Control of a model sized hovercraft

Citation for published version (APA):
Sanders, R. M. W. (2003). *Control of a model sized hovercraft*. (DCT rapporten; Vol. 2003.027). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/2003

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 25. mars. 2021

THE UNIVERSITY OF NEW SOUTH WALES
SYDNEY · AUSTRALIA

TU/e technische universiteit eindhoven

Control of a Model Sized Hovercraft

R.M.W. Sanders

DCT nr. 2003-27

Research report

Coach: prof. V. Solo

University of New South Wales
School of Electrical Engineering & Telecommunications
Systems & Control Research Group

Supervisor: prof.dr.ir.M.Steinbuch

University of Technology Eindhoven
Department of Mechanical Engineering
Control Systems Technology

Eindhoven, March 2003

Foreword

I would like to thank the school of Electrical Engineering & Telecommunications for giving us the opportunity to do our internship at the University of New South Wales. Especially Prof Victor Solo, head of Systems and Control Research group for his supports and valuable advises. I would like to thank also Chris Lu for his helps and discussions in the laboratory and also Dr. R Eaton and Dr. Clements for invaluable advises during our project. Besides the school of EE&T I would also thank the school of Mechanical Engineering for making it possible to do experiments at the the test rig of the department Mechatronics and especially Dr. J Katupitiya and Mr.C J Sanderson.
After all I look back at a great and valuble time at the University in Sydney and in Australia. It was a great experience, but the time to stay was to short.
This report contains the work that I have done with Bart Consten during the period of 9th September to 7th December for the hovercraft project at the Systems and Control Research laboratory at the UNSW.

Summary

The development of the hovercraft started with Sir Christopher Cockerell. He developed for the first time a 'vessel' that could float over land or water. Cockerell used an air cushion underneath the vessel and this cushion reduced the friction between the surface of the water and hulls so much because of an air-lubricated layer. The hovercraft in this project is a two channel AM remote controlled model hovercraft from *HOVERCRAFTmodels.com™*, type electro cruiser. Research purposes in future are making use of vision and control. But before this is possible has to be made a connection with the computer. The purposes for this project are creating an open loop PC controlled hovercraft and deriving a dynamical model for the hovercraft.
A few adjustments are made to the hardware and software to come to a PC controlled hovercraft. The remote control transmitter is first of all transformed to a PC compatible transmitter. Normal remote control is still possible. Therefore are attached two switches on top of the transmitter to switch between normal remote control mode and PC control mode. Remote control mode is active if both switches are pointing to the antenna of the transmitter and PC control model is active when the switches are pointing to the outside. Sending signals out of a PC requires a data acquisition board. The data acquisition board that is used during the project is from AXIOM Technology Co. Ltd, type AX5411H. A real-time environment is desired in control technology and in this project is used Real-time linux. Real-time Linux is a small real-time operating system and is used to get the Linux system under control to meet real-time computing needs. On the concerning computer is therefore installed linux kernel 2.2.18 and rt-linux kernel 3.0.
Control software is designed using an older version of a SISO program of Ray Eaton. The new program makes it possible to send two signals out at one time. Controlling of the hovercraft open loop works with four buttons in the designed Graphical User Interface. Two buttons Left and Right are for controlling the rudder and Up and Down are for controlling the lift motor. Current software and hardware make it possible to control the hovercraft open loop.
Deriving a dynamical model of the hovercraft is done with the Newton-Euler method. A 3 degrees of freedom model is formulated after making a few assumptions. In the model is the moment of inertia about the z-axis an unknown parameter and an experiment at the school of mechanical engineering is done to find out the value of the moment of inertia. A mathematical method is done to check the obtained value of the experiment. The mathematical value is less reliable, but nevertheless are both values lying within 10% of each other. From this result can be concluded that the empirical value is reliable.
Validation of the dynamical model is done as good as possible, because validation experiments require some free space with no obstacles. Validation experiments are done to get an indication of the characteristics of motion. Comparing the experiment results with the results from the simulation gives a reasonable idea about the correctness of the model. The hovercraft model is acceptable for future use. A point of improvement in the hovercraft model is the non-uniform mass dependency effects. It is not necessary to improve the model if a better mass distribution can be created in the new hovercraft.
Two things have to be done to implement vision and control in the future. First, a driver for the X10 Video USB adapter has to be designed and after that can be started with vision and designing a controller. Control purposes can be for example following a straight line or going to a marker on a wall.

Summary (Dutch)

De ontwikkeling van de hovercraft is begonnen met Sir Christopher Cockerell. Hij ontwikkelde voor het eerst een 'vaartuig', dat kon zweven over land of water. Cockerell maakte gebruik van een luchtkussen onder het vaartuig. Dit luchtkussen zorgt voor een aanzienlijke vermindering van wrijving tussen het wateroppervlak en de scheepsromp middels een gecreëerde luchtlaag. De hovercraft in dit project is een twee kanaals AM radio grafisch bestuurbare model hovercraft van *HOVERCRAFTmodels.com™*, type electro cruiser. Toekomstige onderzoeksdoeleinden zijn het toepassen van vision en control, maar voordat dat mogelijk is, zal eerst een aansluiting met te PC gemaakt moeten worden. De doelen van dit project zijn het open loop bestuurbaar maken van de hovercraft en het ontwerpen van een dynamisch model.

Om te komen tot een PC gestuurde hovercraft zijn een aantal aanpassingen gedaan zowel aan de hardware als software. Allereerst is de twee kanaals zender aangepast zodat of PC control of radio grafisch control mogelijk is. Hiervoor zijn twee schakelaars boven op de zender bevestigd. Indien de schakelaars beide naar de antenne van de zender wijzen is de radio grafisch control mode actief. PC control mode is ingeschakeld als beide schakelaars naar buiten wijzen. Om signalen te kunnen versturen naar de zender van de hovercraft is een data acquisitie bord ingebouwd in de computer. Het bord is van AXIOM Technology Co. Ltd., type AX5411H. In de regeltechniek is een real-time omgeving gewenst en in dit project is Real-time linux het besturingsprogramma dat ervoor zorgt dat de normale linux kernel real time wordt. Op de betreffende PC zijn daarom allereerst Linux 2.2.18 en Rt-linux 3.0 geinstalleerd.

De control software is vervolgens ontworpen aan de hand van een ouder SISO programma van Ray Eaton. Het nieuwe programma maakt het mogelijk om twee signalen uit te sturen. De besturing zelf gaat door op de gewenste knoppen te drukken in de ontworpen Graphical User Interface. Twee knoppen Left en Right zijn ter besturing van het roer en de knoppen Up en Down zijn voor het aansturen van de lift motor. De huidige software en hardware maken het mogelijk om de hovercraft open loop te besturen.

Het afleiden van een dynamisch model is gedaan met behulp van Newton-Euler. Na enkele vereenvoudigingen doorgevoerd te hebben is een model met 3 graden van vrijheid opgesteld. In het model was de massatraagheid van de hovercraft om de z-as een onbekende parameter. Om deze parameter te achterhalen is allereerst op empirische wijze bij de faculteit van werktuigbouwkunde een waarde bepaald en vervolgens met een analytische methode een controle check uitgevoerd. De analytische uitkomst is echter minder betrouwbaar vanwege meerdere afschattingen, maar desalniettemin liggen beide waarden niet meer dan 10% uit elkaar. De experimenteel bepaalde waarde voor de massatraagheid kan als betrouwbaar verondersteld worden.

Validatie van het model is zo goed als mogelijk gedaan in verband met de beperkte test ruimte. Uit de validatie experimenten kan toch enig inzicht verkregen worden in de bewegingskarakteristieken van de hovercraft. En door deze karakteristieken te vergelijken met de simulatie kan geconcludeerd worden dat het model voldoet. Echter de niet-uniforme massa verdeling binnen de hovercraft zorgt bij lage lift krachten voor meer wrijving aan de voorkant. Dit verschijnsel is nog niet meegenomen in de modellering en kan in de toekomst verbeterd worden. Het verschijnsel hoeft niet meegenomen te worden als het nieuwe ontwerp van de hovercraft een betere massa verdeling heeft. Om vision en control in de toekomst te gaan implementeren zullen nog twee belangrijke zaken moeten gebeuren. Allereerst zal er een driver geschreven moeten worden die de Video USB adapter ondersteund van X10 en vervolgens kan begonnen worden met vision en een ontwerp van een regelaar die ervoor zal moeten zorgen dat de hovercraft bijvoorbeeld een rechte lijn gaat volgen.

Table of Contents

Introduction

Over the centuries there have been made many efforts to reduce the element of friction between moving parts. Shipbuilders have always known that 'drag' results from the skin friction of water acting on hulls. Early attempts at forcing air through tiny jets either under the hull or all around it failed simply because the engine needed to produce too much air and the propulsion engines put the cost way above any advantage in speed. Many other ideas where tried during the early 20th century until they came together in the genius of one man - Sir Christopher Cockerell. Although Cockerell's first tests were carried out on dry land the main aim was to prove that drag or friction between boats and water could be substantially reduced if the 'craft' floated on an air cushion. And so the 'hovercraft' came in to being. Nowadays there are several types of hovercrafts, but a definition of a hovercraft can still be given:

A hovercraft is a self-propelled vehicle, dynamically supported on a self-generated cushion of air contained in a flexible skirt such that it is totally amphibious and has some ability to travel over less than perfect surfaces. Propulsion is not derived from contact with the water or the ground.

The hovercraft that is used in this project is a lightweight 2-channel model hovercraft from *HOVERCRAFTmodels.com™*, type Electro Cruiser. Aims for this hovercraft project are developing a dynamical model of the hovercraft and making a connection between hovercraft and PC for PC control.
The document is organized as follows: the first chapter gives an introduction about the working principles of the hovercraft. Chapter 2 covers the hardware and software issues in this project. Chapter 3 describes the steps that lead to a 2-channel open loop PC controlled hovercraft. In chapter 4 is derived a dynamical model of the hovercraft. The last chapter gives recommendations for future research and summarizes the project.

1 Background

A hovercraft or also called an air cushion vehicle is a vehicle that can drive on both land and water. This vehicle differs from other vehicles in that way, it needs no surface contact for traction. Obstacles such as gullies and waves can be taken very easily like it is a flat surface. The reason for this is a generated air cushion between the hovercraft itself and the ground surface. The model hovercraft that is used in this project is a remote controlled model hovercraft with two manually controlled input channels. In this chapter will be explained the working principle of the hovercraft.

1.1 Hovercraft principle

In fact there are different types of hovercraft, one of them is an amphibious hovercraft. The model hovercraft is also supposed to be an amphibious hovercraft, this means it can operate over land and over water.
In the hovercraft are placed two propellers both driven by an electric motor and one of them is used to provide lift by keeping a low-pressure air cavity under the craft full of air. As the air pressure is increased the air lifts the craft by filling the cavity. The cavity or chamber in which the air is kept is called a 'plenum' chamber. At the point when the air pressure equals the weight of the hovercraft over the chambers surface area the hovercraft lifts and the air starts to escape through two holes made in the bottom plate. The escaped air creates an air-lubricated layer between the hovercraft and the ground surface. This will lead to a frictionless motion of the hovercraft, taken into account that the minimum contact between skirt and the ground surface during the motion is negligible. The amount of the total weight that a hovercraft can raise is equal to the pressure in the plenum chamber multiplied by the area of the hovercraft. In basic hovercrafts does the air escape around the edge of the skirt, but the main principle is the same. This plenum chamber principle is visualized in fig 1.1.

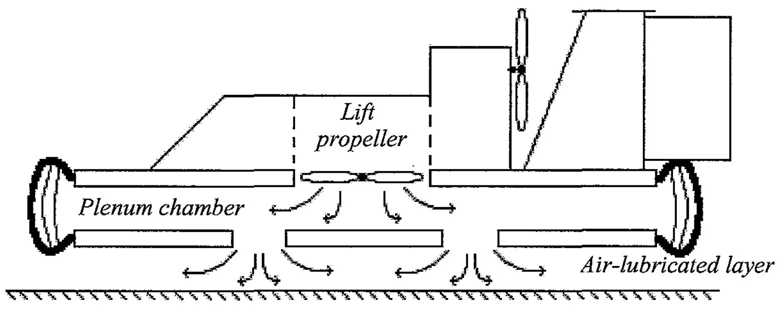


figure 1.1: plenum chamber

A constant feed of air is needed to lift the hovercraft and to compensate for the air being lost through the holes in the bottom plate. The flow must also be greater than the amount of air that escapes through the holes in the bottom plate. The rate of the air loss is not constant, because there is no way of ensuring that any air escapes evenly all around the hovercraft. To maintain also the lift, the engine and propeller have to be sufficiently powerful enough to provide a high airflow rate into the chamber. A cylinder is placed around the lift propeller to improve the efficiency, because it reduces the pressure loss around the propeller tube.

The second propeller is driven by another motor and is placed on the back end of the hovercraft. This motor can only deliver a constant speed to the propeller in contrast to the lift propeller and is used to generate a displacement forward. The propeller can deliver a maximum force of 1,8 N, when there is no wind and the battery is fully charged. This model hovercraft can reach a maximum speed of 3 m/s when full lift is generated and if the rudder is in default position so that the hovercraft only moves in a straight line.
Without a rudder de hovercraft is unmanoeuvrable. Therefore a vertical rudder is placed on the back of the hovercraft behind the back thrust propeller. Conventional hovercrafts normally have a very high rudder position over the center of gravity. Its action not only creates turning moments but also a drifting force and rolling moment, which leads to an outward-banked turn. The position of the rudder

on the model hovercraft is not extremely high, but these effects have to be hold in the back of the head when studying the motion characteristics of the hovercraft.
A camera is also placed on the hovercraft. This camera will be used in future for vision and control aspects. The model hovercraft is visualized in figure 1.2.

1.2 Skirt design

The invention of the flexible skirt was a big step forward in the hovercraft history. Before the flexible skirt had been thought necessary, powerful lift engines were needed to create only a few millimeters lift under the hull hard structure. The main idea behind the air cushion introduction was to create air lubrication between the ground surface and the hovercraft, in a way that leads to a significant reduction of the lift power. The use of an inflated bag around the hovercraft resulted in an efficient air distribution and a reduction of the lift power. Other advantages of the flexible skirt development are a better stability and a better obstacle clearance. The function of skirts for a hovercraft has proved to be as important to the air cushion vehicles as that of rubber tyres for an automobile.
The material type and the structure of the skirt are playing an important role in the skirt design step. The material for a skirt bag should have high tearing and tension strength, but with as few as possible abrasion. The skirt bag is normally divided in several parts to improve the absorption of obstacles.

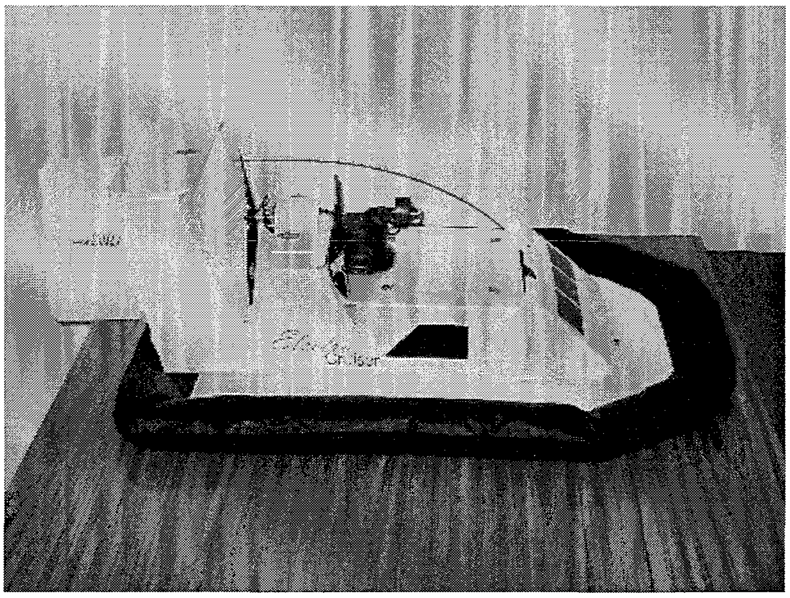


figure 1.2: Electro Cruiser, model hovercraft

2 Set up project equipment

In this chapter will be discussed the set up of hardware and software equipment during the hovercraft project.

2.1 Hardware

One goal is to connect the hovercraft to the computer and control it open loop. A computer is not being able to send signals out without using correct software and a data acquisition board. The software will come up in section 2.2, but the data acquisition board that is used during the project is from AXIOM Technology Co. Ltd, type AX5411H. This DA&C board has 24 digital I/O channels, 16 analog inputs with a 12-bit resolution and a maximum sample rate of 60 KHz. Further has the board 2 analog outputs with a 12-bit resolution and output ranges are available between 0 to 5V or 0 to 10 V. At this time only the two analog outputs are in use. Software cannot communicate with the DAC board if the base address isn't defined. On the DAC board is set the correct base address with a DIP switch. The DIP switch address for this board in hexadecimal notation is 0X320. In switch notation this means: off off on on off on. Also the D/A range is specified on the DAC board with a jumper. The range for signals going out lies between 0 and +5 Volt. More details about the AX5411H DA&C board can be found in the reference manual available at the research laboratory of Systems and control research or on the internet link http://www.axiomtek.com/Download/Manual/AX5411H.pdf.

Future research purposes will be vision and control in combination with the hovercraft. The cameras for vision and control are from X10 model Xcam2 (http://www.x10.com). One camera on the hovercraft for vision and control is not enough, because it is very difficult to get the orientation and position out of one camera image. Therefore a second camera needs to be suspended, to overlook the whole area. X10 has delivered the hardware for multi camera use. With this hardware it is possible to change from one camera to another one. Both cameras are wireless color cameras with a CMOS sensor and a frame rate of 30 frames per second. Images are 510 x 492 pixels with a resolution of 1/3". The range of view is 60° and minimum illumination of 3 lux (f1.9). X10 uses also a VA11A USB adapter to convert video signals into USB, but at the moment there is no support for this USB adapter under linux. People in Europe are developing a driver under linux, which is supporting most camera types including the X10 camera, but there are no working drivers yet. The development of the driver is slower than expected and the perspectives for the near future are not positive. This will result in writing an own driver for the X10 VA11A USB adapter. This adapter is using a Video-USB chip of Sunplus Technology Co, Ltd type SPCA506A1. The user manual of the sunplus chip can be found on the following internet link
http://www.sunplus.com.tw/products/pdf/media/spca/datasheet/ca506a1v10.pdf.

2.2 Software

There are two operating systems, Windows and Linux, available on the computer in the laboratory. The problem with Windows and Linux is that they are designed to optimise average performance and try to give every process a fair share of computing time. This is great for general purpose computing, but for real-time programming are precise timing and predictable performance more important than average performance. For example, if a camera fills a buffer every millisecond, a momentary delay in the process reading that buffer can cause data loss. For Windows and Linux are both methods to create a real-time environment, but the source code for software in Windows is generally not available, so further development is limited. Therefore Linux is chosen instead of Windows as operating system during this project. Real-time Linux is a small real-time operating system and is used to get the Linux system under control to meet real-time computing needs. The basic idea is to run Linux under the control of a real-time kernel. When there is real-time work to be done, the real-time operating system runs its task. When there is no real-time work to be done, the real-time kernel schedules Linux to run. So Linux is the lowest priority task of the RT-Kernel. At the computer belonging to the hovercraft are installed a Linux kernel version 2.2.18 and a RT kernel version 3.0. Both installation instructions are added in the appendices respectively L and M.

3 Hovercraft control

In this chapter are presented the hardware configuration of the hovercraft and the hardware changes to the radio control transmitter to create a PC compatible transmitter. The last section of this chapter gives an explanation of the used control software.

