Teleoperated Tractor:
- development of a graphical user interface

Peter Danielsson

Luleå University of Technology

MSc Programmes in Engineering
Electrical Engineering
Department of Computer Science and Electrical Engineering
Division of EISLAB
Teleoperated Tractor: Development of a Graphical User Interface

Peter Danielsson
June, 2008
Abstract

This thesis is about the development of a graphical user interface for the teleoperated tractor IceMaker I.

There are companies in Arjeplog, Sweden that provide winter test services for vehicle and component manufacturers from all over the world. One of the leading service providers in Arjeplog is IceMakers. They have during 30 years developed into a full service supplier to their clients. The services include arranging accommodation for the company personnel, providing components and equipment desired by the customer, and the most important task of all; making the ice tracks (often on the lakes), varying in shapes and sizes depending upon the type of testing being conducted.

There is a major risk factor involved in the maintenance of the tracks. Running a tractor on the lake ice is not safe since the ice might break. The tractor could end up at the bottom of the lake with the driver trapped inside.

IceMakers want a teleoperated tractor in order to decrease the risks for the personnel maintaining the tracks. Another benefit with a teleoperated tractor is that it provides the possibility to extend the winter test season. IceMakers have submitted the development of the control system to a project group at Luleå University of Technology.

To enable teleoperation, the tractor has been equipped with a computer, some sensors and other necessary equipment. The user controls the tractor with a joystick, and gets video and data feedback from the tractor.

The purpose of this thesis was to develop a user friendly graphical user interface (GUI) for the teleoperated tractor, and also to add support for two new units to the system; a DSP3000 gyro and a GPS receiver.

Modifications of the old controller program have been performed in order to make it compatible with the new devices and the GUI. The program has also been restructured in order to make it easier to expand it in the future. Another improvement of the software is increased safety with the addendum of a watchdog.

The GUI is written in Java. It can show video from two network cameras and different status information from the tractor. Besides that, functions for showing the tractors GPS position on a user defined local map have been developed. The software has performed well during testing.

Keywords: Graphical User Interface, GUI, Teleoperation, Gyro, Hydraulic Steering, GPS
Preface

The work presented in this thesis was performed during the spring of 2008, as the final part of the MSc programme in Electrical Engineering at Luleå University of Technology. The subject was to develop a graphical user interface for a teleoperated tractor that is aimed to remove snow and prepare test tracks on an ice-covered lake. The development of the tractor is a collaboration between Luleå University of Technology and IceMakers in Arjeplog.

I would like to thank Lars Sundström, MD at IceMakers and my examiner Prof. Kalevi Hyyppä at Luleå University of Technology for the opportunity to do this master thesis. I would also like to thank Mattias Jonsson, service manager at IceMakers for the support during the testing of the tractor in Arjeplog. Another person I would like to thank in particular is my supervisor Håkan Fredriksson, who is a Ph.D. student at the university, for his good ideas and support during the project. Finally, I would like to thank my colleagues Prashant Rana and Marion Billaux, who were involved in the project prior this thesis.

Peter Danielsson
Luleå, June 2008
Contents

ABSTRACT .....................................................................................................................................................I

PREFACE ..........................................................................................................................................................II

CONTENTS ......................................................................................................................................................III

1  INTRODUCTION ........................................................................................................................................... 1
  1.1  BACKGROUND ......................................................................................................................................... 1
  1.2  PURPOSE ............................................................................................................................................... 2
  1.3  LIMITATIONS ......................................................................................................................................... 3

