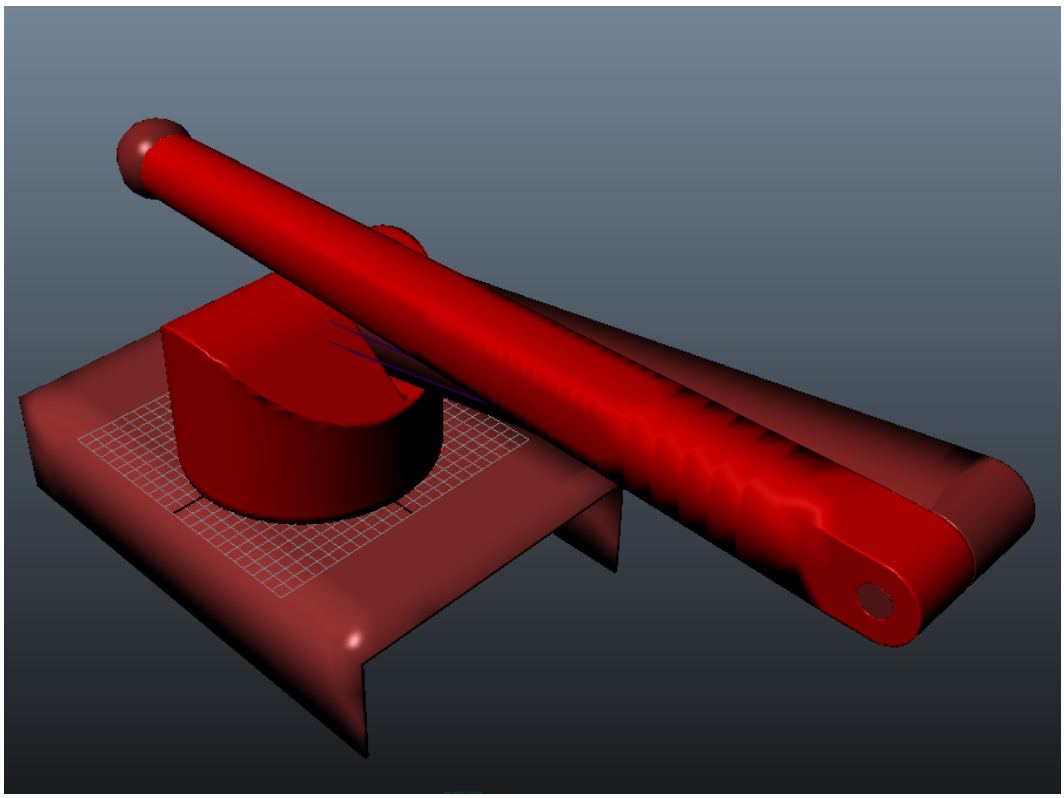

Figure 8.3: Rendering artifacts from low polygon count.

As can bee seen the polygons capture the shape of the arm well but the low polygon count near the beginning of the second joint makes the rendering show jagged edges.

Once you are satisfied with the polygon count and appearance of the model you should remove the Nurbs representation of the model. The simplest way to do

you hover the mouse over the icon

this is to go into the “Hypergrap hierarchy” that can be found in the Window pane. Here you can see the different components of the model and select the components you want. It’s sometimes easier to select things from this menu than in the 3D display window. Select all the Nurbs curves and delete them.

You can now customize the model with different colors this is done by right clicking th object you want to color and then in the shader pane you select the phong shader and then you can eider select a color for the object or you can do more advanced things with texturing by selecting a different option than coloring.

You are now ready to create an egg file. This is done in the command line by invoking the correct egger. In the case of Maya 2011 the line you use is: “maya2egg2011 -o out.egg in.mb”

Once I was able to go from the Inventor model to something that could be satisfactorily rendered in Panda3D the next step was to create a skeleton and associate the joints of the skeleton with the joints of the robot model. And then Import this model into the Panda3D application.

8.4.2 Robot arm and Skeleton

Creating the skeleton was fairly easy. Perhaps the most challenging part was to get the joints in the correct position. once this is done you need to associate the correct robot links with the correct joints of your skeleton. This can be done rather easily in the case of this robot. Normally with models of biological creatures where you want the polygon mesh to deform you need to go through a procedure called skinning which can be quite involved and time consuming. In the case of the robot arm there are no meshes that need to be deformed. each of the links are separate meshes and they only need to be rotated and transformed by the joints.

When it comes to attaching the robot arm to the skeleton it’s a simple mater of parenting the individual links of the arm to the hierarchy of joints in the skeleton. Doing this is once again easiest to do in the “Hypergrap hierarchy”. You want each mesh representation of links to be the child of the correct joint in the skeleton. This can be done selecting first the joint and then the link mesh and then going in to the “Edit” menu and selecting “Parent”. Once this has been done you can verify your work by looking at the Hypergrap hierarchy and checking that the structure is correct. You must also create names for the joints in the skeleton so that you can find them in Panda3D. Once the all the links of the arm is connected to the correct joints in the skeleton you are done with Maya.

Now you need to get the model into Panda3D. The standard method of creating egg files from Maya for Panda3D is not the correct one when it comes to models with joints. The correct lines to use in the command line tool in the case of the models I have is:

- maya2egg2011 -a model -o out.egg robo.mb
- egg-optchar -o out.egg options in.egg
- options = -expose jointSwivel,jointFirstArm,jointSecondArm,jointCamera

You need to run the egg-optchar so that Panda3D does not remove access to the joints when the file is loaded in to the application. The options has to be together with the second line and there can't be any spaces in the names or between the comas.

8.4.3 Motion path

To create the motion path you need to have a nurbs curve in the model. There are several ways to do this however some ways are easier than others. I found that to get a good fit with the rails the easiest way to create the curve, was to import the rail from inventor, and before creating the polygon mesh I create a nurbs plane that intersects the rail where i want to make the path. Once this is done you can create a curve in that intersection.

Once this is done you can remove any exes geometry and then proceed with creating the polygon mesh. The proses of creating the egg files does not change from the standard method. The nurbs curve is in the egg file.

8.4.4 Evaluation of Maya

A benefit of using Maya is that this is the modeling program used in the course at CMU that teaches game programing, and is in part responsible for the development of Panda3D.

I found that the interface of Maya was far easier to grasp than that of Blender. You are not forced to immediately learn many hotkeys to use the basic functionality of the program. There were still some things that didn't feel completely natural, however largely Maya proved to be easier for a beginner to learn and quickly be able to do basic things with.

The time spent learning the tool was not all to long. However I have not tried to learn how to do the more advanced texturing methods. This could take some more time.

The proses of converting the models from their cad format into something that can be used in Panda3D does not take all that much time. This is naturally of importance.

8.5 The application created with Panda3D

In figure 8.4 you can see a basic outlay of how the application looks now. In the top right corner I have created a display of a video to illustrate that Panda3D is capable of playing of video streams although this video stream is not generated by a webcam. The video file is however a MJPEG video file, a format that has been used in the web cam applications that has been created previously.

In the bottom left corner A 3D map view has been created. This is simply the same 3D models that you can see in the top left corner. This view can be changed by pressing the t button on the keyboard. This changes the camera viewpoint and angle so that you see the seen from the other side of the nacelle.