3.1 Control configuration

The hovercraft is a vehicle with 6 degrees of freedom and there are only two input channels that can be used to control the hovercraft. This means the hovercraft is an under-actuated vehicle, because there are less control actuators than degrees of freedom. Let us first consider the control configuration options. The model hovercraft has besides the air rudder also two propellers, one for the lift and one for the drive forward. Due to the limitation of only two output channels, two control configurations can be presented.
The first configuration operates with the lift propeller on a constant maximum lift and the back thrust propeller is variable. So the control channels in this configuration are the back thrust motor and the air rudder. It is hard to keep the hovercraft on the same position during operation in this configuration. This is a balancing problem and balancing the hovercraft is difficult, because not only the flexible structure of the hovercraft also the replacement of the battery pack after recharging leads to small change in the weight distribution. Manoeuvring in a small area makes it difficult to work with this configuration.
The second configuration keeps the back thrust on a constant speed and makes it possible to vary the lift propeller speed. The rudder is now one output channel and the lift motor is one. Using a variable lift creates friction if the lift force is lowered. With creating friction, the hovercraft is not really a hovercraft any more, but it is easier to control it in the current test environment. Configuration two is chosen because of a better manoeuvrability of the hovercraft in the test environment. In figure 3.1 is presented a connection diagram of the motors and radio equipment with respect to configuration two.
BP1 is a 6-cell rechargeable battery pack of 7,2V and 2200 mAh. *BP1* supplies the lift motor *Mo 2* and

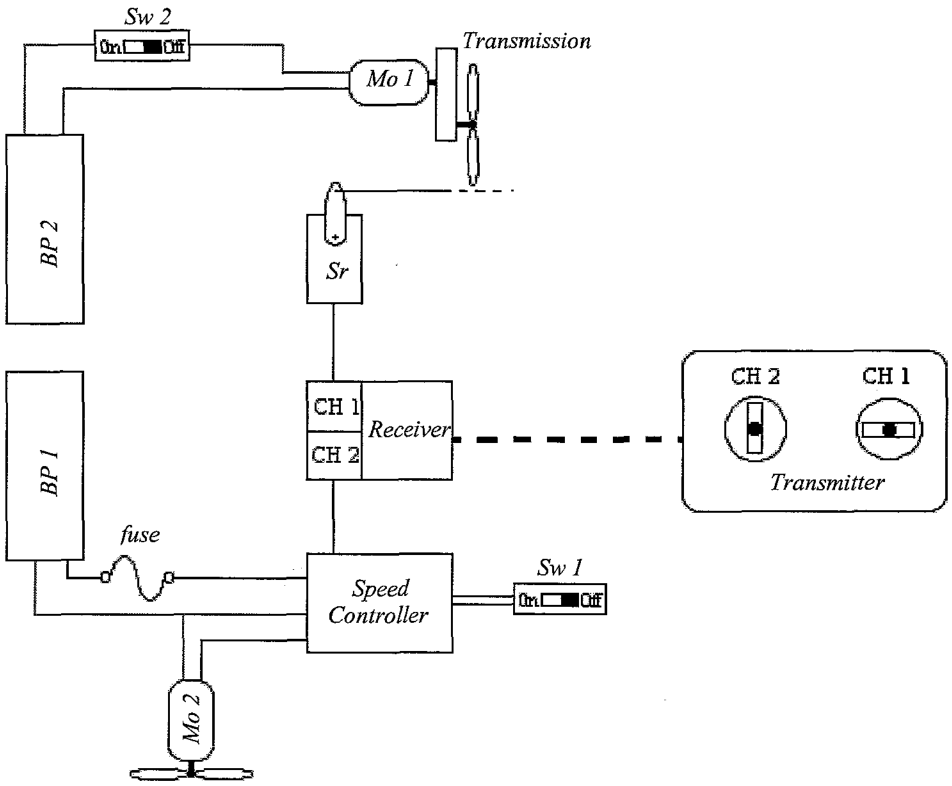


figure 3.1: connection diagram of the radio equipment and the motors

the electronic speed controller, that is also secured with a fuse. Switch *Sw 1* is used to turn the receiver and speed controller on or off. The other rechargeable battery pack *(BP2)* of 7,2 V and 1500 mAh supplies the back thrust motor *Mo 1.* Between *Mo 1* and the back thrust propeller is also placed a transmission, with a gearbox ratio of 1:3. The delivered drive forward is to low without using a transmission. Switch *Sw 2* is used to turn the back thrust motor on or off. To complete the whole circuit, a receiver is needed for receiving the input channels from the radio control transmitter and subsequently sending it to the speedcontroller and the rudder servo, *Sr.* Motors *Mo 1* and *Mo 2* are electro motors of the type Graupner speed 400. The system has a receiving range of 350 yards.

3.2 Open loop PC control

The hovercraft is a remote controlled vehicle, which has to be converted to a PC controlled vehicle. Therefore, the transmitter is converted to a control station that can be used as a normal manual remote controller and it can be used for PC control. Manual remote control has to be fully uncoupled if PC control is used and also in the other way round. Two double pole double throw (dpdt, on-off-on) switches are used to establish an uncoupling of one control mechanism when the other is used. The place where the disconnection takes place is at the joysticks of the transmitter. These joysticks are variable resistors, which can be replaced by a computer output voltage during PC control. Each variable resistor has a certain range in which the hovercraft is responding. The voltage range, initial value and the resolution of each channel are tested and given in table 3.1. It is also verified that the variable resistor is linear.

D/A Channel number	initial value [Volts]	lower boundary [Volts]	upper boundary [Volts]
0	2,80	2,45	2,80
1	2,56	2,10	3,03

table 3.1: channel specification

The implementation of the dpdt switches, *Switch CH2 and CH1,* and the connection diagram of the PC

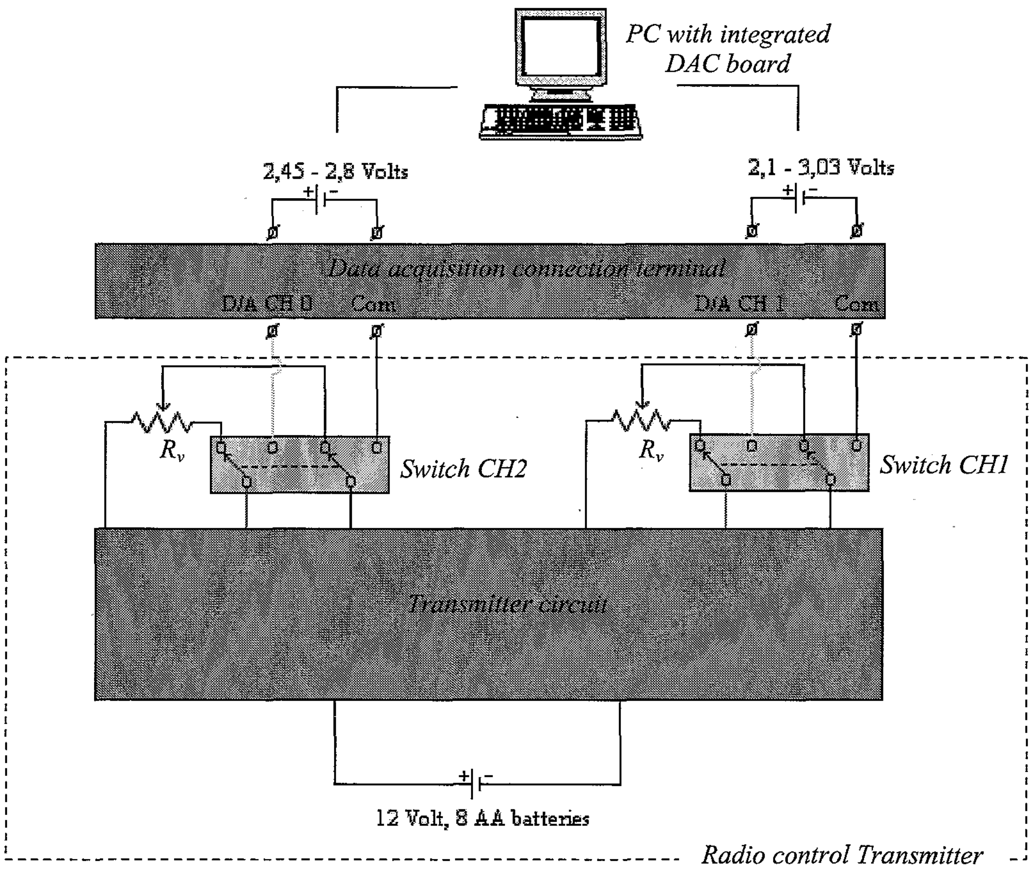


figure 3.2: Converted radio control transmitter

compatible transmitter is visualised in figure 3.2.
In figure 3.2 are used a few names referring to channels. To make it clear, *Switch CH2* and *Switch CH1* are related to the channels on the transmitter. CH2 and CH1 on the transmitter refer to respectively the lift motor output and rudder servo output. D/A CH 0 and D/A CH 1 are used on the data acquisition connection terminal and in the control software on the computer. Here CH0 and CH1 are referred to respectively the lift motor and rudder servo.
Both dpdt switches are placed on top of the remote controller and each has three possible positions. Normal remote control mode is active when both switches are pointed to the transmitter antenna and PC control mode is active when both switches are pointing to the outside. The middle position can be seen as an off position. Be aware of the fact, that the transmitter is sensitive for disturbances and if the transmitter is unplugged of the data acquisition connection terminal during PC control mode, this will lead to another configuration and the hovercraft starts moving. So first turn off the receiver switch *Sw 1*on the hovercraft before disconnecting.

3.3 Control program

Hovercraft is the name of the designed control program based on Rtcon to do real time control of the hovercraft in RTLinux. Rtcon uses real time threads to do data acquisition, calculating controller outputs and sending signals out sequentially. Rtcon is written by Ray Eaton and is based on SISO control and therefore it can send only one signal out at a certain time. To control the hovercraft, two output signals have to be implemented namely one for the air rudder servo and one for the lift motor. Control purposes for this project are controlling the hovercraft open loop by clicking buttons in a created Graphical User Interface. The main structure of Rtcon is left unchanged as much as possible to make future implementation of a feedback controller easier.

3.3.1 Program structure

Hovercraft is using three separated real time threads as mentioned above. The three threads are executed in succession starting with the sampling thread (A/D). When the sampling has taken place, it terminates and starts the control thread. When the control thread is finished it terminates and starts the D/A conversion thread. When the third thread is finished, it stops and awakes the sample thread again. The cycle will start from the beginning again. All the treads are placed in a coded control module *control_mod.o.* The source code of the control module can be found in *rtmodule.c,* appendix B. *Rtmodule.c* starts with making the treads and can also unload the threads when the control module is removed. After the making of the threads will be started with sampling.
Real time threads in RT-Linux can communicate with each other through the use of global memory. This is because they exist in the same kernel space. A normal Linux task, for example the user interface, exists in its own space, separated with other tasks by a protected memory barrier. To communicate with a Linux task, global memory is not sufficient. RT-Linux provides a mechanism known as shared memory to communicate between real-time threads and Linux tasks. This mechanism is provided by *mbuff* module, see appendix J. This means, if a user changes a value in the Graphical User Interface (user space), the controller thread in the kernel space will use this new value in the next calculations. The main part of the shared memory structure is the shared structure *cloop*. This structure contains controller parameters and state variables to share between the user space and the kernel space. The *cloop* (appendix D) structure and its functions are defined in the header file *cloop.h,* appendix C. The functions for sampling, *sample(cloop)*, controlling, *control(cloop)* and digital to analog conversion, *dtoa(cloop)* are placed in a separated file called *thread_code.c,* appendix E. To ease programming, a common interface for sharing an array of long data type (CDSM) in user and kernel space is provided in cdsm.c and cdsm.h, respectively appendix H and I.

A Graphical User Interface (GUI), see fig 3.3, is created in the file *hovercraft.c* using the GTK toolkit, see appendix F. The GUI contains eight buttons four buttons in the middle called Left, Right, Up and Down are used to control the hovercraft. By clicking on the hovercraft control buttons the output voltages change and the hovercraft will move. The change of the output voltages can also be seen on the right side of the hovercraft control buttons. At the bottom of the GUI is a start and stop button to start and stop control. Stop resets the output values, so the rudder and the lift motor are going to their initial position. Clicking on the Quit button means leaving the hovercraft program without resetting the output channels and with the Log button can data be stored in an output file. Clicking on the log button again will stop logging. The name of the log file is *hcraft* and it contains columns with time, r, y, u0 and u1. The time and data of the output channels u1 and u2 are at the moment the only parameters of interest. Reference signal r and input (of DAC board) signal y are not used.

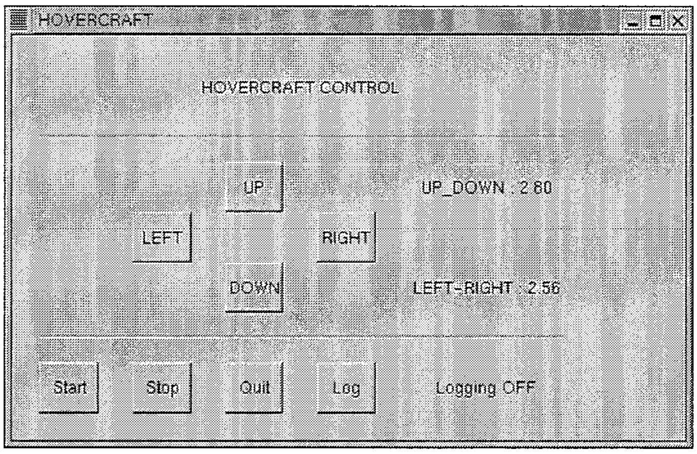


figure 3.3: Graphical User Interface for hovercraft control

The buttons are created in *hovercraft.c* as mentioned earlier. If the user clicks on a button the output changes and the new value has to be sent to the shared memory because the real time threads in the kernel space need to know what the user wants. A hovercraft control button function as for example the *left_button* function works as follows. After clicking on the LEFT button, the new value is obtained by subtracting 0,01 of the old value. The new value has to be checked if the value is lying within a certain range defined in table 3.1, before sending it to *setpt1* in the shared memory structure. Therefore a saturation function in *cloop.c* is used. The boundary values for both channels are specified in function *lookup_sat* in *cloop.c*. Function *left_button* is illustrated below. The resolution of both output channels is 0,01 Volt. For the rudder this means a resolution of 1,29°, within a range of -60° to 60°.

```
void left_button( GtkWidget *widget, int *arg)
{
    float setpt1 = cloop->setpt1;
    char s[14];
    float a,b;
    a=lookup_sat(1);
    b=lookup_sat(2);

    setpt1 -= 0.01;
    setpt1 = sat(setpt1,a,b);
    sprintf(s,"LEFT-RIGHT : %3.2f",setpt1);
    gtk_label_set_text((GtkLabel *)(cgui.label[4]), s);
    cloop->setpt1 = setpt1;
    printf("u1 = %3.2f\n",setpt1);
}
```

Real time threads can now use the latest data from the shared memory when they need it. Sample- and control thread are only there to make implementation of it in the future easier. At the moment it is open loop control, so the digital to analog conversion is the only real time thread that has actually a task. Function *dtoa* is responsible for sending signals out, because it calls on the function *io_dacout*, which is standing in *cloop.c*. But *dtoa* scales first with the function *scale_ftoi* a float to a 12bit integer, taking into account a lower and upper boundary. Function *dtoa* is added below.

```
void dtoa(volatile cloop_t *cloop)
{
    float ut0,ut1;
    int uint0,uint1,stat;
    ut0 = (get(cloop,0)).u0;
    uint0 = scale_ftoi(ut0, 0.0, 5.0, 0, 4096);
    stat = io_dacout(cloop, 0, uint0);
    ut1 = (get(cloop,0)).u1;
    uint1 = scale_ftoi(ut1, 0.0, 5.0, 0, 4096);
    stat = io_dacout(cloop, 1, uint1);
}
```

Io_dacout sends the scaled value to output channel 0 or 1. Function *io_dacout* is presented below. Channel 0 is represented in *io_dacout* by case 0 and channel 1 is represented by case 1. IO_BASE+4 and IO_BASE+5 represent the addresses in I/O space for D/A CH0. For D/A CH1 the addresses are IO_BASE+6 and IO_BASE+7.

```
int io_dacout(volatile cloop_t *cloop, int chan, int data)
{
    unsigned int low = (data << 4) & 0x00f0;
    unsigned int hi = (data >> 4) & 0x00ff;

    if( data < 0 || 4095 < data){
        #ifdef __RTL__
        rtl_printf("ax5411: dac out value (%d) out of range\n",data);
        #else
        printf("ax5411: dac out value (%d) out of range\n",data);
        #endif
        return -1;
    }
    switch(chan) {
        case 0:
            outb(low, IO_BASE+4);
            outb(hi, IO_BASE+5);
            break;

        case 1:
            outb(low, IO_BASE+6);
            outb(hi, IO_BASE+7);
            break;

        default:
            return -1;
            break;
    }
    return 0;
}
```

The steps described above from clicking the buttons to sending signals out will continue as long as the user doesn't press the stop button. Starting Hovercraft is not only running hovercraft, but first has to be inserted the control module in RTLinux. To insert the control module and starting the hovercraft program go through the following steps while running RTLinux with root privileges.

- Go to the directory
  > /home/hovercraft/hover
- Insert the control module
  > insmod *control_mod.o*
- Start Hovercraft
  > ./hovercraft

The Graphical User Interface in figure 3.3 is launched and open loop control is possible.

4 Model

In this chapter will be started with deriving a dynamical model for the hovercraft. Subsequently are determined the model parameters such as moment of inertia and at the end will be given the simulation results with validation of the dynamical model.

4.1 Dynamical model

Analyzing the motion of hovercrafts is possible in 6 degrees of freedom (DOF) using the equations of motion of the hovercraft. But before deriving the equations of motion it is convenient to define the coordinate frames and the 6 DOF as indicated in figure 4.1 and table 4.1.

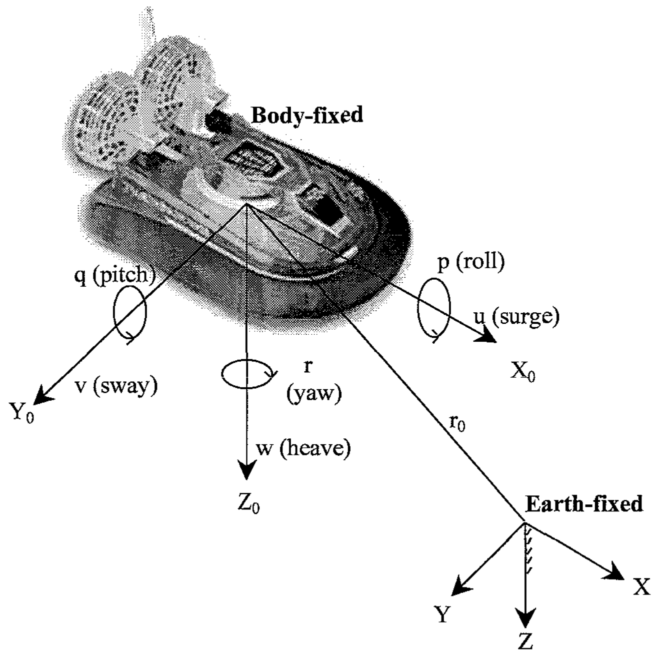


Figure 4.1: Body-fixed and earth-fixed reference frames

DOF		Forces and moments	Linear and angular velocity	Positions and Euler angles
1	motions in x-direction (surge)	X	u	x
2	motions in y-direction (sway)	Y	v	y
3	motions in z-direction (heave)	Z	w	z
4	rotation about the x-axis (roll)	K	p	ϕ
5	rotation about the y-axis (pitch)	M	q	θ
6	rotation about the z-axis (yaw)	N	r	ψ

table 4.1: 6 DOF coordinates specifications

Two coordinate frames are chosen, the moving coordinate frame X_0 Y_0 Z_0, which is a body-fixed frame on the hovercraft and an earth-fixed frame *XYZ*. The motion of the hovercraft will be described relative to the earth-fixed frame.
The rigid body equations of motion are derived using Euler's first and second axioms, respectively:

$$\dot{p}_c = f_c; \qquad p_c = mv_c \tag{4.1}$$

$$\dot{h}_c = m_c; \qquad h_c = I_c\omega \tag{4.2}$$

Here the index c indicates that it is referred to the body's centre of gravity. f_c and m_c are the forces and moments acting on the body, ω is the angular velocity and I_c is the inertia tensor about the centre of gravity of the body. A few assumptions are made when deriving the equations of motion. It will be assumed that the mass of the body is constant in time and the vehicle is rigid. When the hovercraft is assumed rigid this means there are no forces acting between individual elements of mass. Another assumption is made with respect to the earth-fixed frame, this is assumed to be inertial. The following formula is needed to derive the equations of motion for an origin in a body fixed rotating coordinate system.

$$\dot{c} = \overset{\circ}{c} + \omega \times c \tag{4.3}$$

Here $\dot{c}$ is the time derivative of a vector c in the earth-fixed frame and $\overset{\circ}{c}$ is the time derivative in the body fixed reference frame $X_0Y_0Z_0$. This formula describes a relation between a time derivative of an arbitrary vector c in XYZ and $X_0Y_0Z_0$. It can be verified that the angular acceleration $\dot{\omega}$ is equal in the body-fixed and earth-fixed reference frame using equation (4.3).