2  THEORY ........................................................................................................................................................ 4
  2.1  GLOBAL POSITIONING SYSTEM ........................................................................................................... 4
      2.1.1  Principles of Satellite Navigation ................................................................................................ 4
      2.1.2  Geodetic Coordinates .................................................................................................................... 4
      2.1.3  Earth Centred Earth Fixed (ECEF) Coordinates ............................................................................ 5
      2.1.4  Local East, North, Up (ENU) Coordinates ................................................................................... 5
      2.1.5  Conversion From Geodetic Coordinates to Local ENU Coordinates ............................................. 5
  2.2  REAL-TIME PROGRAMMING ................................................................................................................ 6
      2.2.1  What is a thread? ............................................................................................................................. 6
      2.2.2  Concurrency and Parallelism ........................................................................................................ 6
      2.2.3  Synchronization Issues ................................................................................................................ 7
      2.2.4  Critical Sections .............................................................................................................................. 7
      2.2.5  Synchronization Variables ............................................................................................................. 7
      2.2.6  Mutexes ......................................................................................................................................... 7
  2.3  CONTROLLING THE TRACTOR ................................................................................................................. 7
      2.3.1  The Speed Control ......................................................................................................................... 7
      2.3.2  The A2 HST Controller ................................................................................................................. 8
      2.3.3  The Speed Control Devices ......................................................................................................... 8
      2.3.4  How to Control the Speed via Remote Interface ............................................................................ 8
      2.3.5  The Steering Control Unit (SCU) ................................................................................................. 8
      2.3.6  How to Control the Steering via Remote Interface ...................................................................... 8
      2.3.7  The Steering Valve PVG 32 .......................................................................................................... 9
      2.3.8  Control of the Lights and Wipers ................................................................................................. 9
  2.4  STEERING REGULATOR ......................................................................................................................... 10
      2.4.1  Motion Model of the Vehicle ....................................................................................................... 10
      2.4.2  Steering System Regulator ......................................................................................................... 10
      2.4.3  Hydraulic Steering System Properties ......................................................................................... 11

3  METHOD ....................................................................................................................................................... 12
  3.1  WORK APPROACH ............................................................................................................................... 12
      3.1.1  Controller Modifications ................................................................................................................ 12
      3.1.2  Development of the GUI ............................................................................................................ 12
      3.1.3  Testing of the Software ............................................................................................................... 12
  3.2  EQUIPMENT ............................................................................................................................................ 13
      3.2.1  PC104 ........................................................................................................................................... 13
      3.2.2  Router .......................................................................................................................................... 13
      3.2.3  ADAM -Modules .......................................................................................................................... 13
      3.2.4  DSP-3000 Gyro ............................................................................................................................ 14
      3.2.5  GPS Receiver ................................................................................................................................ 14
      3.2.6  Speed Decoder .............................................................................................................................. 14
      3.2.7  Network Cameras ........................................................................................................................ 14
      3.2.8  Safety Switch ................................................................................................................................. 14
      3.2.9  Client Computer ........................................................................................................................... 15

4  RESULT ......................................................................................................................................................... 16
  4.1  RELAY LOGIC ........................................................................................................................................... 16
  4.2  CONTROLLER PROGRAM ..................................................................................................................... 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1</td>
<td>pc104_main</td>
<td>17</td>
</tr>
<tr>
<td>4.2.2</td>
<td>port_management</td>
<td>17</td>
</tr>
<tr>
<td>4.2.3</td>
<td>current_status</td>
<td>18</td>
</tr>
<tr>
<td>4.2.4</td>
<td>watchdog</td>
<td>18</td>
</tr>
<tr>
<td>4.2.5</td>
<td>joystick</td>
<td>19</td>
</tr>
<tr>
<td>4.2.6</td>
<td>adam_modules</td>
<td>20</td>
</tr>
<tr>
<td>4.2.7</td>
<td>steering_regulator</td>
<td>20</td>
</tr>
<tr>
<td>4.2.8</td>
<td>speed_decoder</td>
<td>21</td>
</tr>
<tr>
<td>4.2.9</td>
<td>DSP3000</td>
<td>21</td>
</tr>
<tr>
<td>4.2.10</td>
<td>gps_receiver</td>
<td>21</td>
</tr>
<tr>
<td>4.3</td>
<td>GRAPHICAL USER INTERFACE</td>
<td>22</td>
</tr>
<tr>
<td>4.3.1</td>
<td>GUIFrame</td>
<td>22</td>
</tr>
<tr>
<td>4.3.2</td>
<td>OptionFrame</td>
<td>23</td>
</tr>
<tr>
<td>4.3.3</td>
<td>AboutPopup</td>
<td>23</td>
</tr>
<tr>
<td>4.3.4</td>
<td>GPSdata</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>GUI USER GUIDE</td>
<td>25</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Installation/File Structure</td>
<td>25</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Configuration File</td>
<td>25</td>
</tr>
<tr>
<td>4.4.3</td>
<td>System Requirements</td>
<td>26</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Component Description of the Main Window</td>
<td>26</td>
</tr>
<tr>
<td>4.4.5</td>
<td>GPS Map</td>
<td>29</td>
</tr>
<tr>
<td>4.4.6</td>
<td>Component Description of the GPS Map Window</td>
<td>29</td>
</tr>
<tr>
<td>4.4.7</td>
<td>Option Window</td>
<td>30</td>
</tr>
<tr>
<td>4.4.8</td>
<td>How to Connect to the Controller Program</td>
<td>31</td>
</tr>
<tr>
<td>4.4.9</td>
<td>How to Disconnect from the Controller Program</td>
<td>31</td>
</tr>
<tr>
<td>4.4.10</td>
<td>Joystick</td>
<td>31</td>
</tr>
<tr>
<td>4.5</td>
<td>DISCUSSION</td>
<td>32</td>
</tr>
<tr>
<td>5.1</td>
<td>FUTURE DEVELOPMENT</td>
<td>32</td>
</tr>
<tr>
<td>REFERENCES</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>ATTACHMENT 1: INTERCONNECTION BETWEEN ADAM MODULES</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>ATTACHMENT 2: SAFETY SWITCH &amp; SPEED DECODER COMMANDS</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>ATTACHMENT 3: RELAY CIRCUIT SCHEMATICS</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>ATTACHMENT 4: CURRENT STATUS MESSAGES</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>ATTACHMENT 5: PROGRAM LOG &amp; CONTROL COMMAND SYNTAX</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>ATTACHMENT 6: FORMAT OF COORDINATES IN MAP FILES</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background