To be able to see through the walls of the nacelle, you actually don't have to do anything special to the model. The models are set up by default to be rendered from only one side. as the camera is placed on the "wrong" side of the wall this does not get rendered to the camera. You can set up the model to render objects both on the inside and out but there seams little point in doing so, and in this case not doing so can be used for some good.

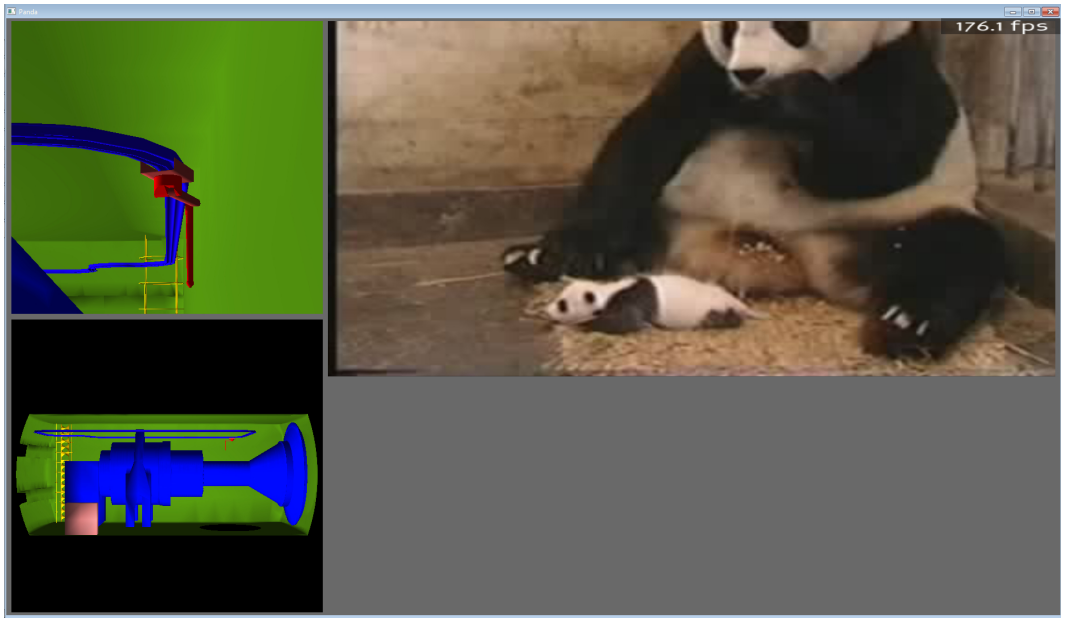

Figure 8.4: The Graphical user interface created with Panda3D.

The top left corner gives you a close up view of the robot. The camera is set to always follow the robot at a given height and an approximate distance.

The robot can be controlled with the arrow keys and the a, s, z and x keys. The forward and backward arrow keys moves the robot forward, while right and

left controls the angle of the base of the robot arm. The other keys controller the angle of the two links. The end effector I have not bothered with articulating in this manor although it wouldn't be particularly difficult to do.

The control of the robot is obviously not meant to be handled in this manner. However for now this is how it is done.

One thing that is possible to do with Panda3D is to create a 3D display region on top of the video that is mostly transparent apart from some objects in the scene that are only partly transparent these objects can be objects that you might like to click on to get information on them. That the position of the real objects ant the rendered objects would rely on precise data on the position and angle of the real camera. If you have this the virtual camera can be set to render the scene from the same position and angle.

If you would like to test the application I have included an installer in the attachment. This installer makes it so that you don't have to install the SDK or the Runtime binaries.

8.5.1 Some observations on the application

I did some testing rather late in the project of the application as it is now. The reason why I didn't doe these tests earlier is that before I had gotten so far that I could test it I started to look at the simulation aspect of this project.

In the limited time I did test the application I did notice some things that are worth mentioning.

One of these things is the attention I gave the different parts of the screen. When trying to control the robot I discovered that it is very hard to focus on any of the other screens. My gaze was transfixed on the top left corner of the screen when ever I tried to move the robot. This you might argue is not so strange as the video does not help much in the navigation, however this might still suggest that when you are control the robot you could make the third person view the main part of the screen area.

Even if the video had shown a view of a real nacelle or just another rendering from the point of view of the camera at the end of the arm. The fact that you can't see the arm makes it hard to know how the arm is situated in its environment. This would especially be the case when you have an arm that has six DoF. If the angle of the camera is not pointing straight with respect to the second joint it would be very hard to obtain or maintain a mental model of what configuration the arm has or for that matter be able to say anything about its position in its environment.

Further it is very difficult to judge the distance to the walls of the nacelle. This makes it very easy to make the robot crash into them. This problem could probably be solved with adding additional detail to the model, either in the form

of more 3D objects or in the form of textures. One idea I have is to use square tiled textures on the surfaces that are very uniform or where you lack enough detail to be able to properly judge the distance.

The square tiles would help you with judging the distance from the camera to that object. This it would do by both giving you a reference object to look at that changes it's size on the screen as you get closer to it. In addition the pattern should allow you to gain a sense of perspective. It can also give information about the shape of objects if the square texture is stretched over it.

The color scheme of these textures can also be different depending on where in the nacelle you are, thereby giving you more information in one view. The square textures would probably also be easier to create than one that attempts to give a more accurate view of the nacelle.

A third observation I have is that sometimes the camera can go trough certain objects. As the objects are only rendered from one side this can be hard to detect. When the camera is moving inside an object you can't detect that this object might collide with the arm.

It would be better if the camera was equipped with collision detection against the rest of the scene, so that the camera would slide along the walls and objects. That way the operator would know that it might be an object behind him that the arm can collide with not one in the view of the camera that is simply not rendered. It would not be a solution to make the application render objects from both sides as this would simply obstruct the view of the robot.

One final remark is that the camera angle and view point could probably be made in a better way. Perhaps by letting it always stay behind the second joint at a certain distance and at an angle to the joint. It might also be beneficial to give some cot role of the angle to the operator. However the operator does have a lot to do with controlling the robot so as much as possible the camera position and angle should be automatic.

Chapter 9

How it all fits together

In this chapter I want to look at how the simulator and the application can be combined and used. In addition we should take a look at some of the loose ends that haven't been heavily covered in the previous chapters. Such topics as how to give control commands to the robot, and to some extent collision avoidance.

9.1 control

The best way to give control commands to the robot would probably be to send reference values for where you want the position of the joints to be.

The alternative of controlling the speed of the robot does not seem to be a good choice. If we did this we would have to have a very good dynamic model of the robot for it to be remotely possible for the simulation to be in sync with the real robot. most likely the position of the virtual model and the real one would diverge fairly quickly.

There might be some way of recalibrating the simulation with real robot from state vectors being sent back to the simulator. This does however seem to be a difficult problem. The Extended Kalman filter does come to mind, however it is hard to say if it would be able to deal with the uncertainties in both the model and the time delay between the simulation and the real robot.

Sending reference values for changes in position seams like a far better option. In such a case you can expect that if both the simulator and the real robot are well regulated in the sense that they will both reach the desired reference value, you would expect the two systems to settle in to the same position every time the operator stops to look at something from the video feed.