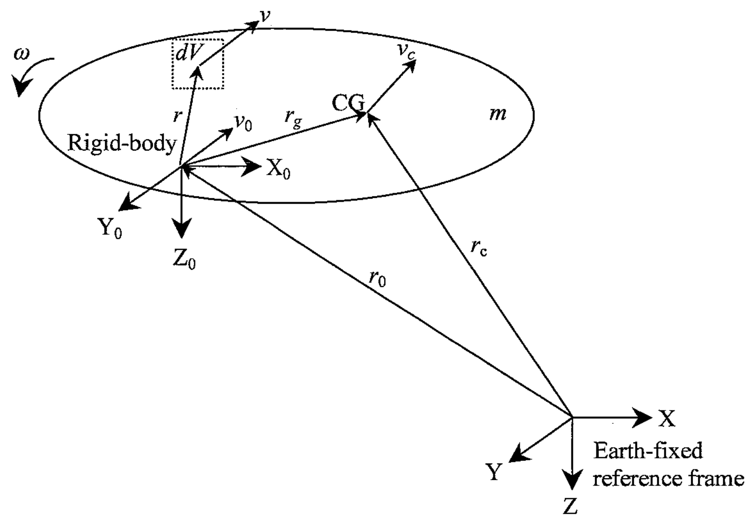


figure 4.2: Earth-fixed frame XYZ and body-fixed frame $X_0Y_0Z_0$

Deriving the equations of motion is divided in two parts. A translation- and rotation part described by respectively equation (4.1) and (4.2).

4.1.1 Translation dynamics

Starting with the translation part it can be seen from figure 4.2 that,

$$r_c = r_0 + r_g \tag{4.4}$$

The velocity of the centre of gravity is obtained by differentiating equation (4.4).

$$v_c = \dot{r}_c = \dot{r}_0 + \dot{r}_g \tag{4.5}$$

Using equation (4.3) and assuming that $v_0 = \dot{r}_0$ and $\overset{\circ}{r}_g = 0$ it follows that,

$$\dot{r}_g = \overset{\circ}{r}_g + \omega \times r_g = \omega \times r_g \tag{4.6}$$

then,

$$v_c = v_0 + \omega \times r_g \tag{4.7}$$

The acceleration of the centre of gravity can be found by differentiating the velocity vector this leads to:

$$\dot{v}_c = \dot{v}_0 + \dot{\omega} \times r_g + \omega \times \dot{r}_g \tag{4.8}$$

which yields

$$\dot{v}_c = \overset{\circ}{v}_0 + \omega \times v_0 + \overset{\circ}{\omega} \times r_g + \omega \times \left(\omega \times r_g\right) \tag{4.9}$$

Substituting equation (4.9) into (4.1) leads to,

$$m\left(\overset{\circ}{v}_0 + \omega \times v_0 + \overset{\circ}{\omega} \times r_g + \omega \times \left(\omega \times r_g\right)\right) = f_0 \tag{4.10}$$

The vector r_g is a null vector when the origin of the body-fixed frame coincides with the centre of gravity of the hovercraft. With the body-fixed frame $X_0Y_0Z_0$ chosen in the centre of gravity equation (4.10) can be rewritten as,

$$m\left(\overset{\circ}{v}_c + \omega \times v_c\right) = f_c \tag{4.11}$$

4.4.2 Rotation dynamics

Rotational equations of motion can be obtained by using equation (4.2). The angular momentum about the origin of the body-fixed frame is defined as:

$$h_0 = \int_V r \times v \ \rho_A dV \tag{4.12}$$

Differentiating equation (4.12) with respect to time leads to,

$$\dot{h}_0 = \underbrace{\int_V r \times \dot{v} \ \rho_A dV}_{\textit{Moment vector } m_0} + \int_V \dot{r} \times v \ \rho_A dV \tag{4.13}$$

The following relation for the velocity of a volume element can be found in figure 4.2.

$$v = \dot{r}_0 + \dot{r} \quad \Rightarrow \quad \dot{r} = v - v_0 \tag{4.14a ,b}$$

Substituting expression (4.14b) into equation (4.13) and making use of property that $v \times v = 0$ leads to a new expression for the derivative of the angular momentum.

$$\dot{h}_0 = m_0 - v_0 \times \int_V v \ \rho_A dV \tag{4.15}$$

or similarly,

$$\dot{h}_0 = m_0 - v_0 \times \int_V \left(v_0 + \dot{r}\right) \rho_A dV = \dot{h}_0 = m_0 - v_0 \times \int_V \dot{r} \ \rho_A dV \tag{4.16}$$

Equation (4.16) can be rewritten using the derivative of the vector r_g.

$$r_g = \frac{1}{m}\int_V r\rho_A dV \xrightarrow{\textit{differentiating leads to}} m\dot{r}_g = \int_V \dot{r}\rho_A dV \tag{4.17a, b}$$

Using the expression $\dot{r}_g = \omega \times r_g$ and equation (4.17b) leads to a new equation for $\dot{h}_0$,

$$\dot{h}_0 = m_0 - m v_0 \times (\omega \times r_g) \tag{4.18}$$

The angular momentum can be written as follows,

$$h_0 = \int_V r \times v \, \rho_A dV = \underbrace{\int_V r \times v_0 \, \rho_A dV}_{1^{st}} + \underbrace{\int_V r \times (\omega \times r) \, \rho_A dV}_{2^{nd}} \tag{4.19}$$

Both terms on the right hand side of expression (4.19) can be simplified. The first term can be reduced with equation (4.17a) if $\times v_0$ is taken out of the integral and the second term is defined as $I_0 \omega$. Equation can now be written as,

$$h_0 = m r_g \times v_0 + I_0 \omega \tag{4.20}$$

Differentiating equation (4.20) gives a new expression for $\dot{h}_0$ assuming that I_0 is time independent.

$$\dot{h}_0 = I_0 \overset{\circ}{\omega} + \omega \times (I_0 \omega) + m(\omega \times r_g) \times v_0 + m r_g \times \left(\overset{\circ}{v}_0 + \omega \times v_0 \right) \tag{4.21}$$

Equation (2.82) can now in combination with $(\omega \times r_g) \times v_0 = -v_0 \times (\omega \times r_g)$ be used to eliminate $\dot{h}_0$ out of (4.21). This results in the rotational equations of the hovercraft.

$$I_0 \overset{\circ}{\omega} + \omega \times (I_0 \omega) + m r_g \times \left(\overset{\circ}{v}_0 + \omega \times v_0 \right) = m_0 \tag{4.22}$$

As mentioned earlier in the translation part the origin of the body-fixed reference frame is chosen to coincide with the centre of gravity of the hovercraft. This will result in simplified rotation equations, because vector r_g is a null vector.

$$I_c \overset{\circ}{\omega} + \omega \times (I_c \omega) = m_c \tag{4.23}$$

The equations of motion for the 6 DOF hovercraft can be written down using the translation part (4.11) and the rotation part (4.23). Vectors v, ω, f_c and m_c are respectively the linear velocities, angular velocities, external forces and moments. They are defined as follows,

$$v = \begin{bmatrix} u \\ v \\ w \end{bmatrix}; \quad \omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix}; \quad f_c = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}; \quad m_c = \begin{bmatrix} K \\ M \\ N \end{bmatrix} \tag{4.24}$$

A few assumptions are also made with respect to the 6 DOF dynamics. At the moment are the rotations about the x and y-axis neglected (roll and pitch) and also the translation in the z-direction (heave), because these DOF have no considerable influence. In ship modelling it has for example significant influence when these DOF are neglected.

$$\begin{aligned} \dot{u} &= vr + \frac{1}{m}X \\ \dot{v} &= -ur + \frac{1}{m}Y \\ \dot{r} &= \frac{1}{I_z}N \end{aligned} \tag{4.25}$$

The forces *X*, *Y* and moment *N* are defined using figure 4.3. The back thrust propeller causes an

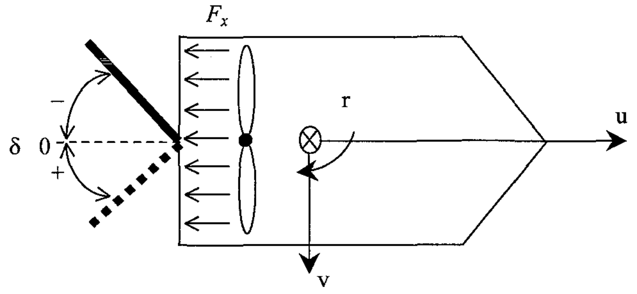


figure 4.3: defining external forces and moments

external force in surge direction of 1,8 N if the rudder stands in its initial position 0. This value for the back thrust force is measured statically with a modern version of a steelyard. It was possible to measure the pulling power of the hovercraft and because there is no friction the pulling force is equal to the back thrust force. A force in sway direction and a moment about the yaw axis result from another rudder position δ. If the rudder stands in a position except 0 than ½ F_x works purely in surge direction and the other ½ F_x can be spit up into a surge part, sway part and a moment about the yaw axis. *X*, *Y* and *N* are defined as,

$$\begin{aligned} X &= \tfrac{1}{2} \cdot F_x (1 + \cos(\delta)) \\ Y &= \tfrac{1}{2} \cdot F_x \sin(\delta) \\ N &= \tfrac{1}{2} \cdot F_x \cdot a \cdot \sin(\delta) \end{aligned} \tag{4.26}$$

Here *a* is the distance in negative surge direction between centre of gravity and the point where the rudder is attached to the hovercraft. The value for *a* is 0,4 m.
In reality the model hovercraft has two main types of friction. The first type of friction is dependent of the lift force. At maximum lift force is no contact with the ground surface and also no friction. If the lift force is lowered to 0 the friction will increase and stop the hovercraft because the coulomb friction is to big for the back thrust force to get over it. Implementation of this friction is done with an arctan function multiplied with a lift force dependency. The lift force F_l is unknown but is chosen equal to the normal force of the hovercraft *mg*= 21N. This means the lift force is variable between 0N and 21 N. The other friction is dependent of the velocity as a result of air friction and can be seen as a damping term. Implementation of these two frictions results in a dynamic model of the hovercraft with 3 DOF.

$$\begin{aligned} \dot{u} &= vr + \frac{1}{m}\left(-d_1 \cdot u + \tfrac{1}{2} \cdot F_x (1 + \cos(\delta)) - \mu_1 \frac{2}{\pi}(mg - F_l)\arctan(5000 \cdot u)\right) \\ \dot{v} &= -ur + \frac{1}{m}\left(-d_2 \cdot v + \tfrac{1}{2} \cdot F_x \sin(\delta) - \mu_2 \frac{2}{\pi}(mg - F_l)\arctan(5000 \cdot v)\right) \\ \dot{r} &= \frac{1}{I_z}\left(-d_3 \cdot r + \tfrac{1}{2} \cdot F_x \cdot a \cdot \sin(\delta) - \mu_3 \frac{2}{\pi}(mg - F_l)\arctan(5000 \cdot r)\right) \end{aligned} \tag{4.27}$$

Damping coefficient d_1 is determined empirical using a digital camera and the equation of $\dot{u}$ above. With the digital camera it was possible to film the passing hovercraft at a constant maximum speed. The covered distance in one frame was measurable using markers on the floor. The equation of $\dot{u}$ can be reduced, because the hovercraft was moving purely in surge direction at constant speed ($\dot{u} = 0$), the hovercraft had maximum lift force and the rudder angle was 0. Damping coefficient d_1 can now be calculated and has a value of 0,6 Ns/m. For d_2 is taken 0,8 Ns/m, because the surface in contact with the air in sway direction is bigger than in surge direction, this is an assumption. Damping coefficient d_3 is also unknown and for this coefficient is therefore chosen a reasonable value of 0,1 Nms/rad.
The friction coefficients μ_1, μ_2 and μ_3 are estimated during the simulation; $\mu_1 = 0{,}1$, $\mu_2 = 0{,}01$ and $\mu_3 = 0{,}004$. To describe the motion of the hovercraft relative to earth-fixed frame, we introduce a coordinate transformation as follows.

$$\begin{aligned} Z_1 &= x \cdot \cos(\psi) + y \cdot \sin(\psi) \\ Z_2 &= -x \cdot \sin(\psi) + y \cdot \cos(\psi) \\ Z_3 &= \psi \end{aligned} \tag{4.28}$$

With this transformation it is possible to analyse the motion of the hovercraft in the earth-fixed frame.

4.2 Moment of inertia

The moment of inertia about the z-axis is an unknown parameter in the derived model. It is essential for future control ambitions to calculate the moment of inertia of the hovercraft. This can be done in two ways, namely empirical and analytical. In the next two sections will be discussed both ways.
In section 4.2 is used J instead of I_z for indicating the moment of inertia. This is done to avoid confusion with the Laplace transformed of the armature current I.

4.2.1 Empirical

The empirical calculation of the moment of inertia is done, using a test installation at the school of Mechanical Engineering. The line-up is a vertical placed controllable DC servomotor of which the angular displacement and indirectly the angular velocity can be measured. On the output shaft is fixed a specially designed timber plate of MDF to carry the hovercraft during the experiment. Fixing the MDF plate on the output shaft is possible using a flange, which is already on the shaft. When the Hovercraft is placed on the plate, its centre of gravity lies automatically on the output shaft of the motor. A drawing of the MDF plate is enclosed in appendix A.
The moment of inertia of the hovercraft can be calculated, if the moment of inertia of the motor and the plate are known. Therefore, two separated experiments are done. The first one is done without using the hovercraft to calculate the inertia of the motor and the MDF plate. The other experiment is done including the hovercraft. The difference in inertia between the two experiments should be the inertia of the hovercraft. For each experiment is measured the angular displacement and the time, while a step input is generated to the motor. In the next part of this section will be gone through the calculation steps that result in a value for the moment of inertia of the hovercraft.
First a transfer function has to be derived with respect to the test system, before calculations can be made. The diagram of the test system is presented in figure 4.4

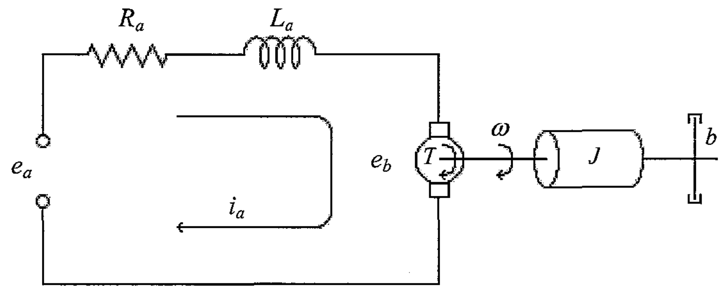


figure 4.4: Diagram test system

In this system,

R_a	= armature resistance, [*Ohm*]
L_a	= armature inductance, [*Henry*]
i_a	= armature current, [*Ampere*]
e_a	= applied armature voltage, [*Volt*]
e_b	= back electromotive force (back emf), [*Volt*]
ω	= angular velocity of the motor shaft, [*Rad/sec*]
T	= torque developed by the motor, [*N.m*]
J	= moment of inertia of the motor and load referred to the motor shaft, [$kg.m^2$]
b	= viscous damping coefficient of the motor and load referred to the motor shaft,[*Nm/rad/sec*]

A rotating circuit called the armature, through which the current i_a flows passes through a magnetic field developed by stationary permanent magnets. This results finally in a torque that turns the output shaft. Since the armature is rotating in a constant magnetic field, the back emf e_b is directly proportional to the angular velocity ω.

$$e_b = K_b \cdot \omega \tag{4.29}$$

,where K_b is called the back emf constant.
The speed of an armature-controlled dc servomotor is controlled by the armature voltage e_a. The differential equation for the armature circuit is

$$L_a \frac{di_a}{dt} + R_a i_a + e_b = e_a \tag{4.30}$$

The armature current produces the torque that is applied to the inertia and damping.

$$J\dot{\omega} + b\omega = T = K_t \cdot i_a \tag{4.31}$$

Where K_t is a constant of proportional, called the motor torque constant. In a consistent set of units, the value of K_t is equal to the value of K_b.
Assuming that all initial conditions are zero, and taking the Laplace transforms of equations (4.29), (4.30), and (4.31), the following equations are obtained:

$$K_b \cdot \Omega(s) = E_b(s) \tag{4.32}$$

$$(L_a s + R_a) \cdot I_a(s) + E_b(s) = E_a(s) \tag{4.33}$$

$$(Js + b)\Omega(s) = T(s) = K_t \cdot I_a(s) \tag{4.34}$$

Considering $E_a(s)$ as the input and $\Omega(s)$ as the output, it is possible to derive the transfer function from equations (4.32), (4.33), and (4.34). The transfer function for the dc servomotor considered is obtained as

$$\frac{\Omega(s)}{E_a(s)} = \frac{K_t}{L_a J s^2 + (L_a b + R_a J)s + R_a b + K_t K_b} \tag{4.35}$$

This transfer function contains several unknown parameters including the moment of inertia J. To calculate the moment of inertia, all the other parameters have to be known. The armature inductance

L_a is a motor constant and has a value of 0,62 mH. L_a in the armature circuit is small and therefore L_a is not determined empirical but taken from the specification data of the motor. The other parameters R_a, b, K_t and K_b are determined empirical, because they have significant values in the transfer function comparing with the inductance.
Determining the armature resistance R_a can be done easily by supplying a small voltage to the armature circuit. When the motor is not moving with the small input voltage because of friction, the back emf will be zero. The armature resistance is 1,779 Ω and is calculated with Ohm's law, because the voltage and the current are measured, see appendix Q.
The back emf/torque constant and the viscous damping of the motor with the supporting plate are measured during one and the same test experiment. For this test, a constant voltage is supplied that results in a constant speed of the motor. During the test are measured the angular velocity, the armature current and the armature voltage. Using equation (4.30) and neglecting the inductance part, it results in a value for the back emf. The back emf constant can subsequently be calculated by making use of equation (4.29) and the values for the angular velocity and the back emf. The value for the back emf constant K_b or torque constant K_t is 0,1066 Vs/rad respectively Nm/A.
The only remaining unknown parameter except for the moment of inertia is the viscous damping coefficient. Making use of equation (4.31), it is possible to calculate the damping coefficient because the angular acceleration is zero and the other parameters are known. It follows that b is 0,000114 Nms/rad.

The following strategy is used to calculate the moment of inertia J. A transfer function of the test installation is available, thus an analytical step response can also be derived. Measuring a step response of the real system and subsequently fitting the analytical step response on it will give a value for J. Writing with the step response in the frequency domain leads to

$$\Omega(s)=\frac{C_{step}\cdot K_t}{s\left(L_aJs^2+\left(L_ab+R_aJ\right)s+R_ab+K_tK_b\right)} \tag{4.36}$$

where C_{step} is the step amplitude.
The step response of the system in the frequency domain is not useful to fit with, therefore an inverse Laplace transformation is needed to transform the step response to the time domain. Calculating the zeros of the denominator called the roots is required to transform from frequency to time domain. Using the roots, the step response can be rewritten in the following form

$$\Omega(s)=\frac{C_{step}K_t}{L_aJ}\cdot\frac{1}{\left(s-p_1\right)\left(s-p_2\right)\left(s-p_3\right)} \tag{4.37a}$$

with

$$p_1=0 \tag{4.37b}$$

$$p_{2,3}=\frac{-\left(R_aJ+L_ab\right)\pm\sqrt{\left(R_aJ+L_ab\right)^2-4L_aJ\left(R_ab+K_tK_b\right)}}{2L_aJ} \tag{4.37c}$$

Taking the inverse Laplace transformation of equation (4.37a) and using (4.37b) and (4.37c), we obtained the following step response in the time domain for the angular velocity ω.

$$\omega(t)=\frac{C_{step}K_t}{L_aJ\cdot P_2P_3}\left(1+\frac{P_3}{P_2-P_3}\cdot\exp\left(P_2t\right)-\frac{P_2}{P_2-P_3}\cdot\exp\left(P_3t\right)\right) \tag{4.38}$$

Measuring a step response on the real system is the next step. A step with amplitude of 24 Volt is supplied to the motor and the angular velocity is measured. This data has to be fitted with the analytical step response function. An optimisation routine called *Fminsearch* in Matlab is used to minimize the squared error between the two step responses. See appendix R for *Fit_J.m* and *Determine_J.m*. The results are presented in figure 4.5. It can be seen that the gearbox has some

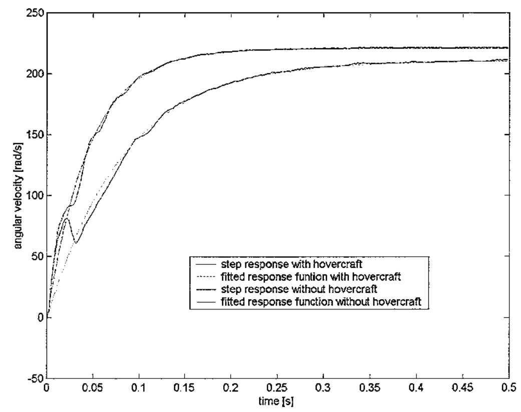


figure 4.5:Moment of inertia fitting results

backlash in the beginning of the acceleration. The motor is turned a little bit through the backlash before each experiment to avoid the backlash as much as possible. The resulting J for the motor including the MDF plate has a value of 0,000295 kg.m^2. All the steps from calculating the system parameters have to be redone to obtain the moment of inertia with the hovercraft on it. This leads to a moment of inertia of 0,000552 kg.m^2. Fitting results including the hovercraft are also presented in figure 4.5.