Arjeplog in the north of Sweden is with respect to the area the fourth largest municipality in the country (12 945 km²) [1]. Despite this, it only has a population of 3 089 (31/12 2007) peoples. Arjeplog municipality consists of a terrain dominated by the Scandinavian mountain range and many water areas. It has astonishing 8 727 lakes and streams.

In the severe winter season these lakes and streams freeze, turning many of the water bodies surrounding Arjeplog to “ice roads” [2]. These “ice roads” are utilized for vehicle testing by car manufacturers from all over the world. Vehicle testing in Arjeplog has lead to many important innovations which are being used by car manufacturers worldwide. For example, the anti-lock braking system (ABS) technology.

There are companies in Arjeplog that provide services for vehicle and component manufacturers. One of the leading service providers in Arjeplog is IceMakers. They have during 30 years developed into a full service supplier to their clients. The services include arranging accommodation for the company personnel, providing components and equipment desired by the customer, and the most important task of all; making the ice tracks, varying in shapes and sizes depending upon the type of testing being conducted.

Once winter hits Arjeplog, the surrounding water bodies freeze. There are often a lot of snow present on these lakes and streams. In the beginning of the winter season, snowmobiles are driven over the areas where the test tracks are to be made. This is to compress the snow and thus decrease the isolation that the snow gives. When the ice is thick enough, a light weighted tractor is sent out to remove this snow giving way to flat ice road. As the ice grows thicker, bigger machines are used to maintain the tracks.

There is a major risk factor involved in the maintenance of the tracks. Running a tractor on the lake ice is not safe since the ice might break. The tractor could end up at the bottom of the lake with the driver trapped inside.

IceMakers want a teleoperated tractor in order to decrease the risks for the personnel maintaining the tracks. Another benefit with a teleoperated tractor is that it provides the possibility to extend the winter test season. IceMakers have submitted the development of the control system to a project group at Luleå University of Technology.

The development of the teleoperated tractor began in the spring of 2007 in a course called “Project in Mechatronics”. The project group consisting of; Marion Billaux, Peter Danielsson and Prashant Rana, laid the foundation for the control system of the tractor. The tractor IceMaker I seen in Figure 1 can be teleoperated from a computer at a remote location through an ordinary wireless network, WLAN. The tractor has been equipped with a computer, some sensors and other necessary equipment. The user controls the tractor with a joystick, and gets video and data feedback from the tractor.
1.2 Purpose

This thesis is a continuation of the work performed by the previous project group, mentioned in section 1.1.

The aim of this project was to:

- Mount a fiber optic gyro, and implement this in the tractors software to provide the possibility to increase the course stability when teleoperating.