It might of course still be desirable to get some corrections to the simulations, however how this could be done I haven't looked at closely.

If both systems are able to track the same path in space you could further do

all the collision avoidance on the simulator end of the "chain".

9.2 Collision avoidance

Collision avoidance is going to be a crucial part of the final product. While the Interface described her can be of grate help when it comes to helping the operator with his situational awareness and navigating the robot It cant guarantee that no collisions can happen. If the arm collides with the surroundings or with it self this can cause damage to both the robot and its environment. This is naturally something we want to avoid.

Setting all the responsibility of not colliding the robot on the operator is not a viable option. We must then find a way to avoid collisions.

I have not looked very hard into this problem this semester. I did some work on it in my pre-project but not a lot. There has however recently a different master thesis that has considered this in a similar setting to the one described here. In that thesis [1] Kristoffer Aasland, investigates the possibility of using a 9 DoF robot arm system to do maintenance on a oil platform.

Unlike this thesis though, the main objective is to check for a solution to the collision avoidance problem. And path planing. The report has tested and concluded that the method used (The Probabilistic Roadmap method) "is a suitable approach to obtain 3D path planner with collision avoidance for the 9 DOF robot manipulator".

It might be possible to use this method on a computer close to the operator station by sending the safe paths to the robot inside the nacelle as well as the simulator. The Advantage of doing it this way is that you don't have to invest in a fast enough computer to be placed inside each of the nacelles where the robot system is installed.

If this option is not suited it might be possible to use a sensor "skin" method. Some approaches to this is described in my pre-project report. If you do this you do want the robot simulation to also reflect the constrictions that the real robot has from the sensor skin. This might be possible to do by using the collision detection method in the 3D model. If you attach some collisions spheres on the arm this can detect when you collide with the environment, when this happens an event is registered. If you handle this event as the sensors on the real robot headless sensing that it is close to an object inside the nacelle, this may let the simulated arm would act as the real one does. I don't however know if this is a viable option. The collision avoidance system should work though.

It might also be interesting to use the Phantom Omni controller (or one of the companies other controllers) as shown in figure 5.1,(or one of the companies other higher DoF controllers) to control the system.

9.3 Using the work I have started on

Aside from going ahead and making all the things like an arm a collision avoidance system and and integrating it with this system there are some other alternatives that can be explored.

You could set up the system to do certain tests, to see how well or bad the system performs. It would be possible to do testing using only simulations. You could set up two real time simulators of the same system and then set up a buffer between these two systems to simulate time delay. one of the simulations can be rendered in the application with out time delay and one can be rendered in place of the video feed with an identical environment and robot as in the other display regions, but with the view the camera at the end of the robot has. This would allow you to test how well the system performs, especially if you set up one scenario where you control the robot with time delay and only the rendered video feed.

The IRB 140 robot arm that has been used in the simulations is a robot that the Industrial Robotics Lab at NTNU owns. It might be possible to set up a test scenario with that robot, or one of the others they have. The material in this report should cover most of what is needed to setup the interface.

In the second test scenario it might also be possible to "simulate" translation in the robot by setting up an environment to navigate trough, on one or more conveyor belts, if it is hard to make that robot move.

Chapter 10

Conclusion

Robotic control in a telepresence system is a difficult task for an operator to perform. This report has shown how you can make a predictive display using a game engine. This predictive display should be able to help in alleviating some of the problems you face in telepresence.

With this report it should be possible to continue work, on using an arm in this project. One of the major achievements in this report is concerned with how you can take 3D models created in cad programs, in to a game engine for real time rendering. The Panda3D game engine seemingly have all the features you need to create a user interface that is capable of creating the predictive display.

The method for converting the models is not all that time consuming when you have gotten used to using the tools. This makes the approach of using cad models a viable solution.

The game engine Panda3D is easy to learn and don't require much from the programmer to give good results in return. From my own experience I would recommend continued use of this to develop the predictive display.

The report also shows you that you can use Simulink to create a real time simulation of a six DoF robot arm. In addition this simulation has been shown to run on available and affordable hardware. Solutions has been suggested for how you can proceed with extending the model, and how this might effect the simulations.

The report also describes how you might extend the system. It also shows that to get a workable solution you must combine solutions from many different disciplines.

My own limited testing of the application has convinced me that having a view of the robot arm that does not come from a camera mounted on the tip of the arm, should be part of the system.

The predictive display should be part of a system using a robotic arm in a telepresence system.

10.1 Future work

As far as future work is concerned what you want to do, depends on what steps you want to take next. However there are some things that should be done no matter what steps are taken. Whether that be testing the display with simulations or testing it with the IRB 140.

The automatic camera control in the predictive display part of the application, should be improved. The solution should allow the camera to move along objects in the scene, and not trough them. It should also be possible to have some manual control over the camera.

How the user interface should be, must be decided. Different ones could be tested with the system to give an indication on what would be the best.

The system models and the regulator should probably be extended, and tested to check for real time simulation capabilities. It might also be interesting to compare the Simulink model and code, with one created in Dymola for performance and stability.

Making a test scenario for the system might also be nice, this would probably depend upon the ability to make new environment 3D models. This modeling does however not have to be done with a CAD tool. A test scenario could also include setting up a real scenario with the IRB 140.

It must be decided what collision avoidance strategy would be the most beneficial. This should then be implemented and tested. This must be combined with a good human machine interface to allow the operator good control over the arm.

Bibliography

[1] Kristoffer Aasland. Optimal 3d path planning for a 9 dof robot manipulator with collision avoidance. Master's thesis, NTNU, 2008.

[2] Kevin Arthur. Effects of field of view on task performance with head-mounted displays. In *Conference companion on Human factors in computing systems: common ground*, CHI '96, pages 29–30, New York, NY, USA, 1996. ACM.

[3] `http://usa.autodesk.com/company/`.

[4] `http://en.wikipedia.org/wiki/Autodesk_Inventor`.

[5] A.K. Bejczy, W.S. Kim, and S.C. Venema. The phantom robot: predictive displays for teleoperation with time delay. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 546 –551 vol.1, may 1990.

[6] Eivind Berntsen. Prestudy into the applicability of using a robotic arm in an o&m telepresence system, operating inside a nacelle, 12 2011.

[7] `http://www.blender.org/`.

[8] Hung Bui. Remote presence inspection for off-shore wind turbines - background study and technology preparation, 12 2011.

[9] Peter Caselitz and Jochen Giebhardt. Rotor condition monitoring for improved operational safety of offshore wind energy converters. *Journal of Solar Energy Engineering*, 127(2):253–261, 2005.

[10] J. Casper and R.R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33(3):367 – 385, june 2003.

[11] J.Y.C. Chen, E.C. Haas, and M.J. Barnes. Human performance issues and user interface design for teleoperated robots. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(6):1231 –1245, nov. 2007.

[12] Zhenyuan Deng and M. Jagersand. Predictive display system for telemanipulation using image-based modeling and rendering. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2797 – 2802 vol.3, oct. 2003.

[13] W.E. Dixon, D. Moses, I.D. Walker, and D.M. Dawson. A simulink-based robotic toolkit for simulation and control of the puma 560 robot manipulator. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 4, pages 2202 –2207 vol.4, 2001.