Now the difference between the two tests is known and represents the hovercraft inertia. One point has to be taken into account before using the moment of inertia. This moment of inertia is referred to the output shaft of the motor, but in reality there is also a gearbox between the motor output shaft and the actually output shaft. A gearbox ratio of 19,2:1 leads to an increase of the moment of inertia with 19,2^2. The moment of inertia about the z-axis of the hovercraft is

$$J_{hovercraft} = (0{,}000552 - 0{,}000295) * 19{,}2^2 = 0{,}0948\ kg.m^2$$

In table 4.2 are summarised the results of the tests with and without the hovercraft. All the data is considered before the gearbox.

Variable/constant parameter	Test without the hovercraft	Test with the hovercraft
Step response voltage, C_{step}	23,895	23,650
Armature resistance, R_a	1,779	1,779
Armature current, i_a	0,235	0,618
Back emf, e_b	23,477	22,551
Viscous damping coefficient b	0,000114	0,000308
Back emf/ torque constant, K_b/K_t	0,1066	0,1061
Moment of inertia before gearbox	0,000295	0,000552

table 4.2: test results

4.2.2 Analytical

Besides the empirical calculation, an analytical calculation of the moment of inertia is done to verify the empirical results. The hovercraft with a total mass of 2,1 kg can roughly be divided in five main parts, with a certain weight and dimension. Namely a battery pack, wireless camera, lift motor, back thrust motor and a main plate. Geometry's for the main parts are chosen basic, like cylinders and beams. Moments of inertia can be calculated for every part with respect to there own centre of gravity. After this, the inertias need to be shifted to the centre of gravity referring to the hovercraft to obtain a moment of inertia for the hovercraft. In figure 4.6 is shown a top view of hovercraft with the placement of the main parts.

The explanation of the letters and the dimensions and masses of each part are specified in table 4.3. Notice that the Lift motor *B* coincides with the centre of gravity of the hovercraft and therefore L_{zb} is zero. *E* is the back thrust motor and is supposed to be a cylinder lying on its side.

	mass [kg]	length [m]	width [m]	radius [m]	shift length [m]
Battery pack, A	0,625	0,13	0,05	-	L_{za}=0,13
Lift motor, B	0,1	-	-	0,015	-
Main plate, C	0,955	0,76	0,39	-	L_{zc}=0,08
Camera, D	0,285	-	-	0,035	L_{zd}=0,11
Back thrust motor, E	0,135	0,07	-	0,015	L_{ze}=0,20

table 4.3: specification hovercraft parts

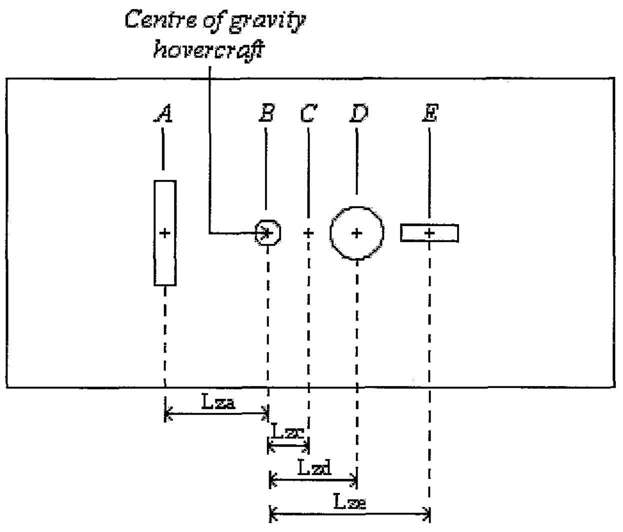


figure 4.6: schematically representation hovercraft

To calculate the total hovercraft inertia, all the separated inertias of battery pack, lift motor, main plate, camera and back thrust motor have to be summarised. The moment of inertia for a beam geometry can be calculated as follows

$$J_i = \frac{m_i}{12}\left(length_i^2 + width_i^2\right) + ml_{zi}^2 \qquad (4.39)$$

where ml^2_{zi} is the shift term to the centre of gravity of the hovercraft and *i* can be *A* or *C*.
For a cylinder, the moment of inertia can be obtained with formula (4.40) including the same shift term as in formula (4.39)

$$J_i = \tfrac{1}{2} m \cdot radius_i^2 + ml_{zi}^2 \qquad (4.40)$$

where *i* can be *B*, *D* or *E*.
The back thrust motor is also a cylinder, but the moment of inertia has to be determined about another axis. Equation (4.40) cannot be used to calculate the moment of inertia of a cylinder on its side. The next formula can be used to calculate the moment of inertia of the back thrust motor.

$$J_E = \frac{m}{12}\left(length_E^2 + 3 \cdot radius_E^2\right) + ml_{zE}^2 \qquad (4.41)$$

In table 4.4 is calculated the hovercraft inertia about the z-axis by summarising the moments of inertia values of the separated hovercraft parts *A* to *E*.

	moment of inertia [kg.m^2]
Battery pack, A	0,0116
Lift motor, B	1,125^e-5
Main plate, C	0,0642
Camera, D	0,0036
Back thrust motor, E	0,0055
Hovercraft inertia	0,0849

table 4.4: inertia calculation analytical

The analytical value for the moment of inertia is 0,0849 kg.m^2. Comparing this value with the obtained empirical value, it is only 10% lower. Taking into account, that the analytical value is a rough calculation, it is only more acceptable to use the empirical value as a modelling parameter for the moment of inertia. The analytical value will only get nearer to the empirical value if the calculation is more exact, because there are a few other components on the hovercraft that are not taken into account. These components, such as rudder servo and speed controller, cause a non-uniform distribution in the main plate that leads to an increase of the analytical moment of inertia.

4.3 Simulation results

Simulation of the hovercraft model is done using Matlab 6.1. The used m-files for simulation of the hovercraft model are added in appendix N, O and P. Modelsim.m contains the model information and model.m contains solving and plotting orders. Shipplot is only used to plot hovercraft shapes on a trajectory in the earth-fixed reference frame.
The simulation results of one simulation are presented below. This simulation starts with maximum lift force and a rudder angle of 0°. After 5 seconds the rudder turns to -10° and stays there till the end of the simulation t=60s. These values are maintained during 45 seconds and from that time the lift force is lowered half for 10 seconds. At t=50s the lift force is lowered to 0 till the end of the simulation t=60s. The results are presented in figure 4.7 and 4.8. In figure 4.7 can be seen the velocities in surge, sway, yaw and the movement of the hovercraft in earth coordinates. The plots of the velocities contain also the input variables *Fl* (lift force) and *Rudder* (rudder angle). Figure 4.8 gives an indication of the orientation of the hovercraft on the trajectory and also is indicated the intervals of input change. The red interval starts at t=5s and ends at t=40s. At t=40s is started lowering the lift force, this is where the color black starts. Keep in mind that the vertical axis in figure 4.8 and 4.7 (movement of the hovercraft in earth coordinates) normally is directed in the opposite direction. The hovercraft is still turning in negative direction about the z-axis.
Balancing the hovercraft with small masses of lead is done, before starting validation of the simulation results. The hovercraft is not stable at all if no balancing masses are used. Placing masses on the body of the hovercraft is done by trial and error. Validation of the simulation is also not easy, because validation experiments in real with the hovercraft require some free space with no obstacles. A few validation experiments are done to get an indication of the characteristics of motion. One of them is a maximum velocity experiment in surge direction. As mentioned earlier this is done using a digital camera and some additional software. With the software and markers on the surface it is possible to determine the covered distance between two frames. By means of the distance and the frame rate can be obtained an indication of the maximum speed of the hovercraft in surge direction. This speed is estimated on 3,0 m/s and turned out to be a good validation value for the simulation. The second validation experiment is done in a bigger room at the school of electrical engineering. This room was actually still to small, but with the lift motor on half power and a rudder angle of 10° it was possible to see some characteristics of the hovercraft in motion. Releasing the hovercraft with the settings above resulted in figure 4.9. This figure can be used for a rough validation of some characteristics of motion and nothing more than that, because the camera is not standing still and it is also standing under an angle. It can be seen that the hovercraft is always directed to the inside of the turning circle during the

experiment as well as during the simulation. The trajectory during validation is similar to the simulation trajectory; first a large turn and after that the hovercraft goes to a reasonable constant turning circle. The difference can be attributed to the use of another lift force (½ F_l) and the non-uniform mass distribution. A non-uniform mass distribution leads to a smaller turning circle, because the heavier front of hovercraft touches just the surface at half lift power.

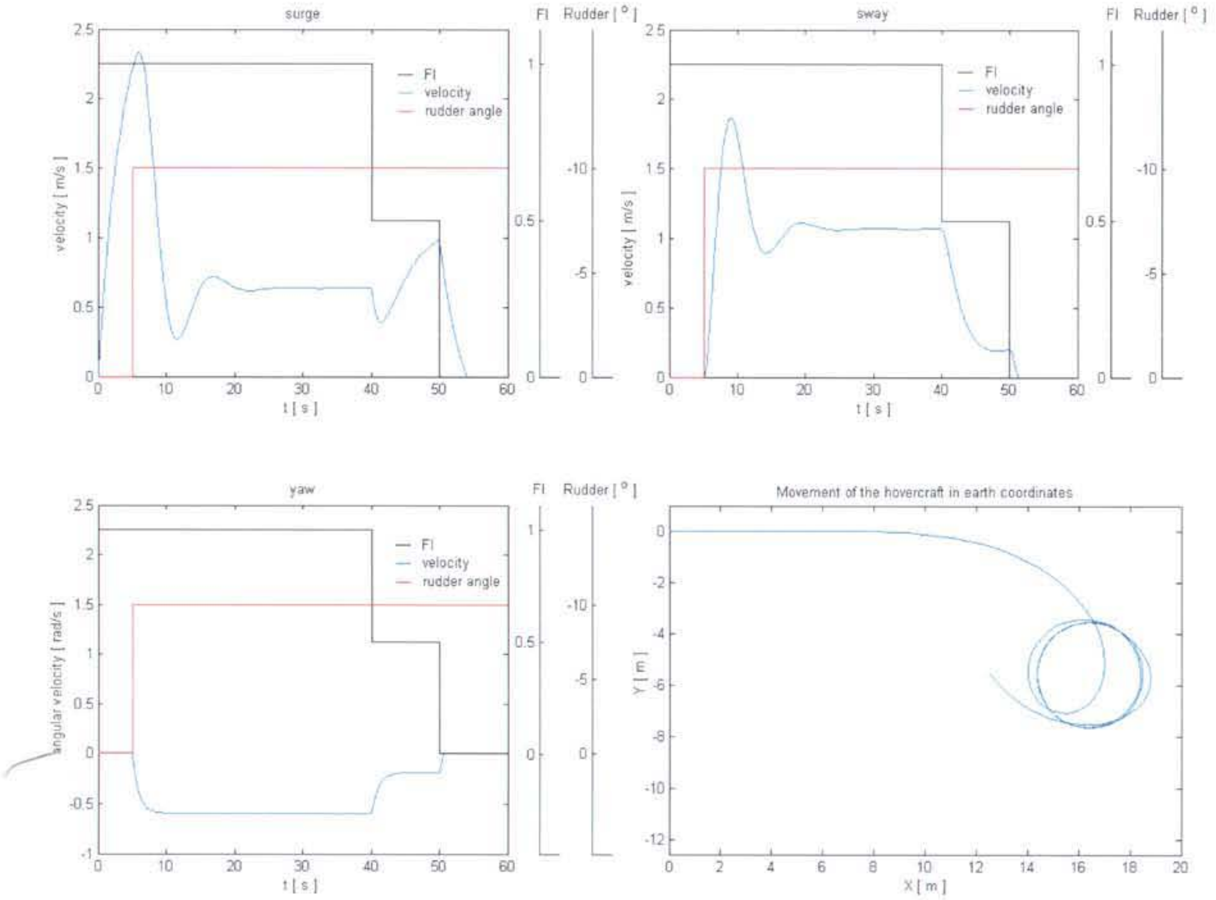


figure 4.7:Simulation results

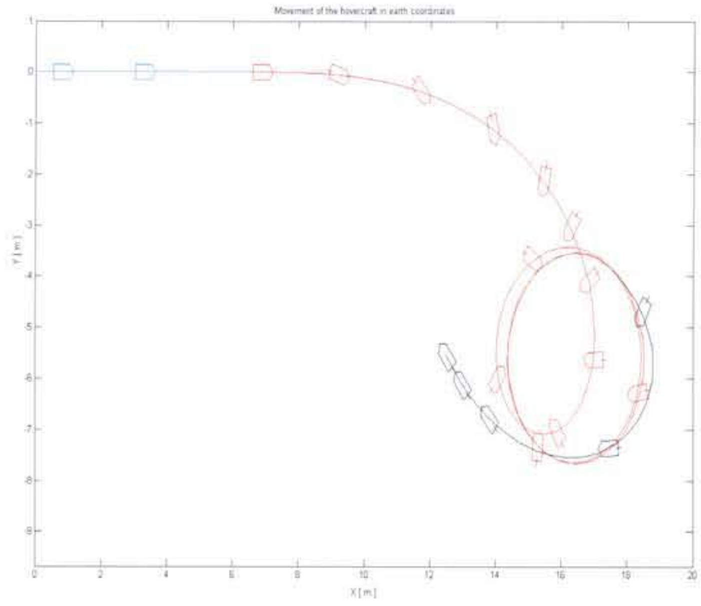


figure 4.8:Hovercraft orientation plot

With these validation results it can be said that the hovercraft model is acceptable for future use. A point of improvement in the hovercraft model is the non-uniform mass dependency effects. Non-uniform mass distribution effects become more dominant when the lift force is lowered. It is not necessary to improve the model if a better mass distribution can be created in the new hovercraft.

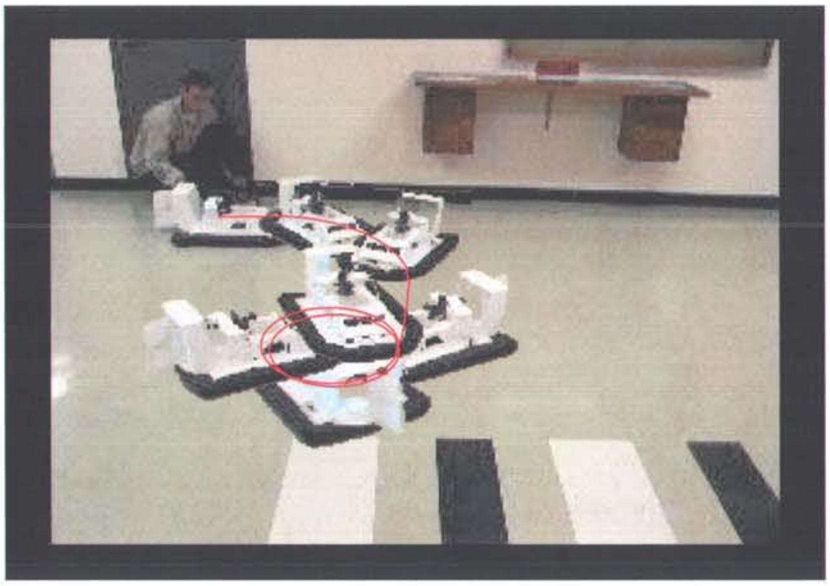


figure 4.9: Validation experiment

5 Recommendations & conclusions

Control engineers are always looking for new challenges and this under actuated model hovercraft is a challenge for control engineers, but before control techniques can be implemented the hovercraft has to be made controllable by PC. Achieving a connection between hovercraft and PC is one aim of this traineeship. Open loop control by PC is made possible and the control program for it makes it already easier to implement a controller in future. Vision and control will be used in future to control the hovercraft. Therefore are used wireless X10 cameras, one camera is placed already on the hovercraft and another camera has to be suspended to overlook the room. The camera on the hovercraft can be used for determining the orientation and the suspended camera can be used for determining the position of the hovercraft. Besides a stabilizing controller has to be designed a driver, that supports the X10 VA11A video usb adapter, to complete the control scheme presented in figure 5.1. Conceivable aims with vision and control can be following a straight line on the ground or going to a marker on a wall.