- Mount GPS equipment, and implement this in the tractors software for presentation of the tractors position in a graphical interface.

- Examine the possibility to provide support for a number of new signals in the system, for example, water temperature and oil pressure. The possibility to do this within this thesis had to be considered.

- Develop a user friendly graphical user interface for teleoperation and surveillance of the tractor. This interface had to clearly show at least the following information:
  - Two video streams from two different network cameras.
  - The vehicles speed.
  - Accelerator pedal status.
  - Steering angle.
  - Wiper and light status.
  - The vehicles position on a map.

Practical tests of the equipment and the software were also a part of the work. The tests were partly performed at IceMakers on one of their testing facilities in Arjeplog. IceMakers provided at these occasions a suitable workshop.
1.3 Limitations

A future goal of the development of the control system is to make the tractor IceMaker I more autonomous. For example, make it possible to let the tractor drive on its own along a path drawn with GPS coordinates. Development and implementation of autonomous algorithms will not be covered within this master thesis that is limited to 20 weeks.
2 Theory

Section 2.1 gives general information about the Global Positioning System. It also presents different coordinate systems that are used to specify locations on the Earth.

2.1 Global Positioning System

The Global Positioning System, GPS developed by the United States Department of Defense, utilize a Medium Earth Orbit constellation of at least 24 satellites [3]. The satellites transmit microwave signals, which can be used by a GPS receiver to determine its location, speed, direction and time. The GPS system became fully operational in 1995. The GPS system has many applications for both the military and civilian industry.

2.1.1 Principles of Satellite Navigation

To determine its position, a typical GPS receiver needs at least signals from four or more satellites [3]. The GPS continually transmits messages containing current time, parameters to calculate the location of the satellite (the ephemeris) and the general system health (the almanac). The speed of the signals is known, they travel at the speed of light through outer space, and slightly slower through the atmosphere. The receiver can therefore compute the distance to each satellite with the arrival time of the message, according to:

\[ x_{\text{distToSat}} = v_{\text{speedOfSignal}} \cdot \Delta t. \]  

(1)

When the distance to the satellites is determined, the receiver can calculate its position using geometry and trigonometry.

In some special cases fewer than four satellites are needed to determine the position of the receiver. If one variable is known a receiver can determine its position using only three satellites. Receivers may also use additional data (doppler shift of satellite signals, last known position, dead reckoning, etc.) to approximate the position when fewer than four satellites are visible.

2.1.2 Geodetic Coordinates

In geodetic coordinates the Earth’s surface is approximated by an ellipsoid and locations near the surface are described in terms of latitude (\( \phi \)), longitude (\( \lambda \)) and height (\( h \)) [4]. The ellipsoid is completely parameterized by the semi-major axis \( a \) and the flattening \( f \).

The global positioning system uses the world geodetic system 1984 (WGS84) to determine the location of a point near the surface of the Earth.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Notation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi-major axis</td>
<td>( a )</td>
<td>6378137.0 m</td>
</tr>
<tr>
<td>Reciprocal of flattening</td>
<td>( 1/f )</td>
<td>298.257223563</td>
</tr>
</tbody>
</table>

Table 1: WGS84 defining parameters.
2.1.3 Earth Centred Earth Fixed (ECEF) Coordinates

The Earth-centred Earth-fixed (ECEF) coordinate system has its origin at the centre of the Earth [4]. The X axis passes through the equator at the prime meridian. The Z axis passes through the North Pole. The Y axis can be determined by the right hand rule to be passing through the equator at 90° longitude.

2.1.4 Local East, North, Up (ENU) Coordinates

![Figure 2: ENU coordinates.](image)

In many applications the local East, North, Up (ENU) Cartesian coordinate system is far more intuitive and practical than ECEF or geodetic coordinates [4]. The local ENU coordinates are formed from a plane tangent to the Earth’s surface fixed to a specific location, and is sometimes known as a “local tangent” or “local geodetic” plane. By convention the east axis is labeled x, the north y and the up z, see Figure 2.