[14] John V. Draper, David B. Kaber, and John M. Usher. Telepresence. *Human factors*, 40(3):354–375, 1998.

[15] Olav Egeland and Jan Tommy Gravdahl. *Modeling and Simulation for Automatic Control*. Marine cybernetics AS, 2002.

[16] N. Elkmann, T. Felsch, and T. Fo andrster. Robot for rotor blade inspection. In *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*, pages 1 –5, oct. 2010.

[17] Viktor Fidje. Remote presence on offshore wind turbines. Master's thesis, NTNU, 2010.

[18] Herman Hoifodt. Dynamic modeling and simulation of robot manipulators: The newton-euler formulation. Master's thesis, NTNU, June 2011.

[19] T. Itoh, Y. Suzuki, and T. Matsui. Alternative predictive display method of motion and force information without using environment model - design of real-image-based network teleoperation system -. In *Control Applications, 2005. CCA 2005. Proceedings of 2005 IEEE Conference on*, pages 314 –321, aug. 2005.

[20] Tor Mehlum Karlsen. Remote presence for offshore wind turbine. Master's thesis, NTNU, 2011.

[21] Hassan K. Khalil. *Nonlinear Systems (3rd Edition)*. Prentice Hall, 3 edition, December 2001.

[22] Abderrahmane Kheddar, Ee-Sian Neo, Riichiro Tadakuma, and Kazuhito Yokoi. Enhanced teleoperation through virtual reality techniques. In Manuel Ferre, Martin Buss, Rafael Aracil, Claudio Melchiorri, and Carlos Balaguer, editors, *Advances in Telerobotics*, volume 31 of *Springer Tracts in Advanced Robotics*, pages 139–159. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-71364-7_10.

[23] `http://www.microsoft.com/download/en/details.aspx?id=27876`.

[24] Bin Lu, Yaoyu Li, Xin Wu, and Zhongzhou Yang. A review of recent advances in wind turbine condition monitoring and fault diagnosis. In *Power Electronics and Machines in Wind Applications, 2009. PEMWA 2009. IEEE*, pages 1 –7, june 2009.

[25] G. Magnani, G. Magnani, P. Porrati, G. Rizzi, P. Rocco, and A. Rusconi. Real-time simulation of a space robotic arm. In *Workshop on robot simulators at the IEEE/RSJ International Conference on Intelligent RObots and Systems*, September 2008.

[26] R.C. Mann, W.R. Hamel, and C.R. Weisbin. The development of an intelligent nuclear maintenance robot. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 621 –623 vol.1, apr 1988.

[27] `http://www.maplesoft.com/products/maple/`.

[28] `http://www.mathworks.se/products/matlab/`.

[29] M.V. Noyes and T.B. Sheridan. A novel predictor for telemanipulation through a time delay. In *Annual Conference Manual Control, MoïňĂett Field, CA*, 1984.

[30] `http://www.panda3d.org`.

[31] `http://pandaboard.org/`.

[32] `http://www.sensable.com/haptic-phantom-omni.htm`.

[33] `https://www.rtai.org/`.

[34] P.A. Tipler and G. Mosca. *Physics for Scientists and Engineers*. Number v. 1 in Physics for Scientists and Engineers. W.H. Freeman, 2003.

[35] Y. Tsumaki and M. Yokohama. Predictive motion display for acceleration based teleoperation. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2927 –2932, may 2006.

[36] Edwin Wiggelinkhuizen, Theo Verbruggen, Henk Braam, Luc Rademakers, Jianping Xiang, and Simon Watson. Assessment of condition monitoring techniques for offshore wind farms. *Journal of Solar Energy Engineering*, 130(3):031004, 2008.

[37] Bob G. Witmer and Michael J. Singer. Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoperators and Virtual Environments*, 1998.

[38] David D. Woods. Visual momentum: a concept to improve the cognitive coupling of person and computer. *International Journal of Man-Machine Studies*, 21(3):229 – 244, 1984.

Appendix A

Panda code

Understanding the code written in Panda3D is hard without having any prior knowledge about panda3D or constantly having to reference the documentation. It would be almost impossible to write comments describing or explaining every line of code without making the code close to unreadable, and certainly messy.

For that reason I will first present the code used here and then write an explanation for some of the less obvious facts. In addition I will try to give some pointers to things that might help in future developments of the code.