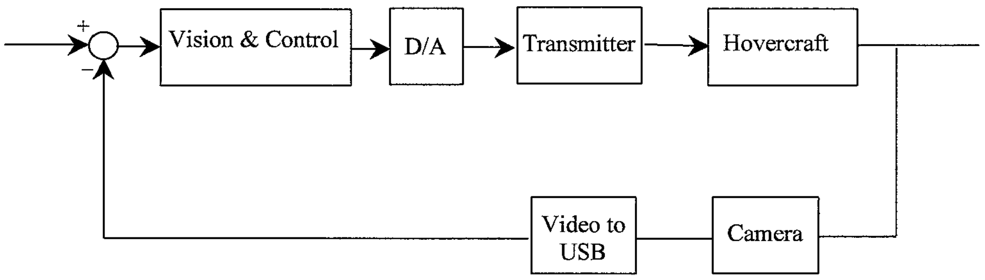


Figure 5.1: Control Scheme

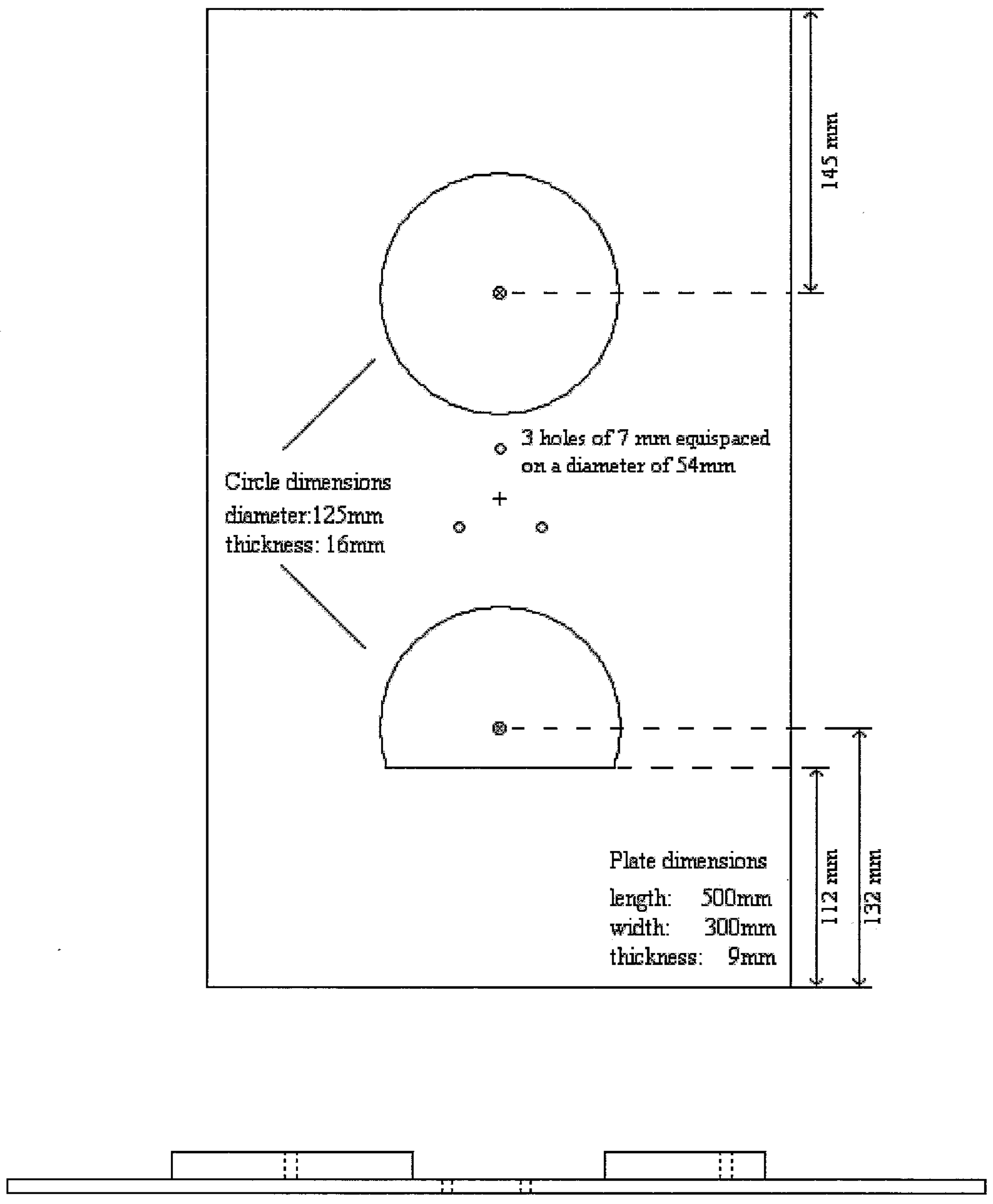
The second aim of this project was deriving a dynamical model for the hovercraft. A 3 DOF dynamical model is derived with Newton-Euler and is simulated in Matlab. In this model was the moment of inertia about the z-axis an unknown parameter. A few additional experiments at the school of Mechanical Engineering are done with success to determine a value for the moment of inertia.
Simulation results in Matlab with the 3 DOF model are validated as good as possible, because space to do validation experiments in real is a problem. The model is answering the expectations, but can be improved on one point. In the model hovercraft is a non-uniform mass distribution that leads to an unwanted behaviour if the lift force is lowered. The hovercraft touches namely the ground in front, because of the heavy two battery packs. A more uniform mass distribution gives a better behaviour of the hovercraft in motion. For the damping coefficients d_2 and d_3 and the friction coefficients μ_1, μ_2 and μ_3 are chosen reasonable values, but for future aspects are maybe more exact values required.
A few improvements can be made with respect to the current hovercraft design. These improvements can be taken into account for building the new hovercraft in the near future. Materials for constructing a hovercraft need to be as light and rigid as possible, these properties improve also the balancing problem. It is very difficult to balance the hovercraft, for example the battery pack can never be placed back on exact the same place after recharging. This will lead to a shift of mass inside the hovercraft. The battery pack is 30% of the total mass of the hovercraft, so this is substantial. All components need to be attached at one and the same place each time. Another problem is that the battery pack causes a non-uniform mass distribution. Two solutions are possible for this problem. The first one solves the problem by moving one 6-cell battery pack to the back of the hovercraft. Using 2-takt fuel engines is the second option it reduces the weight also.
Current hovercraft had initially not a good place for the wireless camera, but between the lift motor and the back thrust motor is created a plateau for the camera. Another optional place is on the back wing, but it is not rigid enough in this design to carry the camera. It is better to place the camera on the back wing if it is possible in the new design, because a better mass distribution will be created.

Bibliography

Nise, Norman S., *Control Systems Engineering*, Second edition. MP: Addison-Wesley, 1995
Ogata, K., *Modern Control Engineering.* Englewood Cliffs, NJ: Prentice-Hall, 1970
Franklin, G.F., Powell, J.D., Emami-Naeini, A., *Feedback Control of Dynamic Systems*, Third edition. MA: Addison-Wesley, 1994.
Fossen, Thor I., *Guidance and Control of Ocean Vehicles*. Chichester, England: John Wiley & Sons, 1994.
Yun, L., Bliault, A., *Theory and design of Air Cushion Craft*. London: Arnold, 2000.

Appendix A: Medium Density Fibreboard test plate

Top view and side view of the MDF plate



Appendix B: rtmodule.c

```
#include <rtl.h>
#include <pthread.h>
#include "mbuff.h"
#include "cloop.h"
#include "cdsm.h"

void *sample_code(void *arg);
void *control_code(void *arg);
void *dtoa_code(void *arg);

pthread_t sample_thread;
pthread_t control_thread;
pthread_t dtoa_thread;

volatile cloop_t *cloop;

/* module initialisation */
int init_module(void)
{
   int module_status=0;
   float a,b;

   /* initialise shared memory */
   cloop = (volatile cloop_t*) mbuff_alloc("control_loop",sizeof(cloop_t));
   if (cloop == NULL) {
      rtl_printf("Shared Memory Allocation Problem : Returning\n");
      return -1;
   }

   CDSM_init();

   if (1) {
      cloop_init(cloop);
      cloop->simulation = 0;
   }
   a=lookup_sat(5);
   b=lookup_sat(3);
   io_dacout(cloop,0,a/5*4095);
   io_dacout(cloop,1,b/5*4095);

   module_status = pthread_create(&sample_thread,NULL,sample_code,0);
   if (module_status != 0) {
      rtl_printf("Thread initialisation failed: sample status
                 %d\n",module_status);
      return module_status;
   }

   module_status = pthread_create(&control_thread,NULL,control_code,0);
   if (module_status != 0) {
      rtl_printf("Thread initialisation failed: control status
                 %d\n",module_status);
      return module_status;
   }



   module_status = pthread_create(&dtoa_thread,NULL,dtoa_code,0);
   if (module_status != 0) {
      rtl_printf("Thread initialisation failed: dtoa status
                 %d\n",module_status);
      return module_status;
   }
   return 0;
}

/* module destroy */
void cleanup_module(void)
{
   pthread_delete_np(sample_thread);
   pthread_delete_np(control_thread);
   pthread_delete_np(dtoa_thread);
   mbuff_free("control_loop", (void *)cloop);
   CDSM_done();
}

/* sampling thread code */
void *sample_code(void *arg)
{
   struct sched_param p;
   hrtime_t now;
   long interval;

   now = gethrtime();

   p.sched_priority = 1;
   pthread_setschedparam(pthread_self(),SCHED_FIFO,&p);
   pthread_setfp_np(pthread_self(),1);
   interval = (long)(cloop->dt * 1000000000.0);
   pthread_make_periodic_np(pthread_self(),now,interval);

   while(1) {
      pthread_wait_np();

      if (cloop->cstate) sample(cloop);

      //rtl_printf("debug message : %d \t %d\n",cloop->cstate,
                   cloop->calibrate_count);
      pthread_wakeup_np(control_thread);
   }

   return 0;
}

/* controller thread code */
void *control_code(void *arg)
{
   struct sched_param p;

   p.sched_priority = 1;
   pthread_setschedparam(pthread_self(),SCHED_FIFO,&p);
   pthread_setfp_np(pthread_self(),1);

   while(1) {
      pthread_suspend_np(pthread_self());

      // run 'control' if not in stop mode
      if (cloop->cstate) control(cloop);
```

```
        //rtl_printf("debug message : control thread\n");
        pthread_wakeup_np(dtoa_thread);
    }

    return 0;
}


/* dtoa thread code */
void *dtoa_code(void *arg)
{
    struct sched_param p;

    p.sched_priority = 1;
    pthread_setschedparam(pthread_self(),SCHED_FIFO,&p);
    pthread_setfp_np(pthread_self(),1);

    while(1) {
        pthread_suspend_np(pthread_self());

              if (cloop->cstate) dtoa(cloop);

        //rtl_printf("debug message : dtoa thread\n");

    }

    return 0;
}
```

Appendix C: cloop.h

```
#define MAX_DATA 1000       // Maximum data points
#define DEF_INTERVAL 0.1
#define CAL_PERIOD 3
#define IO_BASE 0x320

typedef float (fp)(float *, float, float);

/* Signal structure */
typedef struct sig {
        float y;
        float u0;
        float u1;
        float r;
        float time;
} sig_t;


/* Control Loop Structure */
typedef struct {
      sig_t cloop_sig[MAX_DATA];      // control loop signals
      int time_index, last;           // array index and oldest data
      int num_data;                   // number of valid data
      int sig_low, sig_hi;            // signal limits
      float dt;                       // sampling interval
      float param[20];                // control gains, 0-P, 1-I, 2-D, 3-N,
                                         4-low, 5-hi
      int num_params;                 // number of control gains
      float x[20];                    // state variables
      float setpt0, setpt1;
      float ramp_increment;           // step value and ramp increment for
                                         setpoint
      int rampcount;                  // counter for ramp input
      int input;                      // 0-step, 1-ramp
      int adc_chan;                   // A/D channel
      int inchannel;
      int dac_chan;                   // D/A channel
      int filter;                     // low pass filter  coefficient
      int initialised;                // 0-uninitialised, 1-initialised
      int simulation;                 // 0-process, 1-simulation
      int cmode;                      // 0-Manual, 1-PID, 2-OTHER, 3-OTHER
      int cstate;                     // 0-stop, 1-start
      int idle_tag;
      int datalog;                    // 0-not data logging, 1-data logging
      int datastart;                  // time to start data logging
      int dataend;                    // time to end data logging
} cloop_t;


/* signal initialisation function */
void sig_init(volatile sig_t *signal);

/* copy signal */
void sig_copy(volatile sig_t *source, volatile sig_t *dest);



/* Functions for the cloop structure */
/* Initialise the cloop structure */
void cloop_init(volatile cloop_t *cloop);

/* Function to clear the cloop */
void cloop_clear(volatile cloop_t *cloop);

/* Function to reset signal nodes */
void cloop_reset(volatile cloop_t *cloop);

/* Function to sample/scale and store data */
void sample(volatile cloop_t *cloop);

/* Calculate the control */
void control(volatile cloop_t *cloop);

/* D/A function - send control to D/A */
void dtoa(volatile cloop_t *cloop);

/* Set the controller gains */
void set_params(volatile cloop_t *cloop, float *params, int num_params);

/* Function to sample from the A/D */
int io_adcin(volatile cloop_t *cloop, int channel);

/* Function to send data to D/A */
int io_dacout(volatile cloop_t *cloop, int chan, int data);

/* Scale a 12 bit int between low and hi */
float scale_itof(int data, int ilow, int ihi, float low, float hi);

/* Asking function for saturation range values */
float lookup_sat(int i);

/* Scale a float between low and hi to a 12bit int */
int scale_ftoi(float data, float low, float hi, int ilow, int ihi);

/* Filter data using a coeeficient of a */
float filter_data(float ydata, float udata, float a);

/* Function to store data in `fname' */
int store_log(volatile cloop_t *cloop, char *fname);

/* Setup counters on ax5411 card */
void set_io_counters(unsigned int n1, unsigned int n2);

/* Setup the IO (ax5411) card */
int setup_io(volatile cloop_t *cloop);

/* Insert a new node in the circular buffer */
void insert_new(volatile cloop_t *cloop);

/* Write a signal into the newest node of the buffer */
void store_data(volatile cloop_t *cloop, sig_t new_sig);

/* Read of the last `num' signal nodes from the buffer */
int read_data(volatile cloop_t *cloop, sig_t *data_store, int num);


/* Pull of the signal with delay `delay' */
sig_t get(volatile cloop_t *cloop, int delay);
```

```
/* Display signal nodes */
void display(volatile cloop_t *cloop, int num);

/* Function to calculate manual control */
float manual(volatile cloop_t *cloop);

/* Function to calculate pid control */
float pid(volatile cloop_t *cloop);

/* state equation 1 for pid control */
float pidf1(float *params, float x, float u);

/* state equation 4 for pid control */
float pidf4(float *params, float x, float u);

/* Function to implement saturation */
float sat(float x, float xlo, float xhi);

/* 3rd Runga-Kutta integration */
float rk3(volatile cloop_t *cloop, fp f, float x, float u, float dt);
```

Appendix D: cloop.c

```
#ifdef __KERNEL__
#ifdef __RTL__
#include <rtl_printf.h>
#endif /* __RTL__ */
#else
#include <stdio.h>
#endif /* __KERNEL__ */
#include <sys/io.h>
#include <linux/ioport.h>
#include "cloop.h"

/* Functions to operate on the cloop structure */

/* signal initialisation function */
void sig_init(volatile sig_t *signal)
{
   signal->y = 0.0;
   signal->u0 = lookup_sat(5);
   signal->u1 = lookup_sat(3);
   signal->r = 0.0;
   signal->time = 0.0;
}

void sig_copy(volatile sig_t *source, volatile sig_t *dest)
{
   dest->y = source->y;
   dest->u0 = source->u0;
   dest->u1 = source->u1;
   dest->r = source->r;
   dest->time = source->time;
}

/* initialisation function (default) */
void cloop_init(volatile cloop_t *cloop)
{
   int i;

   for (i=0 ; i<MAX_DATA ; i++) sig_init(&(cloop->cloop_sig[i]));
   cloop->time_index = 0; cloop->last = 0; cloop->num_data = 0;
   cloop->dt = DEF_INTERVAL;

   cloop->sig_low = -5; cloop->sig_hi = 5;
   for (i=0 ;  i < 20 ; i++) {
      cloop->x[i] = 0.0;
      cloop->param[i] = 0.0;
   }
   cloop->setpt0 = lookup_sat(5);
   cloop->setpt1 = lookup_sat(3);
   cloop->ramp_increment = 0.0; cloop->rampcount = 0;
   cloop->input = 0;
   cloop->filter = 0.5;
   cloop->adc_chan = cloop->dac_chan = 0;
   cloop->inchannel = 0;
   cloop->cmode = 0;
   cloop->cstate = 0;
   cloop->initialised = 1;
   cloop->simulation = 0;
   cloop->datalog = 0;
```

```
    cloop->datastart = 0;
    cloop->dataend = -1;
    setup_io(cloop);
}

/* reset cloop signal values */
void cloop_reset(volatile cloop_t *cloop)
{
    int i;
    for (i=0 ; i<MAX_DATA ; i++) sig_init(&(cloop->cloop_sig[i]));
}

/* clear cloop signal values */
void cloop_clear(volatile cloop_t *cloop)
{
    cloop_reset(cloop);
    cloop->time_index = 0; cloop->last = 0; cloop->num_data = 0;
    cloop->datalog = 0; cloop->datastart = 0; cloop->dataend = -1;
    cloop->rampcount = 0;
}

/* Set the controller gains */
void set_params(volatile cloop_t *cloop, float *params, int num_params)
{
    int i;

    for (i=0; i < num_params ; i++) cloop->param[i] = params[i];
    cloop->num_params = num_params;
}

/* 3rd order Runge-Kutte integration */
float rk3(volatile cloop_t *cloop, fp f, float x, float u, float dt)
{
    int j;
    float s1, s2, s3, xx;
    float params[20];

    for (j = 0; j < 20; j++) params[j] = cloop->param[j];
    s1 = f(params,x,u);
    xx = x + dt * s1;
    s2 = f(params,xx,u);
    xx = x + dt * (s1 + s2) / 4.0;
    s3 = f(params,xx,u);
    xx = x + (s1 + 4.0 * s3 + s2) * dt / 6.0;
    return xx;
}

/* Function to sample A/D */
int io_adcin(volatile cloop_t *cloop, int channel)
{
    int low = 0, hi = 0, temp = 0, i;
    int data = 0, chan;

         chan = (channel) & 0x000F;

         chan = chan + (chan << 4);

         // Select our channel by writing our value to the MUX
         //  ==> we start and finish on the same channel

    for (i=0; i<200; i++) outb(chan, IO_BASE+2);
```

```
        // clear to A/D register 1st
        outb(0, IO_BASE);

        // wait until the A/D conversion is complete (shouldn't
        // be necessary
        while (inb(IO_BASE+8) & 0x80) {}

    temp = inb(IO_BASE);
    low = (temp >> 4) & 0x000F;
    hi = (inb(IO_BASE+1) << 4) & 0x0FF0;
    data = (hi | low); /* 12 bits */

    outb(0, IO_BASE+9);
    return data;
}

/* Function to send data to D/A */
int io_dacout(volatile cloop_t *cloop, int chan, int data)
{
    unsigned int low = (data << 4) & 0x00f0;
    unsigned int hi = (data >> 4) & 0x00ff;

    if( data < 0 || 4095 < data){
       #ifdef __RTL__
       rtl_printf("ax5411: dac out value (%d) out of range\n",data);
       #else
       printf("ax5411: dac out value (%d) out of range\n",data);
       #endif
       return -1;
    }

    switch(chan) {
       case 0:
          outb(low, IO_BASE+4);
          outb(hi, IO_BASE+5);
          break;

       case 1:
          outb(low, IO_BASE+6);
          outb(hi, IO_BASE+7);
          break;

       default:
          return -1;
          break;
    }
    return 0;
}



/* Scale a 12 bit int between low and hi */
float scale_itof(int data, int ilow, int ihi, float low, float hi)
{
    return ((((float)data - ilow) / (ihi - ilow)) * (hi - low) + low);
}

/* Asking function for saturation range values */
float lookup_sat(int i)
{
   float a=0;

   if(i == 1)
       a=2.10;  // lowest value channel1
   if(i == 2)
    a=3.03;  // highest value channel1
   if(i == 3)
    a=2.56;  // init value channel1
   if(i == 4)
    a=2.45;  // lowest value channel0 (max throttle)
   if(i == 5)
    a=2.80;     // highest/init value channel0

   return a;
}

/* Scale a float between low and hi to a 12bit int */
int scale_ftoi(float data, float low, float hi, int ilow, int ihi)
{
   float temp;
   float il, ih;

   il = (float) ilow; ih = (float) ihi;
   temp = ((data - low) / (hi - low)) * (ih - il) + il;
   return (int)temp;
}

/* Filter data using coefficient a */
float filter_data(float ydata, float udata, float a)
{
   return ((1-a)*udata + a*ydata);
}

/* Function to implement saturation */
float sat(float x, float xlo, float xhi)
{
   return (x < xlo) ? xlo : ((x > xhi) ? xhi : x);
}

/* Setup counter on ax5411 card */
void set_io_counters(unsigned int n1, unsigned int n2)
{
   /* load counter 2 */
   outb(0xb4, IO_BASE+15);
   outb((n2 & 0x00ff), IO_BASE+14);
   outb((n2 >> 8), IO_BASE+14);

   /* load counter 1 */
   outb(0x74, IO_BASE+15);
   outb((n1 & 0x00ff), IO_BASE+13);
   outb((n1 >> 8), IO_BASE+13);
}




/* Setup the IO (ax5411) card */
int setup_io(volatile cloop_t *cloop)
{
   int ad_chan;
   unsigned int n1=0, n2=0;
   unsigned int temp, temp1=1;

   /* request access to the i/o region */
#ifdef __KERNEL__
   release_region(IO_BASE, 16);
   request_region(IO_BASE, 16, "AX5411");
#endif
#ifndef __KERNEL__
   ioperm(IO_BASE,16,1);
#endif

   /* reset control and status */
   outb(0, IO_BASE+9);
   outb(0, IO_BASE+8);

   temp = (unsigned int) (1000000.0 * cloop->dt);
   while (temp > temp1) {
      n1 = temp; n2 = temp1;
      temp = (unsigned int) (temp / 5);
      temp1 = temp1 * 5;
   };
   cloop->dt = n1*n2 / 1000000.0;

   set_io_counters(n1, n2);
   outb(0x00D3, IO_BASE+9);

   ad_chan = (cloop->adc_chan << 4) & 0x00F0;

   /* set mux control for channel */
   outb(ad_chan, IO_BASE+2); // conversion for channel 0-chan

   return 0;
}

/* Insert a new node into the circular buffer */
void insert_new(volatile cloop_t *cloop)
{
   int old;

   old = cloop->time_index;
   cloop->time_index = (cloop->time_index + 1) % MAX_DATA;

   if (cloop->time_index == cloop->last) {
      cloop->num_data--;
      cloop->last = (cloop->last + 1) % MAX_DATA;
   }
   cloop->cloop_sig[cloop->time_index].time = cloop->cloop_sig[old].time;
   cloop->cloop_sig[cloop->time_index].y = cloop->cloop_sig[old].y;
   cloop->cloop_sig[cloop->time_index].u0 = cloop->cloop_sig[old].u0;
   cloop->cloop_sig[cloop->time_index].u1 = cloop->cloop_sig[old].u1;
   cloop->cloop_sig[cloop->time_index].r = cloop->cloop_sig[old].r;

   cloop->num_data++;
}


/* Write a new signal into the newest node of the circular buffer */
void store_data(volatile cloop_t *cloop, sig_t new_sig)
{
   sig_copy(&new_sig, &(cloop->cloop_sig[cloop->time_index]));
```

```
}

/* Read off the oldest `num' signal nodes from the end of the buffer */
int read_data(volatile cloop_t *cloop, sig_t *data_store, int num)
{
   int i;

   if (num > cloop->num_data) {
      #ifdef __RTL__
      rtl_printf("Not enough data\n");
      #endif
      return 0;
   }

   for (i = 0; i < num; i++) {
      sig_copy(&(cloop->cloop_sig[cloop->last]), &(data_store[i]));
      cloop->last = (cloop->last + 1) % MAX_DATA;
      cloop->num_data--;
   }

   return num;
}

/* Pull off the signal with delay `delay' */
sig_t get(volatile cloop_t *cloop, int delay)
{
   int index;
   sig_t temp;

   sig_init(&temp);
   if (cloop->num_data == 0) {
      #ifdef __RTL__
      rtl_printf("Not enough data\n");
      #endif
      return temp;
   }

   index = cloop->time_index - delay;
   if (index < 0) index += MAX_DATA;
   sig_copy(&(cloop->cloop_sig[index]), &temp);
   return temp;
}

/* Display `num' past signal values */
void display(volatile cloop_t *cloop, int num)
{
   int i;
   sig_t temp;

   for (i=0;i<num;i++) {
      temp = get(cloop,num-i-1);
#ifndef __KERNEL__
      printf("y : %d \t y1 : %d \t u : %d \t r : %d \t time : %d
             \n",temp.y,
      temp.u0,temp.u1,temp.r,temp.time);
#endif
#ifdef __RTL__
      rtl_printf("y : %d \t y1 : %d \t u : %d \t r : %d \t time : %d
       \n",(int)(100*temp.y),
      (int)(100*temp.u0),(int)(100*temp.u1),(int)(100*temp.r),
      (int)temp.time);
#endif
    }
}

/* function to log data to a file `fname' */
int store_log(volatile cloop_t *cloop, char *fname)
{
#ifndef __KERNEL__
    int i, start, end;
    FILE *fd_open;
    int fd_write;
    sig_t temp;
    float u0,u1,y,r;
    float time;
    float oldu,oldy;

    start = cloop->datastart;
    if (cloop->dataend < cloop->datastart) end = cloop->num_data-1;
    else end = cloop->dataend;

    fd_open = fopen(fname, "w+");
    if (fd_open == NULL) {
        printf("Error opening file \n");
        return -1;
    }

    for ( i=start ; i<=end ; i++) {
        temp = get(cloop,end-i);
        time = temp.time; u0 = temp.u0; y = temp.y; u1 = temp.u1
       ;r = temp.r;
        fprintf(fd_open, "%4.2f \t %4.3f \t %4.3f\t %4.3f\t
       %4.3f\n",time,r,y,u0,u1);
    }

    fclose(fd_open);
#endif
    return 0;
}
```