2.1.5 Conversion from Geodetic Coordinates to Local ENU Coordinates

The conversion from geodetic coordinates to local ENU coordinates is a two-step process:

1. Convert geodetic coordinates to ECEF coordinates.
2. Convert ECEF coordinates to local ENU coordinates.

From Geodetic to ECEF Coordinates

Geodetic coordinates (latitude \( \phi \), longitude \( \lambda \), height \( h \)) can be converted into ECEF coordinates using the following formulas [4]:

\[
X = \left( \frac{a}{\chi} + h \right) \cos \varphi \cos \lambda
\] (2)
\[ Y = \frac{a + h}{\chi} \cos \phi \sin \lambda \]  

(3)

\[ Z = \frac{a(1-e^2)}{\chi} + h \sin \phi \]  

(4)

Where \( \chi = \sqrt{1-e^2 \sin^2 \phi} \), \( a \) and \( e^2 \) are the semi-major axis and the square of the first numerical eccentricity of the ellipsoid respectively.

**From ECEF to ENU Coordinates**

To transform from ECEF coordinates to local coordinates we need a local reference point [4]. If the reference point is located at \{X_r, Y_r, Z_r\} and an object at \{X_p, Y_p, Z_p\}, then the vector pointing from the reference point to the object in the ENU frame is:

\[
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
= 
\begin{bmatrix}
-\sin \lambda & \cos \lambda & 0 \\
-\sin \varphi \cos \lambda & -\sin \varphi \sin \lambda & \cos \varphi \\
\cos \varphi \cos \lambda & \cos \varphi \sin \lambda & \sin \varphi
\end{bmatrix}
\begin{bmatrix}
X_p - X_r \\
Y_p - Y_r \\
Z_p - Z_r
\end{bmatrix}.
\]  

(5)

**2.2 Real-Time Programming**

A system that needs to react to events (inputs) within certain deadlines is said to be a real-time system [10]. Hard real-time systems are used when it is critical that an event is reacted to within a strict deadline. An example of a hard real-time system is the airbag. If the airbag is released too late after a car crash, then it would be useless. A soft real-time system has less deadline requirements. A washing machine is an example of a soft real-time system. If the washing machine starts one second later than expected when the user has pushed the start button, it would only delay the finishing time and the system would still work. Section 2.2 describes fundamental terms when it comes to real-time programming.

**2.2.1 What is a thread?**

A multitasking operating system can do several things concurrently by running more than a single process [5]. A process can also do several things concurrently by running more than one thread. Each thread can execute its instructions independently, this makes it possible for a multithreaded process to perform numerous tasks concurrently. For example, one thread can run the GUI, a second can handle the input from a user, and a third can perform calculations.

All threads in a process have a shared state. They are in the exact same memory space, and thus see the same functions and data. If one thread change a process variable, then all other threads will see the change if they later access it.

**2.2.2 Concurrency and Parallelism**

The definition of concurrency is that two or more threads can be in the middle of executing code at the same time [5]. Only one thread can be on the CPU at any given time. The illusion of that all the threads are running simultaneously is made possible by switching between the threads with a high frequency.
The definition of parallelism is that two or more threads in fact run simultaneously on different CPUs. Many threads can run in parallel, and also concurrently, on a multiprocessor machine.

2.2.3 Synchronization Issues

Reliably synchronization of the different threads when writing any kind of concurrent program is vital to avoid unwanted bugs [5]. Without synchronization, it’s possible for two threads to change some data at the same time, one will overwrite the other.

2.2.4 Critical Sections

A section of code that must complete with no interruption that affects its completion is called a critical section [5]. A critical section is handled by locking a lock, manipulating the data and then releasing the lock. If one thread has locked a shared data structure, and there is another thread that wants to execute the same critical section, it will have to wait until the first thread finishes. Critical sections should be made as short and optimized as possible since they may significantly affect the concurrency of the program.

2.2.5 Synchronization Variables

Synchronization is provided by a set of functions that manipulate special structures in user memory [5]. POSIX implements three synchronization variables, (locks, semaphores and condition variables), and the function pthread_join() to provide this functionality.

Locks are used to protect shared data. Semaphores, conditions variables, join etc. are used to prevent threads from running when there is nothing for them to do.

2.2.6 Mutexes

The simplest synchronization variable is the mutual exclusion lock [5]. Example 1 shows the POSIX syntax for this kind of lock.