A.1 The code

```
from panda3d.core import loadPrcFileData
loadPrcFileData("",
"""
win-origin 64 25
win-size 1855 1050
sync-video 0
""")

from direct.showbase.ShowBase import ShowBase
from pandac.PandaModules import *
from direct.actor.Actor import Actor
from direct.directutil.Mopath import Mopath
from direct.interval.IntervalGlobal import *
import sys

from camOb import *
```

```
class InspectorGadget(ShowBase):

    def __init__(self):
        self.roboPos=3000
        ShowBase.__init__(self)
        base.cam.node().getDisplayRegion(0).setActive(0)
        base.setFrameRateMeter(True)
        #self.render.setShaderAuto()
        #render.setAntialias(AntialiasAttrib.MAuto)
        taskMgr.add(self.movement,"moveTask")
        base.disableMouse()

        self.setupMovie()
        self.eventSetup()
        self.camP1 = camOb(0.005, 0.3, 0.505, 0.995)
        self.camP2 = camOb(0.005, 0.3, 0.005, 0.495)
        self.cam1 = self.makeNewDrAndCam('cam1')
        self.cam2 = self.makeNewDrAndCam('cam2')
        self.setupEnv()
        self.setupRobo()
        self.setupMopath()
        self.mopath.goTo(self.robo, self.roboPos)
        self.setupCD()
        self.splitScreen()
        render.analyze()

        self.campos = 1
        self.cam1.setPos(self.robo.getX()+700,self.robo.
            getY(),self.robo.getZ()-75)


    def windowAspectEventHandler(self):
        """
        This method is evocked by the eventhnadler when the
            size of the window changes.
        It curently makes sertain that the relative
            dimensions of the model in
        the two 3D renderings are corect and is the same
            when the aspect ratio changes.
        """
```

```
        self.cam1.node().getLens().setAspectRatio(self.
            camP1.getAsp(base.win))
        self.cam2.node().getLens().setAspectRatio(self.
            camP2.getAsp(base.win))

    def setKey(self, key, value):
        self.keyMap[key] = value

    def eventSetup(self):
        """
        This function is used to set up events and register
            event handlers for these events.
        """
        self.keyMap = {"swivel-left":0, "swivel-right":0, "
            forward":0, "humerus-left":0, "humerus-right":0,
             "ulna-right":0, "ulna-left":0, "backward":0, "
            toggle":0}
        self.accept('aspectRatioChanged', self.
            windowAspectEventHandler)
        self.accept("escape", sys.exit)
        self.accept("t", self.eventToggleCamPos)
        self.accept("arrow_left", self.setKey, ["swivel-
            left",1])
        self.accept("arrow_right", self.setKey, ["swivel-
            right",1])
        self.accept("arrow_up", self.setKey, ["forward",1])
        self.accept("arrow_down", self.setKey, ["backward"
            ,1])
        self.accept("a", self.setKey, ["humerus-left",1])
        self.accept("s", self.setKey, ["humerus-right",1])
        self.accept("z", self.setKey, ["ulna-left",1])
        self.accept("x", self.setKey, ["ulna-right",1])
        self.accept("arrow_left-up", self.setKey, ["swivel-
            left",0])
        self.accept("arrow_right-up", self.setKey, ["swivel
            -right",0])
        self.accept("arrow_up-up", self.setKey, ["forward"
            ,0])
        self.accept("arrow_down-up", self.setKey, ["
```

```
        backward",0])
    self.accept("a-up", self.setKey, ["humerus-left"
        ,0])
    self.accept("s-up", self.setKey, ["humerus-right"
        ,0])
    self.accept("z-up", self.setKey, ["ulna-left",0])
    self.accept("x-up", self.setKey, ["ulna-right",0])


def eventToggleCamPos(self):
    """
    This function is used to change the camera position
        and angle of the map view camera.
    """
    if (self.campos):
        self.cam2.setPos(0, 3500, 400)
        self.cam2.setH(180)
        self.campos = 0
    else :
        self.cam2.setPos(0, -3500, 400)
        self.cam2.setH(0)
        self.campos = 1

def setupEnv(self):
    """
    This function initialises the environment and the 3
       D modeles.
    """
    self.nacelle = self.loader.loadModel("models/
       Nacelle3.egg")
    self.nacelle.reparentTo(render)
    self.nacelle.setPos(0,0,0)
    self.nacelle.flattenLight()
    ambientLight = AmbientLight("ambientLight")
    ambientLight.setColor(Vec4(.3, .3, .3, 1))
    plight = PointLight('plight')
    plight.setColor(VBase4(1, 1, 1, 1))
    plight.setShadowCaster(True, 512, 512)
    plnp = render.attachNewNode(plight)
    plnp.setPos(0, -850, 0)
```

```
        render.setLight(plnp)

        render.setLight(render.attachNewNode(ambientLight))


    def setupRobo(self):
        """
        This function loads the robot in to teh sene graph
            so that it can be rendered.
        It finds the joints in the model and makes them
            avialbel to be manipulated.
        """
        self.robo = Actor("models/robo2.egg")
        self.robo.reparentTo(render)
        self.humerus = self.robo.controlJoint(None, '
            modelRoot', 'jointFirstArm')
        self.ulna = self.robo.controlJoint(None, 'modelRoot
            ', 'jointSecondArm')
        self.swivel = self.robo.controlJoint(None, '
            modelRoot', 'jointSwivel')
        self.Camera = self.robo.controlJoint(None, '
            modelRoot', 'jointCamera')


    def setupMopath(self):
        """
        Here the motion path is loade up from the egg file
            and made avilable as a Mopath object.
        """
        self.mopath = Mopath()
        self.mopath.fFaceForward = 1
        self.mopath.loadFile("models/Nacelle3.egg")
        self.mopathLengt=self.mopath.getMaxT()

    def setupCD(self):
        base.cTrav = CollisionTraverser()
        base.cTrav.showCollisions(render)
        self.notifier = CollisionHandlerEvent()
        #self.notifier.addInPattern("%fn-in-%in")
        #self.accept("frowney-in-floor", self.onCollision)
```

```
def setupMovie(self):
    """
    This function setsup the video and starts palying
        it.
    """
    tex = MovieTexture("name")
    assert tex.read("mov/PandaSneezesMjpeg.avi"), "
        Failed to load video!"
    cm = CardMaker("Panda sneese card");
    cm.setFrame(-0.39, 0.99, -0.2, 0.99)
    cm.setUvRange(tex)
    card = NodePath(cm.generate())
    card.reparentTo(render2d)
    card.setTexture(tex)
    card.setTexScale(TextureStage.getDefault(), tex.
        getTexScale())
    tex.play()


def makeNewDrAndCam(self, camName):
    dr = base.win.makeDisplayRegion()
    dr.setClearColor(VBase4(0, 0, 0, 1))
    dr.setClearColorActive(True)
    dr.setClearDepthActive(True)
    cam1 = render.attachNewNode(Camera(camName))
    dr.setCamera(cam1)
    cam1.setPos(0, -3500, 400)
    return cam1


def splitScreen(self):
    dr1 = self.cam1.node().getDisplayRegion(0)
    dr2 = self.cam2.node().getDisplayRegion(0)
    dr1.setDimensions(self.camP1.x1, self.camP1.x2,
        self.camP1.y1, self.camP1.y2)
    dr2.setDimensions(self.camP2.x1, self.camP2.x2,
        self.camP2.y1, self.camP2.y2)
    self.cam1.node().getLens().setAspectRatio(self.
        camP1.getAsp(base.win))
```

```
        self.cam2.node().getLens().setAspectRatio(self.
            camP2.getAsp(base.win))


    def movement(self, task):
        self.cam1.lookAt(self.robo)
        if (self.keyMap["humerus-left"]!=0):
            self.humerus.setR(self.humerus.getR() + 75 *
                globalClock.getDt())
        if (self.keyMap["humerus-right"]!=0):
            self.humerus.setR(self.humerus.getR() - 75 *
                globalClock.getDt())
        if (self.keyMap["ulna-left"]!=0):
            self.ulna.setR(self.ulna.getR() + 75 *
                globalClock.getDt())
        if (self.keyMap["ulna-right"]!=0):
            self.ulna.setR(self.ulna.getR() - 75 *
                globalClock.getDt())
        if (self.keyMap["swivel-right"]!=0):
            self.swivel.setHpr(270,self.swivel.getP() + 75
                * globalClock.getDt(),0)
        if (self.keyMap["swivel-left"]!=0):
            self.swivel.setHpr(270,self.swivel.getP() - 75
                * globalClock.getDt(),0)

        if (self.keyMap["forward"]!=0):
            self.roboPos=(self.roboPos-50 * globalClock.
                getDt())%self.mopathLengt
            self.mopath.goTo(self.robo, self.roboPos)
        if (self.keyMap["backward"]!=0):
            self.roboPos=(self.roboPos+50 * globalClock.
                getDt())%self.mopathLengt
            self.mopath.goTo(self.robo, self.roboPos)


        camvec = self.robo.getPos() - self.cam1.getPos()
        camdist = camvec.length()
        camvec.normalize()

        if (camdist > 400.0):
```

```
            res = self.cam1.getPos() + camvec*1
            res.setZ(self.robo.getZ()-75)
            self.cam1.setPos(res)
            camdist = 170.0
        if (camdist < 50.0):
            res = self.cam1.getPos() + camvec*1
            res.setZ(self.robo.getZ()-75)
            self.cam1.setPos(res)

        return task.cont


m = InspectorGadget()

run()
```

A.2 Explaining the code

The first lines seven lines of code are there to modify some of the standard variables in Panda3D. You can notice Thoth the window size is set and also the placement. The "sync-video 0" line makes it so that the rendering can go faster than the refresh rate of the monitor this can be beneficial while testing the model as you get somewhat of a measure of how much the graphics card is taxed.

A.2.1 Inspector Gadget

The InspectorGadget class represent the main body of code in the app. The object is created just before the program starts to run. It inherits from the ShowBase class. This is of importance as the ShowBase class, and its initialization gives access to quite a lot of the functionality of Panda3D If you did not inherit from this class you would have to do quit a lot in therms of setup. The class is over 2000 lines long.