Appendix E: thread_code.c

```
#include <rtl_printf.h>
#include "cloop.h"
#include "cdsm.h"

/* Function to sample/scale and store data */
/* Called directly from 'sample' thread */
void sample(volatile cloop_t *cloop)
{
   int data=0;
   float sdata, fdata=0.0, fdata1=0.0;
   static float yold, y1old;
   float setpt=0.0;
   sig_t new_signal;


   /* get copy of old data */
   new_signal = get(cloop,0);

   /* create space for new data */
   insert_new(cloop);


   if (cloop->simulation) {
      fdata = 0.95*new_signal.y + 0.05*new_signal.u0;
      //fdata = yold + 0.35*fdata;
   }

   else {
      data = io_adcin(cloop,0);
      sdata = scale_itof(data, 0, 4096, -10.0, 10.0);
      fdata = filter_data(yold, sdata, cloop->filter);

      data = io_adcin(cloop,1);
      sdata = scale_itof(data, 0, 4096, -10.0, 10.0);
      fdata1 = filter_data(y1old, sdata, cloop->filter);
   }

   // update old values
   yold = fdata; y1old = fdata1;


   /* update setpoint */
   switch (cloop->input) {
   case 0: setpt = cloop->setpt0;
      break;
   case 1: setpt = new_signal.r + cloop->ramp_increment;
      if (setpt >= 1.0) {
         setpt = -1.0;
         cloop->rampcount++;
      }
      if (cloop->rampcount == 5) setpt = 0;
      break;
   default: break;
   }
```

```
    /* update new signal */
    new_signal.y = fdata;
    new_signal.r = setpt;
    new_signal.time += cloop->dt;

    /* store in CDSM */
    CDSM_set(0, (long)(100*fdata));
    CDSM_set(1, (long)(100*setpt));

    /* store the value in y buffer */
    store_data(cloop, new_signal);

}

/* Calculate the control */
/* Called directly from 'control' thread */
void control(volatile cloop_t *cloop)
{
    float cont0 = 0.0, cont1 = 0.0;
    float a,b,c,d;

    // define some temporary static variables to use for memory
    //static float x[10];

    // Declare a 'signal' node. Each node has setpoint (r), output (y),
    // input (u), and time (time) elements. To retrieve a node, you use
    // the 'get' function. That is, 'get(cloop,delay)' where delay is
    // the signal delay in discrete-time. Example: to retrieve the control
    // data from 2 samples back use 'get(cloop,2).u'

    sig_t curr_signal;

    // we are in run mode
    // control mode: 0 - manual control
    // control mode: 1 - PID control
    // can simply add in another case statement and define another control
       scheme
    // the control action is returned to the variable 'cont'
    switch (cloop->cmode) {
       case 0: cont0 = cloop->setpt0;
               cont1 = cloop->setpt1;
          break;
       case 1: cont0 = pid(cloop);
          break;
       default: break;
    }

#ifdef __RTL__
    rtl_printf("mode = %d %d\n",cloop->cmode,(int)(100*cont0));
#endif

    /* apply signal limits */
    a=lookup_sat(1);
    b=lookup_sat(2);
    c=lookup_sat(4);
    d=lookup_sat(5);
    cont0 = sat(cont0, c,d);
    cont1 = sat(cont1, a,b);


    curr_signal = get(cloop,0);
    curr_signal.u0 = cont0;
    curr_signal.u1 = cont1;

    CDSM_set(2, (long)(100*cont0));
    CDSM_set(3, (long)(100*cont1));

    store_data(cloop, curr_signal);
}

/* Function to send to dtoa */
/* Called directly by 'dtoa' thread */
void dtoa(volatile cloop_t *cloop)
{
    float ut0,ut1;
    int uint0,uint1,stat;

    ut0 = (get(cloop,0)).u0;
    uint0 = scale_ftoi(ut0, 0.0, 5.0, 0, 4096);
    stat = io_dacout(cloop, 0, uint0);
    ut1 = (get(cloop,0)).u1;
    uint1 = scale_ftoi(ut1, 0.0, 5.0, 0, 4096);
    stat = io_dacout(cloop, 1, uint1);
}

/* Function to calculate pid control */
float pid(volatile cloop_t *cloop)
{
    float kp, ti, td, N;
    sig_t curr_signal;

    N = cloop->param[3];
    kp = cloop->param[0];
    ti = cloop->param[1];
    td = cloop->param[2];
    curr_signal = get(cloop,0);


    if (td > 0.0) {
        cloop->x[4] = rk3(cloop, pidf4, cloop->x[4], curr_signal.y,
        cloop->dt);
        cloop->x[3] = curr_signal.y + N * (curr_signal.y - cloop->x[4]);
    }
    else {
        cloop->x[3] = curr_signal.y;
    }
    cloop->x[2] = kp * (curr_signal.r - cloop->x[3]);
    if (ti >= 20) cloop->x[1] = 0;
    else {
        cloop->x[1] = rk3(cloop, pidf1, cloop->x[1], cloop->x[2],
        cloop->dt);
    }

    return (cloop->x[1] + cloop->x[2]);


}



/* state equation 1 for pid control */
float pidf1(float *params, float x, float u)
```

```
{
   float temp, ti;

   ti = params[1];
   temp = ((sat(x+u,params[4],params[5]) - x) / ti);

   return temp;
}


/* state equation 2 for pid control */
float pidf4(float *params, float x, float u)
{
   float td, N;

   td = params[2]; N = params[3];
   return (10 * (u - x) );
}
```

Appendix F: hovercraft.c

```
#include <stdio.h>
#include <unistd.h>
#include <gtk/gtk.h>
#include <gtk/gtkhscale.h>
#include <gtk/gtkvscale.h>
#include "cloop.h"
#include "hovercraft.h"
#include "mbuff.h"

/* shared memory variable */
volatile cloop_t *cloop;
volatile unsigned char *cch;
volatile int *flag;

/* gui object */
guiobj cgui;

/* This callback quits the program */
gint delete_event( GtkWidget *widget, GdkEvent  *event, gpointer   data )
{
    gtk_main_quit ();
    return(FALSE);
}

/* function to initialise gui */
int gui_init(void)
{
   /* initialise gtk */
   gtk_init (0,0);

   /* init shared memory */
   cloop = (volatile cloop_t*) mbuff_alloc("control_loop",sizeof(cloop_t));
   if (cloop == NULL) {
      printf("Shared Memory Allocation Failed!\n");
      return -1;
   }
   cch = (volatile unsigned char *) mbuff_alloc("char",sizeof(char));
   flag = (volatile int *) mbuff_alloc("flag",sizeof(int));
   *flag = 0;
   /* test to see if cloop is initialised */
   if (!(cloop->initialised)) {
      printf("Control Loop Not Initialised \n");
      return -1;
   }
   cloop->cstate = 0;
   cloop->cmode = 0;


   return 0;
}

/* callback for start button */
void start_function( GtkWidget *widget, int *arg)
{
   /* start the control loop */
   cloop->cstate = 1;
}
```

```
/* callback for stop button */
void stop_function( GtkWidget *widget, GtkLabel *label)
{
   float a,b;
   char s0[14],s1[14];
   a=lookup_sat(5);
   b=lookup_sat(3);

   /* stop the control loop */
   cloop->cstate = 0;
   io_dacout(cloop,0,a/5*4095);
   io_dacout(cloop,1,b/5*4095);
   cloop->setpt0=a;
   cloop->setpt1=b;
   sprintf(s0,"UP-DOWN : %3.2f",a);
   gtk_label_set_text((GtkLabel *)(cgui.label[3]), s0);
   sprintf(s1,"LEFT-RIGHT : %3.2f",b);
   gtk_label_set_text((GtkLabel *)(cgui.label[4]), s1);

   store_log(cloop,"hcraft");
   cloop_clear(cloop);
   gtk_label_set_text(label, "Logging OFF");

}

/* callback for up button */
void up_button( GtkWidget *widget, int *arg)
{
   float setpt0 = cloop->setpt0;
   char s[14];
   float a,b;

   a=lookup_sat(4);
   b=lookup_sat(5);
   setpt0 -= 0.01;
   setpt0 = sat(setpt0,a,b);
   sprintf(s,"UP-DOWN : %3.2f",setpt0);
   gtk_label_set_text((GtkLabel *)(cgui.label[3]), s);
   cloop->setpt0 = setpt0;
   printf("u0 = %3.2f\n",setpt0);
}

/* callback for down button */
void down_button( GtkWidget *widget, int *arg)
{
   float setpt0 = cloop->setpt0;
   char s[14];
   float a,b;

   a=lookup_sat(4);
   b=lookup_sat(5);
   setpt0 += 0.01;
   setpt0 = sat(setpt0,a,b);
   sprintf(s,"UP-DOWN : %3.2f",setpt0);
   gtk_label_set_text((GtkLabel *)(cgui.label[3]), s);
   cloop->setpt0 = setpt0;
   printf("u0 = %3.2f\n",setpt0);
}
```

```
/* callback for left button */
void left_button( GtkWidget *widget, int *arg)
{
   float setpt1 = cloop->setpt1;
   char s[14];
   float a,b;

   a=lookup_sat(1);
   b=lookup_sat(2);
   setpt1 -= 0.01;
   setpt1 = sat(setpt1,a,b);
   sprintf(s,"LEFT-RIGHT : %3.2f",setpt1);
   gtk_label_set_text((GtkLabel *)(cgui.label[4]), s);
   cloop->setpt1 = setpt1;
   printf("u1 = %3.2f\n",setpt1);
}


/* callback for right button */
void right_button( GtkWidget *widget, int *arg)
{
   float setpt1 = cloop->setpt1;
   char s[14];
   float a,b;

   a=lookup_sat(1);
   b=lookup_sat(2);
   setpt1 += 0.01;
   setpt1 = sat(setpt1,a,b);
   sprintf(s,"LEFT-RIGHT : %3.2f",setpt1);
   gtk_label_set_text((GtkLabel *)(cgui.label[4]), s);
   cloop->setpt1 = setpt1;
   printf("u1 = %3.2f\n",setpt1);
}


/* callback for data log button */
void data_log_function( GtkWidget *widget, GtkLabel *label)
{
   /* start datalog flag */
   if (cloop->datalog) {
      cloop->datalog = 0;
      cloop->dataend = cloop->time_index;
   }
   else {
      cloop->datalog = 1;
      cloop->datastart = cloop->time_index;
   }


   if (cloop->datalog) gtk_label_set_text(label, "Logging ON");
   else gtk_label_set_text(label, "Logging OFF");

}



/* callback for quit button */
void quit_function( GtkWidget *widget, int *arg)
```

```
{
   float a,b;
   a=lookup_sat(5);
   b=lookup_sat(3);

   /* stop the control loop */
   cloop->cstate = 0;
   io_dacout(cloop,0,a/5*4095);
   io_dacout(cloop,1,b/5*4095);
   gtk_idle_remove( cloop->idle_tag );
   store_log(cloop,"hcraft");
   cloop_clear(cloop);
   gtk_main_quit ();

}

/* idle function */
int hover_idle(GtkLabel *label)
{
   /* sig_t curr;
   char s0[4],s1[4];
   GtkLabel *label0, *label1;

   if (*flag) {
      label0 = (GtkLabel *) (cgui.label[3]);
      label1 = (GtkLabel *) (cgui.label[4]);
      curr = get(cloop,0);
      if (*cch == 'i') curr.u0 += 0.1;
      if (*cch == 'm') curr.u0 -= 0.1;
      if (*cch == 'j') curr.u1 -= 0.1;
      if (*cch == 'l') curr.u1 += 0.1;

      curr.u0 = sat(curr.u0,0.0,5.0);
      curr.u1 = sat(curr.u1,0.0,5.0);
      sprintf(s0,"%3.2f",curr.u0);
      gtk_label_set_text(label0, s0);
      sprintf(s1,"%3.2f",curr.u1);
      gtk_label_set_text(label1, s1);

      printf("Char = %c \t %f\t %f\n",*cch,curr.u0,curr.u1);

      store_data(cloop, curr);
      *flag = 0;
      }*/
   return TRUE;
}

/* gui for lab */
void gui_lab( GtkWidget *window)
{

   /* remove initial window */
   /* Create the new window */
   cgui.window[0] = gtk_window_new (GTK_WINDOW_TOPLEVEL);
   gtk_widget_set_usize (cgui.window[0],500,300);
   gtk_widget_set_uposition (cgui.window[0], 0, 0);


   /* Set the window title */
   gtk_window_set_title (GTK_WINDOW (cgui.window[0]), "HOVERCRAFT");
```

```
/* Set a handler for delete_event that immediately exits GTK. */
gtk_signal_connect(GTK_OBJECT(cgui.window[0]), "delete_event",
GTK_SIGNAL_FUNC (delete_event), NULL);

/* Sets the border width of the window. */
gtk_container_set_border_width (GTK_CONTAINER (cgui.window[0]), 20);

/* Create a 5x1 table */
cgui.table[0] = gtk_table_new (6, 7, TRUE);
gtk_table_set_row_spacings (GTK_TABLE (cgui.table[0]), 0);
gtk_table_set_col_spacings (GTK_TABLE (cgui.table[0]), 25);


/* Put the table in the main window */
gtk_container_add (GTK_CONTAINER (cgui.window[0]), cgui.table[0]);

/* create controller label */
cgui.label[0] = gtk_label_new("HOVERCRAFT CONTROL");
gtk_label_set_justify(GTK_LABEL(cgui.label[0]),GTK_JUSTIFY_LEFT);
gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
cgui.label[0], 1, 5, 0, 1); gtk_widget_show (cgui.label[0]);

/* include seperator */
cgui.separator = gtk_hseparator_new ();
gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
cgui.separator, 0, 6, 1, 2); gtk_widget_show(cgui.separator);


/* put in buttons at bottom - up button */
cgui.button = gtk_button_new_with_label ("UP");
gtk_signal_connect(GTK_OBJECT (cgui.button), "clicked",
GTK_SIGNAL_FUNC (up_button), NULL);
gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
cgui.button, 2, 3, 2, 3); gtk_widget_show (cgui.button);

/* put in buttons at bottom - down button */
cgui.button = gtk_button_new_with_label ("DOWN");
gtk_signal_connect(GTK_OBJECT (cgui.button), "clicked",
GTK_SIGNAL_FUNC (down_button), NULL);
gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
cgui.button, 2, 3, 4, 5); gtk_widget_show (cgui.button);

/* put in buttons at bottom - left button */
cgui.button = gtk_button_new_with_label ("LEFT");
gtk_signal_connect(GTK_OBJECT (cgui.button), "clicked",
GTK_SIGNAL_FUNC (left_button), NULL);
gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
cgui.button, 1, 2, 3, 4); gtk_widget_show (cgui.button);

/* put in buttons at bottom - right button */
cgui.button = gtk_button_new_with_label ("RIGHT");
gtk_signal_connect(GTK_OBJECT (cgui.button), "clicked",
GTK_SIGNAL_FUNC (right_button), NULL);
gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
cgui.button, 3, 4, 3, 4); gtk_widget_show (cgui.button);



/* create up-down label */
cgui.label[3] = gtk_label_new("UP_DOWN : 2.80");
gtk_label_set_justify(GTK_LABEL(cgui.label[3]),GTK_JUSTIFY_LEFT);
```

```
    gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
    cgui.label[3], 4, 6, 2, 3); gtk_widget_show (cgui.label[3]);

    /* create left-right label */
    cgui.label[4] = gtk_label_new("LEFT-RIGHT : 2.56");
    gtk_label_set_justify(GTK_LABEL(cgui.label[4]),GTK_JUSTIFY_LEFT);
    gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
    cgui.label[4], 4, 6, 4, 5); gtk_widget_show (cgui.label[4]);


    /* include seperator */
    cgui.separator = gtk_hseparator_new ();
    gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
    cgui.separator, 0, 6, 5, 6); gtk_widget_show(cgui.separator);

    /* put in buttons at bottom - start button */
    cgui.button = gtk_button_new_with_label ("Start");
    gtk_signal_connect(GTK_OBJECT (cgui.button), "clicked",
    GTK_SIGNAL_FUNC (start_function), NULL);
    gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
    cgui.button, 0, 1, 6, 7);  gtk_widget_show (cgui.button);

    /* create logging status label */
    cgui.label[5] = gtk_label_new("Logging OFF");
    gtk_label_set_justify(GTK_LABEL(cgui.label[5]),GTK_JUSTIFY_LEFT);
    gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
    cgui.label[5], 4, 6, 6, 7); gtk_widget_show (cgui.label[5]);

    /* put in buttons at bottom - stop button */
    cgui.button = gtk_button_new_with_label ("Stop");
    gtk_signal_connect(GTK_OBJECT (cgui.button), "clicked",
    GTK_SIGNAL_FUNC (stop_function), GTK_LABEL(cgui.label[5]));
    gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
    cgui.button, 1, 2, 6, 7); gtk_widget_show (cgui.button);

    /* put in buttons at bottom - log button */
    cgui.button = gtk_button_new_with_label ("Log");
    gtk_signal_connect(GTK_OBJECT(cgui.button), "clicked",
    GTK_SIGNAL_FUNC(data_log_function), GTK_LABEL(cgui.label[5]));
    gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
    cgui.button, 3, 4, 6, 7); gtk_widget_show (cgui.button);

    /* put in buttons at bottom - quit button */
    cgui.button = gtk_button_new_with_label ("Quit");
    gtk_signal_connect(GTK_OBJECT (cgui.button), "clicked",
    GTK_SIGNAL_FUNC (delete_event), NULL);
    gtk_table_attach_defaults (GTK_TABLE(cgui.table[0]),
    cgui.button, 2, 3, 6, 7); gtk_widget_show (cgui.button);

    /* show window and table*/
    gtk_widget_show(cgui.table[0]);
    gtk_widget_show(cgui.window[0]);

    gtk_main();

}


/* main gui function - run from main */
void gui_main(guiobj *gui)
{
    gui_lab(gui->window[0]);

    /* main gtk function */
    /*gtk_main();*/

}

int main( int   argc, char *argv[] )
{
    float a,b;
    /* initialisation */
    if (gui_init() == -1) {
       printf("Initialisation Problem - EXITING \n");
       return -1;
    }


    /* reset card */

    a=lookup_sat(5);
    b=lookup_sat(3);
         ioperm(IO_BASE,16,1);
    io_dacout(cloop,0,a/5*4095);
    io_dacout(cloop,1,b/5*4095);

    /* Create main gui */
    gui_main(&cgui);

    io_dacout(cloop,0,a/5*4095);
    io_dacout(cloop,1,b/5*4095);
    /* free shared memory */
    mbuff_free("control_loop", (void *)cloop);


    return 0;
}
```