Example 1

```c
pthread_mutex_lock(&mutex_ex);
   Critical section
   ................
   ............
}
pthread_mutex_unlock(&mutex_ex);
```

The first thread that locks the mutex gets ownership, any other threads that try to lock the mutex will fail, causing the calling thread to go to sleep. When the owner unlocks the mutex, one of the sleepers will be awakened, and given the chance to get ownership.

2.3 Controlling the Tractor

Section 2.3 gives a brief description of the steering and speed control systems on the tractor and how we control these systems remotely. The control of the lights and wipers is also explained in this section.

2.3.1 The Speed Control

The speed and direction (forward or reverse) changes in the tractor are controlled by a hydrostatic transmission (HST) [2]. All the power is transferred from the engine to the wheels
with hydraulics, through foot pedal operation. The system consists of a pump and a hydrostatic motor.

2.3.2 The A2 HST Controller

The A2 HST controller is a pre-programmed electrical device [2]. It receives signals from different devices in the tractor, and processes the information. The controller provides output commands to the proportional solenoids in the transmission and to the display panel to inform the driver.

2.3.3 The Speed Control Devices

The main way to control the speed of the tractor is to use the forward pedal or the reverse pedal [2]. Each pedal is connected to the HST controller, and sending a voltage that is proportional to the pedal position. When a pedal is not pressed, the controller receives 0.8V. And when a pedal is at the maximum pressed position, the controller receives 4.2V. The fact that the pedal is sending 0.8V when it’s not pressed makes it possible for the controller to check if there is any connection problem. In this case it will display an error message and the tractor will not move.

There is also a gearlever on the tractor, and a throttle control for changing the rpm of the tractor's engine. The scale between the voltage sent to the controller and the resulting velocity of the tractor is highly dependent on the current gear and the engine rpm.

The distinctiveness of this system is that there is no real need to have a brake. As soon as the pedal is released, the controller receives the minimum signal and stops the tractor. There are however two braking pedals. They are mainly to be used as steering brakes in case of hard environment, like a wet field or deep snow.

2.3.4 How to Control the Speed via Remote Interface

Because of the controller checking for connection (0.8V), only one pedal at a time is disconnected from the controller [2]. Our voltage command (between 0.8V and 4.2V) is sent to the controller instead of the voltage signal from the disconnected pedal. The controller then processes the received voltage as if it was coming from the pedal.

2.3.5 The Steering Control Unit (SCU)

The steering wheel of the tractor is connected to the steering cylinder through the SCU [2]. The cylinder is itself directly connected to the front wheels, changing the angle of the wheels as it moves left or right.

The main part of the SCU is the open center control valve. If the steering wheel is turned to the right or to the left, the valve directs the hydraulic pressure to the corresponding cylinder hose.

2.3.6 How to Control the Steering via Remote Interface

An additional steering valve with a voltage controlled unit is connected to the tractor's steering cylinder [2]. It is possible to switch between the valve and the SCU. The cylinder position is controlled by the voltage sent to the steering valve.
2.3.7 The Steering Valve PVG 32

The PVG 32 valve seen in Figure 3 has been added to the tractor [2]. First, the valve is turned on and off with a 12V supply voltage. If the valve is off, then it’s disconnected from the steering cylinder and the SCU controls the tractor. When the valve is on, the steering wheel has no effect on the cylinder, which is controlled only by the valve.

To be able to control the steering, the valve can receive a proportional command voltage. This command unit is composed of three signals:
- the reference voltage $V_{\text{ref}}$: 11V to 30V
- the reference ground
- the signal voltage: - 0.25 * $V_{\text{ref}}$: maximum turning speed to the right
  - 0.50 * $V_{\text{ref}}$: no movement
  - 0.75 * $V_{\text{ref}}$: maximum turning speed to the left

**How to Get a Suitable Reference Voltage**

As the relation between the turning speed and the signal voltage depends on the value of $V_{\text{ref}}$, it’s important to know the exact value of $V_{\text{ref}}$ and to be sure that it is stable in the long run [2].

The tractor is running on a 12V battery, but this voltage source is not stable enough for $V_{\text{ref}}$. To get a suitable signal, a 12V to 15V converter makes sure that the signal is above 12V. A 12V voltage regulator is then used to get a really stable and clean signal.