Initialization

The next line of interest is the "base.cam.node().getDisplayRegion(0).setActive(0)" line here we deactivate the camera that is associated with the display region created by ShowBase during its initialization. The reason this must be done is that this camera is meant to be used with a display region that covers the whole window. The way this manifests it self is that if the display region changes its aspect ratio compared to the window the rendered 3D models will eider be stretched or

squashed depending on witch way the aspect ratio is changed. This is not mentioned anywhere in the manual and the tutorial code for creating multiple display regions does not mention anything about this.

Next there are some lines of code that has been removed that sets anti aliasing effects on the models. This did not render all the parts of the model well so it has been removed.

The "taskMgr.add(self.movement,"moveTask")" statement adds the movement function as a task that Panda3D will execute before drawing a new frame.

The "base.disableMouse()" statement makes it so that th mouse does not have any impact on the camera angles or positions in the scene.

The next lines in the init function are mostly concerned with setting up two new cameras to be used in rendering the two display regions that are created and given a position in the coordinate system of the window. The environment models are loaded up as well as the robot. The motion path is also loaded up and created as an object. After that the robot is positioned on the a spot on the motion path. Then the camera of the display region of the predictive display is initialized. After this initializing the system is done and it is ready to run.

Explanation on some functions

The windowAspectEventHandler, eventSetup, setKey, eventToggleCamPos shouldn't be so hard to understand. They all involve setting up events and event handlers.

The setupEnv, setupRobo and setup setupMopath functions shouldn't be so hard to understand either they load up the models and set some lighting effects as you need light so the models can be rendered. There are however more lighting options available to be used and I cant say that the lighting that has been set up here is th optimal one.

The setupCD function is an attempt I had to setup collision detection for the system. this I never completed.

The setupMovie function is responsible for creating the area of the screen where the movie is rendered. Some of what is used here would probably also have to be used, if you were to set up support for the webcam video stream. You are likely going to have to create a separate thread to handle the connection and perhaps also some unpacking of the video stream. To do this you should not use the threading functionality of Python.

Panda3D uses the threading functionality of C++. Photon threads can be used in some cases, however it is likely to be problematic. Panda3D does however let you create threads trough their own interface. This is the recommended method for creating threads in Panda3D. To get more information on this you should look in the Panda3D manual.

The makeNewDrAndCam and the splitScreen functions are used to create the new cameras and the display regions. The splitScrean function is responsible for setting the size and position of the display regions.

If you look in the splitScrean function three is a cal to getLense. This returns the lens of the camera you cal. This lens has several features you can change to effect how the scene it captured and rendered. you can control the lens like you would an ordinary camera lens or trough simpler means. some of the things you can so is control the field of vision of the lens so that you can capture more of a scene.

The last function is the movement function this is responsible for moving the robot when the keys are pressed and making the camera follow the robot. If we were to get the position states from the simulations this is where you would use them to animate the virtual arm.

Appendix B

Software

In This section I want to cover some of the quirks I have struggled with when it comes to the software used In this project. This is done in the hope that things will go smother if the software is used in future developments.

As I have used Panda3D [30] to render the 3D model there are certain file formats that are compatible and certain tolls that can be used in conjunction with Panda3D. Panda3D uses its own file format to load the geometry data up to the graphics card, namely .bam files. These files are created from a different format namely .egg files that is also specific to panda. The bam files only works with the Panda3D version that converted them. The .egg files are not specific to one version but should work on most others to. When you want to develop new content to use in the game engine, you want to convert what ever format you have into .egg files and then let the .bam generation take place on startup of your program.

There are several Python scripts written to convert different kinds of file formats with geometries data into egg files. Many off these scripts comes bundled in the Panda3D installation[1], in table B.1 the egger scripts that comes with Panda3D can be seen.

dae2egg	maya2egg7	maya2egg2009
dxf2egg	maya2egg8	maya2egg2010
flt2egg	maya2egg65	maya2egg2011
lwo2egg	maya2egg85	x2egg
maya2egg6	maya2egg2008	VRML2egg

Table B.1: Scripts bundled in Panda for egg conversion

[1]These files can be found in the bin directory of the Panda3D installation

Appendix C

Equations of motion

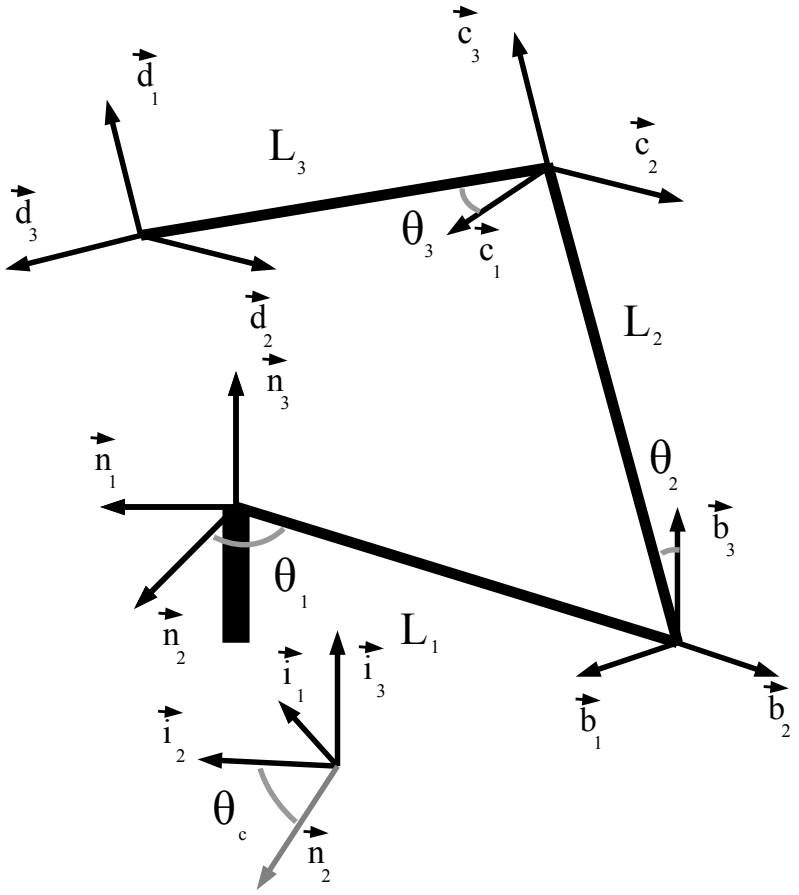

Figure C.1: Coordinate frames used to describe the robot and cart. (Gravity acts upwards in the direction of $\vec{i_3}$)

To find the equations of motion I am going to use Lagrange's method as described in [15]. We then need to find the Lagrangian. The Lagrangian is described as follows:

$$L(q, \dot{q}, t) = T(q, \dot{q}, t) - U(q) \tag{C.1}$$

Where q is the vector of generalized coordinates. T is the kinetic energy of the system and U is the potential energy of the system.