/* gui end */

Appendix G: hovercraft.h

```
// type definition of the gui

typedef struct gtk_gui
{
   GtkWidget *window[3];
   GtkWidget *table[2];
   GtkWidget *label[6];
   GtkWidget *button;
   GtkWidget *radio[8];
   GSList *group[3];
   GtkObject *adjust[10];
   GtkWidget *separator;
   GtkWidget *spin[4];
   GtkWidget *textbox;
   GtkWidget *hbox;
   GtkWidget *vscrollbar;
} guiobj;
```

Appendix H: CDSM.c

```
/*
   File name: cdsm.c

   Data Sharing  interface

*/

#include "cdsm.h"
#ifdef __KERNEL__
#include <mbuff.h>
#include <linux/malloc.h>
#endif

inline void CDSM_init(void) {

   CDSM_data = (volatile long*)mbuff_alloc(CDSM_DATA_SI,
                 CDSM_NUMBER_OF_CHANNEL*sizeof(long));

}

inline void CDSM_done(void) {
   mbuff_free(CDSM_DATA_SI,(void*)CDSM_data);

}

inline void CDSM_set(int chan, long data)
{
   *(CDSM_data+chan) = data;
}

inline long CDSM_get(int chan)
{
   return *(CDSM_data+chan);
}
```

Appendix I: CDSM.h

```
/*
   CDSM - Common Data Sharing Mechanism

   written by: Linh Vu
   (C) 2002

*/

#ifndef __CDSM_H__
#define __CDSM_H__

// maximum of 16 channel, can change to any number
#define CDSM_NUMBER_OF_CHANNEL 128

// data in string id
#define CDSM_DATA_SI        "CDSM_DATA_SI"

// mechanism:
// common data sharing mechanism (CDSM)
//
// data will be shared among modules
// by read/write to an array of long type elements (also pointer)


volatile long *CDSM_data;

inline void CDSM_init(void);
inline void CDSM_done(void);
inline void CDSM_set(int chan, long data);
inline long CDSM_get(int chan);


#endif
```

Appendix J: mbuff.h

```
#ifndef __MBUFF_H
#define __MBUFF_H
#ifdef __cplusplus
extern "C" {
#endif


#define RTL_SHM_MISC_MINOR 254

/* max length of the name of the shared memory area */
#define MBUFF_NAME_LEN 32
/* max number of attached mmaps per one area */
#define MBUFF_MAX_MMAPS 16


#ifdef  SHM_DEMO
#define MBUFF_DEV_NAME "./mbuff"
#else
#define MBUFF_DEV_NAME "/dev/mbuff"
#endif

#ifdef __KERNEL__
#include <linux/types.h>
#include <linux/fs.h>
#else
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#endif


#define MBUFF_VERSION "0.7.2"

/*
   All ioctl()s are called with name filled in with the appropriate
   name for the mbuff to be referenced.  Calls to any ioctl() makes
   that mbuff "active", i.e., read(), write(), and mmap() use that
   mbuff.  I didn't do this yet.
   ioctl()s:

   ALLOCATE:
      Call with size=0 to just find out if the area exists; no
      mbuff will be allocated.  Otherwise, allocate an mbuff with
      that size.
   DEALLOCATE:
      Decrease reference count for an mbuff.

   issues:
      - using this method, it is *really* easy to get dangling
        mbuffs, i.e., mbuffs that nobody owns.  When you close
   /dev/mbuff, it would be a good idea to decrease the ref
   count of the active mbuff.
 */
```

```
#define IOCTL_MBUFF_INFO 0
#define IOCTL_MBUFF_ALLOCATE 1
#define IOCTL_MBUFF_DEALLOCATE 2
#define IOCTL_MBUFF_SELECT 3
#define IOCTL_MBUFF_LAST IOCTL_MBUFF_SELECT

struct mbuff_request_struct{
    unsigned int flags;

    char name[MBUFF_NAME_LEN+1];

    size_t size;

    unsigned int reserved[4];
};

#ifndef __KERNEL__

/* you can use mbuff_alloc several times, the buffer
will be deallocated when mbuff_free was called the same number of times
AND area is not mmaped anywhere anymore
AND it is not used in the kernel as well */
/* if you have a need to mmap the area at the specific address, use
 * mbuff_alloc_at */

inline void * mbuff_alloc_at(const char *name, int size, void * addr) {
    int fd;
    struct mbuff_request_struct req={0,"default",0,{0}};
    void * mbuf;

    if(name) strncpy(req.name,name,sizeof(req.name));
    req.name[sizeof(req.name)-1]='\0';
    req.size = size;
    if(( fd = open(MBUFF_DEV_NAME,O_RDWR) ) < 0 ){
        perror("open failed");
        return NULL;
    }
    size=ioctl(fd,IOCTL_MBUFF_ALLOCATE,&req);
    if(size<req.size)
        return NULL;
/* the type of first mmap's argument depends on libc version? This really
 * drives me crazy. Man mmap says "void * start" */
    mbuf=mmap(addr, size,PROT_WRITE|PROT_READ,MAP_SHARED|MAP_FILE,fd, 0);
    if( mbuf == (void *) -1)
        mbuf=NULL;
    close(fd);
    return mbuf;
}

inline void * mbuff_alloc(const char *name, int size) {
    return mbuff_alloc_at(name, size, NULL);
}

inline void mbuff_free(const char *name, void * mbuf) {
    int fd;
    struct mbuff_request_struct req={0,"default",0,{0}};
    int size;

    if(name) strncpy(req.name,name,sizeof(req.name));
    req.name[sizeof(req.name)-1]='\0';
    if(( fd = open(MBUFF_DEV_NAME,O_RDWR) ) < 0 ){
        perror("open failed");
        return;
    }
    size=ioctl(fd,IOCTL_MBUFF_DEALLOCATE,&req);
    if(size > 0) munmap( mbuf, size);
    close(fd);
    /* in general, it could return size, but typical "free" is void */
    return;
}

/* mbuff_attach and mbuff_detach do not change usage counters -
   area allocated using mbuff_attach will be deallocated on program
   exit/kill
   if nobody else uses it - mbuff_detach is not needed -
   the only lock keeping area allocated is mmap */

inline void * mbuff_attach_at(const char *name, int size, void * addr) {
    int fd;
    struct mbuff_request_struct req={0, "default", 0, {0}};
    void * mbuf;

    if(name) strncpy(req.name, name, sizeof(req.name));
    req.name[sizeof(req.name)-1]='\0';
    req.size = size;
    if(( fd = open(MBUFF_DEV_NAME, O_RDWR) ) < 0 ){
        perror("open failed");
        return NULL;
    }
    ioctl(fd,IOCTL_MBUFF_ALLOCATE,&req);
    mbuf=mmap(addr, size, PROT_WRITE|PROT_READ, MAP_SHARED|MAP_FILE, fd, 0);
    /* area will be deallocated on the last munmap, not now */
    ioctl(fd, IOCTL_MBUFF_DEALLOCATE, &req);
    if( mbuf == (void *) -1)
        mbuf=NULL;
    close(fd);
    return mbuf;
}

inline void * mbuff_attach(const char *name, int size) {
    return mbuff_attach_at(name, size, NULL);
}

inline void mbuff_detach(const char *name, void * mbuf) {
    int fd;
    struct mbuff_request_struct req={0,"default",0,{0}};
    int size;

    if(name) strncpy(req.name,name,sizeof(req.name));
    req.name[sizeof(req.name)-1]='\0';
    if(( fd = open(MBUFF_DEV_NAME,O_RDWR) ) < 0 ){
        perror("open failed");
        return;
    }
    size=ioctl(fd,IOCTL_MBUFF_SELECT,&req);
    if(size > 0) munmap( mbuf, size);
    close(fd);
    /* in general, it could return size, but typical "free" is void */
    return;
}
#else
```

```
struct mbuff{
   struct mbuff *next;
   struct mbuff *prev;

   char name[MBUFF_NAME_LEN+1];

   struct vm_area_struct *(vm_area[MBUFF_MAX_MMAPS]);
   struct file *file;

   unsigned char *buf;
   unsigned long size;
   int count;  /* number of allocations from user space */
   int kcount; /* number of allocations from kernel space */
   int open_cnt;/* #times opened */
   int open_mode;
};

extern struct mbuff * mbuff_list_lookup_name(const char *name,int
priority);
extern struct mbuff * mbuff_list_lookup_buf(void *buf);
extern int shm_allocate(const char *name,unsigned int size, void **shm);
extern int shm_deallocate(void * shm);

static inline void * mbuff_alloc(const char *name, int size) {
   void *tmp=NULL;
   if( shm_allocate(name, size, &tmp) > 0 )
      return tmp;
   else
      return NULL;
}
static inline void mbuff_free(const char *name, void * mbuf) {
/* it would be no problem to deallocate using only name */
   shm_deallocate(mbuf);
}
/* in kernel space implementing "nonlocking" attach and detach
   would be very unsafe (deallocation from user space possible at any time)
*/
#define mbuff_attach(name,size) mbuff_alloc(name,size)
#define mbuff_detach(name,mbuf) mbuff_free(name,mbuf)

extern char mbuff_default_name[];
extern int mbuff_ioctl(struct inode *inode, struct file *file, unsigned int
cmd,
   unsigned long arg);
#ifdef LINUX_V22
extern int mbuff_mmap(struct file *file, struct vm_area_struct *vma);
#else
extern int mbuff_mmap(struct inode *inode, struct file *file,
   struct vm_area_struct *vma);
#endif
extern int mbuff_open_with_name( struct inode *inode, struct file *file,
   const char * name);


#endif
#ifdef __cplusplus
}
#endif
#endif
```

Appendix K: makefile

```
all: hovercraft control_mod.o

include rtl.mk

control_mod.o: cloop.o thread_code.o rtmodule.o cdsm.o
    ld -r -o control_mod.o rtmodule.o cloop.o thread_code.o cdsm.o

cloop_nrt.o: cloop.c
    gcc -c -O2 -o cloop_nrt.o cloop.c


hovercraft: hovercraft.c cloop_nrt.o
    gcc -O2 -c -o hovercraft.o hovercraft.c `gtk-config --cflags`
    gcc -o hovercraft hovercraft.o cloop_nrt.o `gtk-config --libs`

clean:
   rm -f *.0 hovercraft
```

Appendix L: Linux kernel 2.2.18 installation

```
Linux kernel release 2.2.xx

These are the release notes for Linux version 2.2.  Read them carefully,
as they tell you what this is all about, explain how to install the
kernel, and what to do if something goes wrong.

However, please make sure you don't ask questions which are already
answered
in various files in the Documentation directory.  See DOCUMENTATION below.

WHAT IS LINUX?

  Linux is a Unix clone written from scratch by Linus Torvalds with
  assistance from a loosely-knit team of hackers across the Net.
  It aims towards POSIX compliance.

  It has all the features you would expect in a modern fully-fledged
  Unix, including true multitasking, virtual memory, shared libraries,
  demand loading, shared copy-on-write executables, proper memory
  management and TCP/IP networking.

  It is distributed under the GNU General Public License - see the
  accompanying COPYING file for more details.

ON WHAT HARDWARE DOES IT RUN?

  Linux was first developed for 386/486-based PCs.  These days it also
  runs on ARMs, DEC Alphas, SUN Sparcs, M68000 machines (like Atari and
  Amiga), MIPS and PowerPC, and others.

DOCUMENTATION:

 - There is a lot of documentation available both in electronic form on
   the Internet and in books, both Linux-specific and pertaining to
   general UNIX questions.  I'd recommend looking into the documentation
   subdirectories on any Linux FTP site for the LDP (Linux Documentation
   Project) books.  This README is not meant to be documentation on the
   system: there are much better sources available.

 - There are various README files in the Documentation/ subdirectory:
   these typically contain kernel-specific installation notes for some
   drivers for example. See ./Documentation/00-INDEX for a list of what
   is contained in each file.  Please read the Changes file, as it
   contains information about the problems, which may result by upgrading
   your kernel.

INSTALLING the kernel:

 - If you install the full sources, do a

       cd /usr/src
       gzip -cd linux-2.2.XX.tar.gz | tar xfv -

   to get it all put in place. Replace "XX" with the version number of the
   latest kernel.

 - You can also upgrade between 2.2.xx releases by patching.  Patches are
   distributed in the traditional gzip and the new bzip2 format.  To
   install by patching, get all the newer patch files and do

       cd /usr/src
       gzip -cd patchXX.gz | patch -p0

   or
       cd /usr/src
       bzip2 -dc patchXX.bz2 | patch -p0

   (repeat xx for all versions bigger than the version of your current
   source tree, _in_order_) and you should be ok.  You may want to remove
   the backup files (xxx~ or xxx.orig), and make sure that there are no
   failed patches (xxx# or xxx.rej). If there are, either you or me has
   made a mistake.

   Alternatively, the script patch-kernel can be used to automate this
   process.  It determines the current kernel version and applies any
   patches found.

       cd /usr/src
       linux/scripts/patch-kernel

   The default directory for the kernel source is /usr/src/linux, but
   can be specified as the first argument.  Patches are applied from
   the current directory, but an alternative directory can be specified
   as the second argument.

 - Make sure you have no stale .o files and dependencies lying around:

       cd /usr/src/linux
       make mrproper

   You should now have the sources correctly installed.

SOFTWARE REQUIREMENTS

   Compiling and running the 2.2.xx kernels requires up-to-date
   versions of various software packages.  Consult
   ./Documentation/Changes for the minimum version numbers required
   and how to get updates for these packages.  Beware that using
   excessively old versions of these packages can cause indirect
   errors that are very difficult to track down, so don't assume that
   you can just update packages when obvious problems arise during
   build or operation.

CONFIGURING the kernel:

 - Do a "make config" to configure the basic kernel.  "make config" needs
   bash to work: it will search for bash in $BASH, /bin/bash and /bin/sh
   (in that order), so one of those must be correct for it to work.

   Do not skip this step even if you are only upgrading one minor
   version.  New configuration options are added in each release, and
   odd problems will turn up if the configuration files are not set up
   as expected.  If you want to carry your existing configuration to a
   new version with minimal work, use "make oldconfig", which will
   only ask you for the answers to new questions.

 - Alternate configuration commands are:
    "make menuconfig"  Text based color menus, radiolists & dialogs.
    "make xconfig"     X windows based configuration tool.
   "make oldconfig"   Default all questions based on the contents of
            your existing ./.config file.

   NOTES on "make config":
   - having unnecessary drivers will make the kernel bigger, and can
     under some circumstances lead to problems: probing for a
     nonexistent controller card may confuse your other controllers
   - compiling the kernel with "Processor type" set higher than 386
     will result in a kernel that does NOT work on a 386.  The
     kernel will detect this on bootup, and give up.
   - A kernel with math-emulation compiled in will still use the
     coprocessor if one is present: the math emulation will just
     never get used in that case.  The kernel will be slightly larger,
     but will work on different machines regardless of whether they
     have a math coprocessor or not.
   - the "kernel hacking" configuration details usually result in a
     bigger or slower kernel (or both), and can even make the kernel
     less stable by configuring some routines to actively try to
     break bad code to find kernel problems (kmalloc()).  Thus you
     should probably answer 'n' to the questions for
          "development", "experimental", or "debugging" features.

 - Check the top Makefile for further site-dependent configuration
   (default SVGA mode etc).

 - Finally, do a "make dep" to set up all the dependencies correctly.

COMPILING the kernel:

 - Make sure you have gcc-2.7.2 or newer available.  It seems older gcc
   versions can have problems compiling newer versions of Linux.  This
   is mainly because the older compilers can only generate "a.out"-format
   executables.  As of Linux 2.1.0, the kernel must be compiled as an
   "ELF" binary.  If you upgrade your compiler, remember to get the new
   binutils package too (for as/ld/nm and company).

   Please note that you can still run a.out user programs with this
   kernel.

 - Do a "make zImage" to create a compressed kernel image.  If you want
   to make a boot disk (without root filesystem or LILO), insert a floppy
   in your A: drive, and do a "make zdisk".  It is also possible to do
   "make zlilo" if you have lilo installed to suit the kernel makefiles,
   but you may want to check your particular lilo setup first.

 - If your kernel is too large for "make zImage", use "make bzImage"
   instead.

 - If you configured any of the parts of the kernel as `modules', you
   will have to do "make modules" followed by "make modules_install".
   Read Documentation/modules.txt for more information.  For example,
   an explanation of how to use the modules is included there.

 - Keep a backup kernel handy in case something goes wrong.  This is
   especially true for the development releases, since each new release
   contains new code which has not been debugged.  Make sure you keep a
   backup of the modules corresponding to that kernel, as well.  If you
   are installing a new kernel with the same version number as your
   working kernel, make a backup of your modules directory before you
   do a "make modules_install".
 - In order to boot your new kernel, you'll need to copy the kernel
   image (found in /usr/src/linux/arch/i386/boot/zImage after compilation)
   to the place where your regular bootable kernel is found.

   For some, this is on a floppy disk, in which case you can "cp
   /usr/src/linux/arch/i386/boot/zImage /dev/fd0" to make a bootable
   floppy.  Please note that you can not boot a kernel by
   directly dumping it to a 720k double-density 3.5" floppy.  In this
   case, it is highly recommended that you install LILO on your
   double-density boot floppy or switch to high-density floppies.

   If you boot Linux from the hard drive, chances are you use LILO which
   uses the kernel image as specified in the file /etc/lilo.conf.  The
   kernel image file is usually /vmlinuz, or /zImage, or /etc/zImage.
   To use the new kernel, save a copy of the old image and copy the new
   image over the old one.  Then, you MUST RERUN LILO to update the
   loading map!! If you don't, you won't be able to boot the new kernel
   image.

   Reinstalling LILO is usually a matter of running /sbin/lilo.
   You may wish to edit /etc/lilo.conf to specify an entry for your
   old kernel image (say, /vmlinux.old) in case the new one does not
   work.  See the LILO docs for more information.

   After reinstalling LILO, you should be all set.  Shutdown the system,
   reboot, and enjoy!

   If you ever need to change the default root device, video mode,
   ramdisk size, etc.  in the kernel image, use the 'rdev' program (or
   alternatively the LILO boot options when appropriate).  No need to
   recompile the kernel to change these parameters.

 - Reboot with the new kernel and enjoy.

IF SOMETHING GOES WRONG:

 - If you have problems that seem to be due to kernel bugs, please check
   the file MAINTAINERS to see if there is a particular person associated
   with the part of the kernel that you are having trouble with. If there
   isn't anyone listed there, then the second best thing is to mail
   them to me (torvalds@transmeta.com), and possibly to any other
   relevant mailing-list or to the newsgroup.  The mailing-lists are
   useful especially for SCSI and networking problems, as I can't test
   either of those personally anyway.

 - In all bug-reports, *please* tell what kernel you are talking about,
   how to duplicate the problem, and what your setup is (use your common
   sense).  If the problem is new, tell me so, and if the problem is
   old, please try to tell me when you first noticed it.

 - If the bug results in a message like

    unable to handle kernel paging request at address C0000010
    Oops: 0002
    EIP:   0010:XXXXXXXX
    eax: xxxxxxxx   ebx: xxxxxxxx   ecx: xxxxxxxx   edx: xxxxxxxx
    esi: xxxxxxxx   edi: xxxxxxxx   ebp: xxxxxxxx
    ds: xxxx  es: xxxx  fs: xxxx  gs: xxxx
    Pid: xx, process nr: xx
    xx xx xx xx xx xx xx xx xx xx
```

```
  or similar kernel debugging information on your screen or in your
  system log, please duplicate it *exactly*.  The dump may look
  incomprehensible to you, but it does contain information that may
  help debugging the problem.  The text above the dump is also
  important: it tells something about why the kernel dumped code (in
  the above example it's due to a bad kernel pointer). More information
  on making sense of the dump is in Documentation/oops-tracing.txt

- You can use the "ksymoops" program to make sense of the dump.  Find
  the C++ sources under the scripts/ directory to avoid having to do
  the dump lookup by hand:

- In debugging dumps like the above, it helps enormously if you can
  look up what the EIP value means.  The hex value as such doesn't help
  me or anybody else very much: it will depend on your particular
  kernel setup.  What you should do is take the hex value from the EIP
  line (ignore the "0010:"), and look it up in the kernel namelist to
  see which kernel function contains the offending address.

  To find out the kernel function name, you'll need to find the system
  binary associated with the kernel that exhibited the symptom.  This is
  the file 'linux/vmlinux'.  To extract the namelist and match it against
  the EIP from the kernel crash, do:

     nm vmlinux | sort | less

  This will give you a list of kernel addresses sorted in ascending
  order, from which it is simple to find the function that contains the
  offending address.  Note that the address given by the kernel
  debugging messages will not necessarily match exactly with the
  function addresses (in fact, that is very unlikely), so you can't
  just 'grep' the list: the list will, however, give you the starting
  point of each kernel function, so by looking for the function that
  has a starting address lower than the one you are searching for but
  is followed by a function with a higher address you will find the one
  you want.  In fact, it may be a good idea to include a bit of
  "context" in your problem report, giving a few lines around the
  interesting one.

  If you for some reason cannot do the above (you have a pre-compiled
  kernel image or similar), telling me as much about your setup as
  possible will help.

- Alternately, you can use gdb on a running kernel. (read-only; i.e. you
  cannot change values or set break points.) To do this, first compile the
  kernel with -g; edit arch/i386/Makefile appropriately, then do a "make
  clean". You'll also need to enable CONFIG_PROC_FS (via "make config").

  After you've rebooted with the new kernel, do "gdb vmlinux /proc/kcore".
  You can now use all the usual gdb commands. The command to look up the
  point where your system crashed is "l *0xXXXXXXXX". (Replace the XXXes
  with the EIP value.)

  gdb'ing a non-running kernel currently fails because gdb (wrongly)
  disregards the starting offset for which the kernel is compiled.
```