To develop the model we must first define our generalized coordinates. We do not yet know how the rails will be placed inside the nacelle and what configuration it will have. While it is likely that the rails will have to dip down or pull upwards I will here constrain myself to modeling the rails by letting them only move in the horizontal plane. Thus the robot and cart will be able to move in the x and y direction and in the bends there will be a momentum on the robot about it's central z axis. This gives us the angle of rotation θ_c about the z axis, x and y as generalized coordinates from the fact that the whole robot moves along the rail.

It does not stop there of course. We must also model the robots joints, the base and the end effector joints. I will base the model upon the arm that I modeled in Inventor.

The first generalized coordinate we get from the robot is the angle of rotation θ_b between the base of the robot and the cart. Then we have one angle between the base and the first arm θ_{a1} and a second angle between the first and second arm θ_{a2}. We must also take the end effectors joints in to our model, as it has three degrees of freedom we need three generalized coordinates to model it's motion. The end effector has roll θ_r pitch θ_p and yaw θ_{ya} angels that need to be included as generalized coordinates. The vector of generalized coordinates then consist of these nine variables:

$$q = |x, y, \theta_c, \theta_b, \theta_{a1}, \theta_{a2}, \theta_r, \theta_p, \theta_{ya}| \tag{C.2}$$

As can be seen this models the rail as only allowing the cart to move in the horizontal plane. and in these coordinates I have taken in to account the generalized coordinates describing the position of the end effector. To make tings a little simpler I will first develop the equations of motion for the system without taking the end effector into account. giving us this vector fro the generalized coordinates.

$$q = |x, y, \theta_c, \theta_b, \theta_{a1}, \theta_{a2}| \tag{C.3}$$

The T in C.1 is the kinetic energy of the system, where as the U in C.1. We then need to find The kinetic energy of the system and the potential energy. The potential energy is relatively simple to find. In [34] we can find this equation for potential energy:

$$U = mhg \tag{C.4}$$

Where m is the mass of the object, h is the distance from the highest point of the center of mass of an object to it's lowest point, and g is the acceleration of gravity. To model the robotic arm I will partition the robot into many masses as listed here:

m_c	:	mass of the cart.
m_b	:	mass of the swivel base.
m_{a1}	:	mass of the first arm link.
m_j	:	mass of the joint between the two links.
m_{a2}	:	mass of the second arm link.
m_e	:	mass of the end effector.

Table C.1: The masses of the system.

When it comes to potential energy m_c and m_b does not ad to it. m_{a1}, m_j, m_{a2} and m_e does. The equations for m_{a1} and m_j is very similar and we only need to derive one equation for them. The same is true for m_{a2} and m_e but they share the same equation.

For m_{a1} and m_j the equation is as in C.5, while for m_{a2} and m_e the equations are as in equation C.6 m_{a1} and m_e. You naturally have to change M to be the corresponding mass and L_1 and L_2 will have to be altered so that they represent the correct distance to the center of mass.

$$gML_2(1 - \cos\theta_2) \tag{C.5}$$

$$gM(L_2(1 - \cos\theta_2) + L_3(1 - \sin\theta_3) \tag{C.6}$$

To find the kinetic energy of the system we use the equation (C.7) as found in [34].

$$K = \frac{1}{2}I\dot{\theta}^2 + \frac{1}{2}m\vec{v}^2 \tag{C.7}$$

for our system we must sum up the contribution to the total kinetic energy from all the masses. We can start with finding the position of all the masses based upon the position of the robot and the angles of each of the joints.

$$\vec{r_2} = x\vec{i_1} + y\vec{i_2} + L_z\vec{i_3} \tag{C.8}$$

Where L_z is a constant distance that describes the height of the two masses and x and y describes the position.

To describe the position of a center of mass along the first arm link we can use equation C.9. This equation also be made to describes the position of the center of mass of the joint by changing the two constants L_1 and L_2.

$$\vec{r_2} = x\vec{i_1} + y\vec{i_2} + L_1\vec{b_2} + L_2\vec{c_3} \tag{C.9}$$

Equation C.10 can be used to describe any center of mass along L_3 and thus it can also describe the center of mass of the end effector.

$$\vec{r_2} = x\vec{i_1} + y\vec{i_2} + L_1\vec{b_2} + L_2\vec{c_3} + L_3\vec{d_3} \tag{C.10}$$

The relation between the different coordinate frames can be found as in C.11. From these relations we can find the position of the centers of mass in the ground reference i.

$$\begin{aligned}
\vec{n}_1 &= \vec{i}_1\cos\theta_c + \vec{i}_2\sin\theta_c \\
\vec{n}_2 &= -\vec{i}_1\sin\theta_c + \vec{i}_2\cos\theta_c \\
\vec{n}_3 &= \vec{i}_3 = \vec{b_3} \\
\vec{b}_1 &= \vec{n}_1\cos\theta_1 + \vec{n}_2\sin\theta_1 \\
\vec{b}_2 &= -\vec{n}_1\sin\theta_1 + \vec{n}_2\cos\theta_1 = \vec{d_2} = \vec{c_2} \\
\vec{b}_3 &= \vec{n}_3 = \vec{i}_3 \\
\vec{c}_1 &= \vec{b}_1\cos\theta_2 - \vec{b}_3\sin\theta_2 \\
\vec{c}_2 &= \vec{n}_3 = \vec{i}_3 \\
\vec{c}_3 &= \vec{b}_1\sin\theta_2 + \vec{b}_3\cos\theta_2 \\
\vec{d}_1 &= -\vec{c}_1\sin\theta_3 + \vec{c}_3\cos\theta_3 \\
\vec{d}_2 &= \vec{n}_3 = \vec{i}_3 \\
\vec{d}_3 &= \vec{c}_1\cos\theta_3 + \vec{c}_3\sin\theta_3
\end{aligned} \tag{C.11}$$

Now we can find this relation for $\vec{b}_2$, $\vec{c}_3$ and $\vec{d}_3$:

$$\begin{aligned}
\vec{b}_2 &= \vec{i}_1 \cos\theta_c \sin\theta_1 + \vec{i_2} \sin\theta_c \sin\theta_1 - \vec{i}_1 \sin\theta_c \cos\theta_1 + \vec{i}_2 \cos\theta_c \cos\theta_1 \\
\\
\vec{c}_3 &= \vec{i}_1 \cos\theta_c \cos\theta_1 \cos\theta_2 + \vec{i_2} \sin\theta_c \cos\theta_1 \cos\theta_2 \\
&\quad -\vec{i}_1 \sin\theta_c \sin\theta_1 \cos\theta_2 + \vec{i_2} \cos\theta_c \sin\theta_1 \cos\theta_2 + \vec{i}_3 \sin\theta_2 \\
\\
\vec{d}_3 &= \vec{i}_1 \cos\theta_c \cos\theta_1 \cos\theta_2 \cos\theta_3 + \vec{i}_2 \sin\theta_c \cos\theta_1 \cos\theta_2 \cos\theta_3 \\
&\quad -\vec{i}_1 \sin\theta_c \sin\theta_1 \cos\theta_2 \cos\theta_3 + \vec{i}_2 \cos\theta_c \sin\theta_1 \cos\theta_2 \cos\theta_3 \\
&\quad -\vec{i}_3 \sin\theta_2 \cos\theta_3 \\
&\quad +\vec{i}_1 \cos\theta_c \cos\theta_1 \sin\theta_2 \sin\theta_3 + \vec{i}_2 \sin\theta_c \cos\theta_1 \sin\theta_2 \sin\theta_3 \\
&\quad -\vec{i}_1 \sin\theta_c \sin\theta_1 \sin\theta_2 \sin\theta_3 + \vec{i}_2 \cos\theta_c \sin\theta_1 \sin\theta_2 \sin\theta_3 \\
&\quad +\vec{i}_3 \cos\theta_2 \sin\theta_3 \qquad \text{(C.12)}
\end{aligned}$$