Appendix M: RT-Linux 3.0 installation

```
        RTLINUX INSTALLATION INSTRUCTIONS
            FSM Labs, Inc.
           http://www.fsmlabs.com


DOWNLOADING THE APPROPRIATE LINUX KERNEL:
-----------------------------------------

In order to compile the RTLinux kernel, you first need to download the
kernel for which RTLinux was built.  To do so, note that there are two
patches in the top-level directories:

    kernel_patch-2.2, and
    kernel_patch-2.4

Where:

kernel_patch-2.2 is for 2.2.18 (x86 only):
 http://ftp.kernel.org/pub/linux/kernel/v2.2/linux-2.2.18.tar.gz

kernel_patch-2.4 is for 2.4.0-test1 (x86, PowerPC, Alpha):
 http://ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.0-test1.tar.gz

Choose which of the two Linux kernels you would like to run, and
download it from the appropriate website, as described above.


PREPARING FOR INSTALLATION:
---------------------------

Make sure you have gcc 2.7.2.3 or egcs-1.1.2 or egcs-2.91 installed.
You can verify that with

    gcc -v

On Debian, it is enough to install the "gcc272" package. On RedHat systems,
one needs to install the "kgcc" RPM.
You may be able to use other compiler versions, but this is not
recommended.


RTLINUX INSTALLATION:
---------------------

If you have downloaded the RTLinux distribution with a prepatched
kernel, skip steps 1 and 2. Quick check: if your kernel contains file
arch/i386/kernel/rtlinux.c, you do not need to patch the kernel.

1. put a fresh copy of the Linux kernel in the /usr/src/linux
directory:

    cd /usr/src
    tar xzf linux-2.2.18.tar.gz
    cd linux

1.b. If you haven't done so already, put a fresh copy of the RTLinux
kernel in the /usr/src/rtlinux directory:
   cd /usr/src
   tar xzf rtlinux.tar.gz
```

1.c. Create a symbolic link from within the rtlinux directory to the linux directory:

```
   cd /usr/src/rtlinux
   ln -sf /usr/src/linux ./linux
```

2. Patch the kernel with the RTLinux patch:

```
   cd /usr/src/linux
   patch -p1 < /usr/src/rtlinux/kernel_patch-2.2
```

OR, if you're using a 2.4.xx kernel:

```
   patch -p1 < /usr/src/rtlinux/kernel_patch-2.4
```

3. Now, configure the Linux kernel:

```
    cd /usr/src/linux
    make config
      or
    make menuconfig
      or
    make xconfig
```

Note: Enabling APM support is not recommended. APM BIOS calls may have unpredictable effect on real-time performance.

Note: On Alpha, you need to enable RTLinux Support (CONFIG_RTLINUX). On i386
and PPC, this is done automatically.

Note: Please make sure to specify the correct CPU type for the target machine.

4. After you are finished configuring the Linux kernel, type:

```
    make dep
```

Note. Steps 5 through 7 are x86-specific.

5. Compile the Linux kernel and modules:

```
   make bzImage
   make modules
```

5.b Install the Linux modules:

```
   make modules_install
   cp arch/i386/boot/bzImage /boot/rtzImage
```

6. Configure LILO. To do so, edit /etc/lilo.conf to contain the following piece (you only need to do this once):

```
image=/boot/rtzImage
   label=rtlinux
   read-only
   root=/dev/hda1
```

```
WARNING: replace /dev/hda1 in the above with your root filesystem. The
  easiest way to find out which filesystem it should be, take a look
  at the existing entry in your /etc/lilo.conf for "root=".
  Alternatively, type "df", and look for the line for "/" in the
  "mounted on" column. The corresponding entry in the "Filesystem"
  column is your root filesystem.

7. Install LILO. To do so, type:
   /sbin/lilo

7.b. Restart the computer:

   /sbin/shutdown -r now

7.c Load the RTLinux kernel: At the LILO: prompt, press "Shift" or
  "Tab".  This will give you a listing of the available kernels.
  Enter:

   rtlinux

RTLinux should boot.

8. Configure RTLinux:
   cd /usr/src/rtlinux
   make config OR  make menuconfig  OR make xconfig

9. Compile RTLinux:
   make
   make devices
   make install

The last step will create the directory:

     /usr/rtlinux-xx (xx denotes the version)

which contains the default installation directory for RTLinux which is
needed to create and compile user programs (that is, it contains the
include files, utilities, and documentation). It will also create a
symbolic link:

     /usr/rtlinux

which points to /usr/rtlinux-xx.  In order to maintain future
compatibility, please make sure that all of your own RTLinux programs
use /usr/rtlinux as its default path.

POST INSTALLATION AND RUNNING RTLINUX PROGRAMS:
-----------------------------------------------

To be able to run any programs, you must first load the rtlinux
modules. To do so, type:

   /usr/rtlinux/bin/rtlinux start

Or

   /usr/rtlinux/bin/rtlinux start <programname>

where <programname> is the name of the rtlinux program/module you want
to run.
```

You can also try running the examples. To do so, simply go to the appropriate directory under /usr/rtlinux/examples and type:

```
make test
```

For example:

```
cd /usr/rtlinux/examples/sound
make test
```

SPECIAL NOTES:

If you change any Linux kernel options, please don't forget to do:

```
cd /usr/src/rtlinux
make clean
make
make install
```

DOCUMENTATION AND SOURCES OF HELP:

The docs/html/GettingStarted document contains a brief introduction to RTLinux. Additional documents in docs/html also provide information about other aspects to RTLinux such as web installation, CD installation, FAQ, and RTiC-Lab.

In case of problems, please consult the FAQ first, available in the docs/ directory.

If all of the above fails, you can obtain help -- free of charge -- from your peers via the rtl@rtlinux.org mailing list for which you can un/subscribe to via http://www.rtlinux.org/mailing_lists.html.

FSM Labs further provides commercial support, development, and training. Please contact FSM Labs at

    business@fsmlabs.com for

additional information or visit their website at

    http://www.fsmlabs.com.

Appendix N: model.m

```
clear all
close all

%Starting point
x0=0;
y0=0;
fi0=0;
u0=0;
v0=0;
r0=0;
z10=cos(fi0)*x0+sin(fi0)*y0;
z20=-sin(fi0)*x0+cos(fi0)*y0;
z30=fi0;

[t,sol]=ode45('modelsim7ehv',[0,60],[z10,z20,z30,u0,v0,r0]);
z1=sol(:,1);z2=sol(:,2);z3=sol(:,3);u=sol(:,4);v=sol(:,5);w=sol(:,6);

x=[];
y=[];
fi=[];
for hihi=1:length(sol(:,1))
    trans=inv([cos(z3(hihi,1)) sin(z3(hihi,1)) ;-sin(z3(hihi,1))
                cos(z3(hihi,1)) ]);
   x=[x trans(1,:)*[z1(hihi,1) z2(hihi,1)]'];
   y=[y trans(2,:)*[z1(hihi,1) z2(hihi,1)]'];
   fi=[fi z3(hihi,1)];
end

figure(1),plot(t,u),title('surge');
xlabel('t [ s ]'),ylabel('velocity [ m/s ]')
hold on
plot([0 40],[2.25 2.25],'k',[40 40],[2.25 1.125],'k'
     ,[40 50],[1.125 1.125],'k',[50 50],[1.125 0],'k',[50 60],[0 0],'k')
plot([0 5],[0 0],'r',[5 5],[0 1.5],'r',[5 60],[1.5 1.5],'r')
figure(2),plot(t,v),title('sway');
xlabel('t [ s ]'),ylabel('velocity [ m/s ]')
figure(3),plot(t,w),title('yaw');
xlabel('t [ s ]'),ylabel('angular velocity [ rad/s ]')

figure(4)
okj=find(t<5*5&t>0);
plot([x(okj) x(okj(length(okj))+1)],[y(okj) y(okj(length(okj))+1)],'b')
hold on
kleurtjes=['r' 'r' 'r' 'r' 'r' 'r' 'r' 'k' 'k' 'k' 'k']
for ay=1:11
    okj=find(t<(ay+1)*5&t>ay*5);
    plot([x(okj) x(okj(length(okj))+1)],
         [y(okj) y(okj(length(okj))+1)],kleurtjes(ay))
    hold on
end
title('Movement of the hovercraft in earth coordinates');
hold on
x2=x(find(t>=45));
y2=y(find(t>=45));
[i,j]=size(sol);
lll=0.3; %length of the ship
n=[30 34 82 99 105 109 112 114 116 119
   124 131 138 150 190 227 245 259 578 4601];
```

```
colortjes=['b' 'b' 'r' 'r' 'r' 'r' 'r' 'r'
           'r' 'r' 'r' 'r' 'r' 'r' 'r' 'k' 'k' 'k' 'k' 'k'];
for nij=1:1:length(n)
   shipplot(x(1,n(nij)),y(1,n(nij)),fi(1,n(nij)),lll,colortjes(nij));
end

xlabel('X [ m ]'),ylabel('Y [ m ]')

figure(5),subplot(2,42,1:16);
plot(t,u),hold on,title('surge');
xlabel('t [ s ]'),ylabel('velocity [ m/s ]'),
plot([0 40],[2.25 2.25],'k',[40 40],[2.25 1.125],'k',[40 50],
[1.125 1.125],'k',[50 50],[1.125 0],'k',[50 60],[0 0],'k')
plot([0 5],[0 0],'r',[5 5],[0 1.5],'r',[5 60],[1.5 1.5],'r')

subplot(2,42,18),title('Fl'),subplot(2,42,20),title('Rudder');

subplot(2,42,23:38)plot(t,v),hold on,
title('sway');xlabel('t [ s ]'),ylabel('velocity [ m/s ]'),
plot([0 40],[2.25 2.25],'k',[40 40],[2.25 1.125],'k',[40 50],
[1.125 1.125],'k',[50 50],[1.125 0],'k',[50 60],[0 0],'k')
plot([0 5],[0 0],'r',[5 5],[0 1.5],'r',[5 60],[1.5 1.5],'r')

subplot(2,42,40),title('Fl'),subplot(2,42,42),title('Rudder');

subplot(2,42,43:58),plot(t,w),hold on,title('yaw'),
plot([0 40],[2.25 2.25],'k',[40 40],[2.25 1.125],'k',[40 50],
[1.125 1.125],'k',[50 50],[1.125 0],'k',[50 60],[0 0],'k')
plot([0 5],[0 0],'r',[5 5],[0 1.5],'r',[5 60],[1.5 1.5],'r')
xlabel('t [ s ]'),ylabel('angular velocity [ rad/s ]'),

subplot(2,42,60),title('Fl'),subplot(2,42,62),title('Rudder');

subplot(2,42,65:84)
okj=find(t<5*5&t>0);
plot([x(okj) x(okj(length(okj))+1)],[y(okj) y(okj(length(okj))+1)],'b')
hold on
for ay=1:11
    okj=find(t<(ay+1)*5&t>ay*5);
    plot([x(okj) x(okj(length(okj))+1)],
         [y(okj) y(okj(length(okj))+1)],kleurtjes(ay))
    hold on
end
    title('Movement of the hovercraft in earth coordinates');
hold on
[i,j]=size(sol);

xlabel('X [ m ]'),ylabel('Y [ m ]')

set(4,'Position',[0 50 530-30 530-50])
set(5,'Position',[500 50 530-30 530-50])
```

Appendix O: modelsim.m

```
function [xdot]=modelsim4(t,x)

% Defining parameters
m=2.1;
d11=0.6;
d22=0.8;
d33=.1;
a=0.4;
g=10;
mu1=0.1;
mu2=0.01;
mu3=0.004;
Jz=0.0948;

T1=5;
Fx=[1.8 1.8 1.8 1.8 1.8 1.8 1.8 1.8 1.8 1.8 1.8];% back thrust force
ang=[0 -10 -10 -10 -10  -10 -10 -10 -10 -10 -10];% rudder angle
Fl=[21 21 21 21 21 21 21 21 10 10 0];% lift force
%Simulation intervals
if t<=T1
   Fxx=Fx(1);          %in Newton
   d=ang(1)/360*2*pi;  % in rad
   Fll=Fl(1) ;
elseif t>T1&t<=2*T1
   Fxx=Fx(2);
   d=ang(2)/360*2*pi;
   Fll=Fl(2) ;
elseif t>2*T1&t<=3*T1
   Fxx=Fx(3);
   d=ang(3)/360*2*pi;
   Fll=Fl(3) ;
elseif t>3*T1&t<=4*T1
   Fxx=Fx(4);
   d=ang(4)/360*2*pi;
   Fll=Fl(4) ;
elseif t>4*T1&t<=5*T1
   Fxx=Fx(5);
   d=ang(5)/360*2*pi;
   Fll=Fl(5) ;
elseif t>5*T1&t<=6*T1
   Fxx=Fx(6);
   d=ang(6)/360*2*pi;
   Fll=Fl(6) ;
elseif t>6*T1&t<=7*T1
   Fxx=Fx(7);
   d=ang(7)/360*2*pi;
   Fll=Fl(7) ;
elseif t>7*T1&t<=8*T1
   Fxx=Fx(8);
   d=ang(8)/360*2*pi;
   Fll=Fl(8) ;
elseif t>8*T1&t<=9*T1
   Fxx=Fx(9);
   d=ang(9)/360*2*pi;
   Fll=Fl(9) ;
```

```
elseif t>9*T1&t<=10*T1
   Fxx=Fx(10);
   d=ang(10)/360*2*pi;
   Fll=Fl(10) ;
else
   Fxx=Fx(11);
   d=ang(11)/360*2*pi;
   Fll=Fl(11) ;
end

xdot=[x(4)+x(2)*x(6)
      x(5)-x(1)*x(6)
      x(6)
      x(5)*x(6)-d11/m*x(4)+1/m*2/4*Fxx*(1+cos(d))
          -mu1/m*(m*g-Fll)*atan(5000*x(4))*2/pi
      -x(4)*x(6)-d22/m*x(5)+1/m*2/4*Fxx*sin(d)
          -mu2/m*(m*g-Fll)*atan(5000*x(5))*2/pi
      -d33/Jz*x(6)+1/Jz*2/4*a*Fxx*sin(d)
          -mu3/Jz*(m*g-Fll)*atan(5000*x(6))*2/pi];
```

Appendix P: shipplot.m

```
function y=shipplot(x,y,fi,length,kleur)
% y=shipplot(x,y,fi,length,kleur)
% give the coordinates, angle, length of the ship and color of the ship
lll=length; %length of the ship
llll=lll;
Ax=x-cos(fi)*lll;Ay=y-sin(fi)*lll;
Ax2=x-cos(fi)*1.4*lll;Ay2=y-sin(fi)*1.4*lll;
Ax3=x-cos(fi)*0.8*lll;Ay3=y-sin(fi)*0.8*lll;
Bx=x+cos(fi)*lll;By=y+sin(fi)*lll;
Dx=Ax-sin(fi)*0.5*llll;
Dy=Ay+cos(fi)*0.5*llll;
Ex=Ax+sin(fi)*0.5*llll;
Ey=Ay-cos(fi)*0.5*llll;
Cx=Bx-sin(fi)*0.5*llll-0.5*llll*tan(40/360*2*pi)*cos(fi);
Cy=By+cos(fi)*0.5*llll-0.5*llll*tan(40/360*2*pi)*sin(fi);
Fx=Bx+sin(fi)*0.5*llll-0.5*llll*tan(40/360*2*pi)*cos(fi);
Fy=By-cos(fi)*0.5*llll-0.5*llll*tan(40/360*2*pi)*sin(fi);
plot([Ax3 Ax2],[Ay3 Ay2],kleur,[Dx Ex],[Dy Ey],kleur,
     [Ex Fx],[Ey Fy],kleur,[Fx Bx],[Fy By],kleur,
     [Bx Cx],[By Cy],kleur,[Cx Dx],[Cy Dy],kleur)
```

Appendix Q: Armature resistance test

Armature Resistance Test on Maxon Motor			
	Voltage (V)	Current (A)	Resistance (Ω)
	0,333	0,1833	1,814
	0,368	0,2216	1,661
	0,403	0,2375	1,697
	0,477	0,2444	1,952
	0,474	0,2506	1,891
	1,327	0,6790	1,954
	1,470	0,8200	1,793
	1,466	0,8694	1,686
	1,310	0,7580	1,728
	1,510	0,9350	1,615

Average Measured armature Resistance (Ω)	1,779

Appendix R: Fitting files

Determine J.m

```
clear all
load data
t             % time vector
om            % measured angular velocity

a=23.9;          % step input amplitude
kt=0.1065;       % torque constant
kv=kt;           % back emf constant
b=0.000114;      % damping constant
la=0.62/1000;    % inductance
ra=1.779;        % resistance

options=optimset('TolFun',[1e-18],'Tolx',[1e-18]);
J=fminsearch('Fit_J',0.001,options,t2,om2)

yfit=[];
p1=(ra*J+la*b)/2/la/J-sqrt((ra*J+la*b)^2-4*la*J*(ra*b+kt*kv))/2/la/J;
p2=(ra*J+la*b)/2/la/J+sqrt((ra*J+la*b)^2-4*la*J*(ra*b+kt*kv))/2/la/J;
for i=1:length(t)
   t2=t(i);
   yfit=[yfit; kt/J/la*a/p1/p2*(1+p2/(p1-p2)*exp(-p1*t2)-p1/(p1-p2)*exp(-
p2*t2))];
end
hold on
plot(t,om,t,yfit,'r')
```

Fit_J.m

```
function sse=Fit_J(params,input,output)
t2=input;
J=params;
a=23.5;
kt=0.1065;
kv=kt;
b=0.000308;
la=0.62/1000;
ra=1.779;
p1=(ra*J+la*b)/2/la/J-sqrt((ra*J+la*b)^2-4*la*J*(ra*b+kt*kv))/2/la/J;
p2=(ra*J+la*b)/2/la/J+sqrt((ra*J+la*b)^2-4*la*J*(ra*b+kt*kv))/2/la/J;
fitted_curve=[];
for i=1:length(t2)
   t=t2(i);
   fitted_curve=[fitted_curve; kt/J/la*a/p1/p2*(1+p2/(p1-p2)*exp(-p1*t)
                 -p1/(p1-p2)*exp(-p2*t))];
end

error_v=fitted_curve-output;
sse=sum(error_v.^2);
```