Combining this with C.8, C.9 and C.10 yields these vectors for the position:

$$\vec{r}_1 = [x, y, 0]$$

$$\begin{aligned}
\vec{r}_2 &= [x + L_1 \cos\theta_c \sin\theta_1 - L_1 \sin\theta_c \cos\theta_1 + L_2 \cos\theta_c \cos\theta_1 \cos\theta_2 - L_2 \sin\theta_c \sin\theta_1 \cos\theta_2, \\
&\quad y + L_1 \sin\theta_c \sin\theta_1 + L_1 \cos\theta_c \cos\theta_1 + L_2 \sin\theta_c \cos\theta_1 \cos\theta_2 + L_2 \cos\theta_c \sin\theta_1 \cos\theta_2, \\
&\quad L_2 \sin\theta_2]
\end{aligned}$$

$$\begin{aligned}
\vec{r}_3 &= [x + L_1 \cos\theta_c \sin\theta_1 - L_1 \sin\theta_c \cos\theta_1 + L_2 \cos\theta_c \cos\theta_1 \cos\theta_2 - L_2 \sin\theta_c \sin\theta_1 \cos\theta_2 \\
&\quad +L_3 \cos\theta_c \cos\theta_1 \cos\theta_2 \cos\theta_3 - L_3 \sin\theta_c \sin\theta_1 \cos\theta_2 \cos\theta_3 + L_3 \cos\theta_c \cos\theta_1 \sin\theta_2 \sin\theta_3 \\
&\quad -L_3 \sin\theta_c \sin\theta_1 \sin\theta_2 \sin\theta_3, \\
&\quad y + L_1 \sin\theta_c \sin\theta_1 + L_1 \cos\theta_c \cos\theta_1 + L_2 \sin\theta_c \cos\theta_1 \cos\theta_2 + L_2 \cos\theta_c \sin\theta_1 \cos\theta_2 \\
&\quad +L_3 \sin\theta_c \cos\theta_1 \cos\theta_2 \cos\theta_3 + L_3 \cos\theta_c \sin\theta_1 \cos\theta_2 \cos\theta_3 + L_3 \sin\theta_c \cos\theta_1 \sin\theta_2 \sin\theta_3 \\
&\quad +L_3 \cos\theta_c \sin\theta_1 \sin\theta_2 \sin\theta_3, \\
&\quad L_2 \sin\theta_2 - L_3 \sin\theta_2 \cos\theta_3 + L_3 \cos\theta_2 \sin\theta_3] \qquad \text{(C.13)}
\end{aligned}$$

By differentiating C.13 we can find the speed of each of the mass centers.

C.1 Inertia

The inertia of the two links changes with respect to the angle of rotation between the base and the cart θ_1 when θ_2 and θ_3 changes. The inertia in the first link is

only affected by changes in θ_2 while the second links inertia is effected by both θ_2 and θ_3. In this section estimates of the equations describing this will be developed.

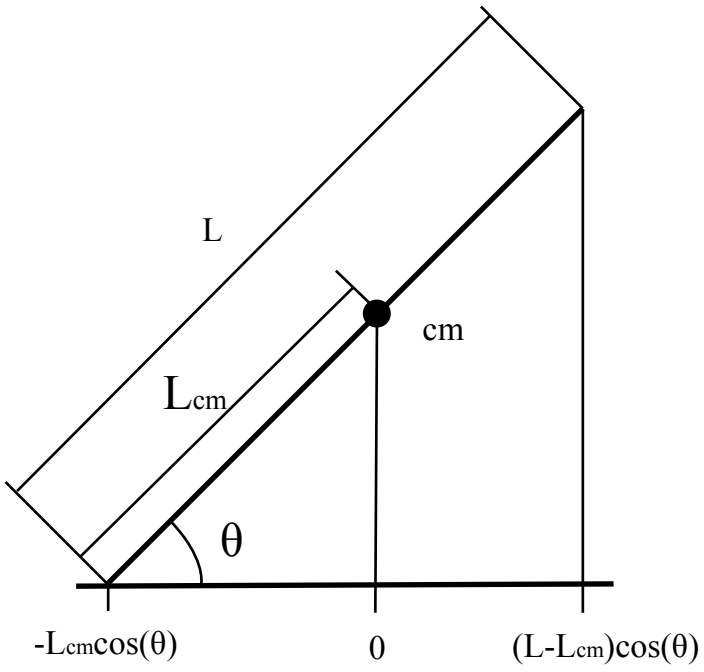

Figure C.2: Diagram for calculating the inertia

$$I = \int r^2 \mathrm{d}m$$

$$\mathrm{d}m = \frac{M}{|L\cos\theta|}\mathrm{d}r = \frac{M}{\sqrt{L^2\cos\theta^2}}\mathrm{d}r$$

$$I = \int_{-L_{cm}cos(\theta)}^{(L-L_{cm})cos\theta} \frac{M}{\sqrt{L^2\cos\theta^2}} r^2 \,\mathrm{d}r$$

Solving this equation we end up with:

$$I = \frac{M}{3}()\cos\theta$$

At this point I found that something very similar had been done and that the calculations and a Simulink model had already been set up. And so I decided to go with these instead. However I doe have some reflections when it comes to the model I have been deriving here.

The way I have modeled this robot I will end up with a larger set of equations than what has been developed in the other master thesis. This is despite the fact that both systems wold have had the same degrees of freedom. The reason for this is that I partition the segments of the robot into smaller parts. This I did to more accurately and easily calculate the inertia of the links. A larger equation set would mean more calculations in each time step of the simulator which might make it harder to get a small enough time step.

Simplifying this as has been done in the other master thesis might be necessary to run the simulator. Especially as needier of the modeling takes into account all the degrees of freedom in the actual system. When I model the system I don't handle the degrees of freedom in the end effector assuming that movement here don't affect the rest of the system to a large extent. Where as the model that I ended up using don't yet handle translation along the tracks or rotation of the cart.

Appendix D

The Attachments

In the attached zip file The 3D models created for use in this project are included. This includes Inventor models, the Maya models and the egg files that have been created.

The code for the game engine is naturally included. This can be executed and extended if you have the Panda3D SDK installed.

There is also an installer for the application. With this you don't have to install the SDK or the runtime environment to run the application.

The Simulink model of the robot arm is also included. In the same folder I have included the Maple script created by Høifødt and slightly modified by me. This script creates the equations of motion in addition to some of the code used in the model.

I have also included the scraped Maple files that I was working on. These are somewhat chaotic. The maple script created by Høifødt is a far better example of how it should be done.

I also included a Pdf of the report from the previous semester.

If you want to get your hands on the original Simulink model and Maple scripts created by Høifødt you should search them up in Daim.