2.3.8 Control of the Lights and Wipers

The lights and the wipers are controlled by mechanical switches on the tractor consoles [2]. By short-circuiting the switches with car relays, they can be controlled remotely. The coils of the relays are connected to computer controlled switches. Two controlled switches are used. One is controlling both the front and rear wipers, and the other is controlling the extra lights.
2.4 Steering Regulator

Section 2.4 describes the regulator that is used to control the steering of the tractor.

2.4.1 Motion Model of the Vehicle

Figure 4: Motion model used for controlling the tractor.

No direct position feedback is used for the steering angle of the front wheels [6]. Instead the steering angle is estimated by a motion model of the vehicle using gyro and speedometer information. The motion model used is the three wheeled bicycle model shown in Figure 4. The rotation speed $\omega$ of the vehicle is measured around the rear axle, and the vehicle velocity $v$ is measured at the rear wheels. The steering angle of the vehicle model can be calculated with:

$$\alpha_{\text{model}} = \arctan\left(\frac{\omega l}{v}\right).$$  \hspace{1cm} (6)

The calculated steering angle does only represent the true steering angle when there is no slip between the wheels and the surface. However, in our application this difference is disregarded as long as the tractor moves according to the motion model.

2.4.2 Steering System Regulator

The steering actuator consists of a hydraulic cylinder, manoeuvred by an electrical controllable proportional hydraulic valve [6]. The velocity of the hydraulic cylinder is very much dependent on the engine rpm, and also if other hydraulic equipment requires much hydraulic oil flow. Therefore, it is justified to use a regulator that is tolerant to the constantly changing working conditions. Changes in the hydraulic system can, when using a simple proportional regulator, be seen as changes in the amplification of the regulator. We utilize a proportional regulator that uses the difference between the reference angle $\alpha_{\text{ref}}$ and the angle calculated according to the motion model, $\alpha_{\text{model}}$:

$$y_a = k(\alpha_{\text{ref}} - \alpha_{\text{model}}).$$  \hspace{1cm} (7)

The parameter $k$ is the gain in the regulator.


2.4.3 Hydraulic Steering System Properties

![Hydraulic cylinder with the active retraction and extension areas marked.](image)

The proportional valve controls the flow of oil to the hydraulic cylinder [6]. The active piston area, and hence the volume in the cylinder, is larger during extension than retraction, see Figure 5. Therefore, the same amount of oil flow causes a lower cylinder speed during extension than in the retraction direction [7]. The ratio $R$ is used to compensate for the speed difference. It is calculated as:

$$R = \frac{\text{ExtendArea}}{\text{RetractArea}}, \quad (8)$$

where $\text{ExtendArea}$ and $\text{RetractArea}$ are the active cylinder areas shown in Figure 5.

Finally, there is a dead band in the steering actuator system. The absolute value of the control signal to the actuator $y_{\text{out}}$ has to be larger than a specific threshold value before the hydraulic cylinder starts to move. Therefore the parameter $y_{\text{th}}$ is added (or subtracted, depending on the sign of $y_a$) to the output signal, see Example 2.

**Example 2**

- if $y_a \geq 0$
  
  $$y_{\text{out}} = y_a + y_{\text{th}}$$

- else
  
  $$y_{\text{out}} = y_a / R - y_{\text{th}}$$
3 Method

3.1 Work Approach

3.1.1 Controller Modifications

The graphical user interface (GUI) needs status information from the onboard controller, for example, light, wiper, and speed status. The old version of the controller software had no support for sending such information. So the first step towards a fully functional GUI was to add support for sending status information from the controller software. It was also necessary to write drivers for the two new units in the system, a DSP3000 gyro and a GPS receiver. Besides that there was a need to change the structure of the software to make it more compatible for future expansions, this was done by dividing the program into different modules. Another improvement of the software is increased safety with the addendum of a watchdog.

3.1.2 Development of the GUI

I have chosen Java as the programming language for the GUI since it has good support for graphical programming, and is platform independent. I have used the programming tool NetBeans IDE during the development of the GUI. With NetBeans it’s possible to create graphical components by dragging and positioning them from a palette onto a canvas. This feature is very convenient since it lets the developer to focus on the GUI design and functionality rather than the syntax for a specific component. Besides that, NetBeans provides good support for creating UML diagrams and Java API documentation.

3.1.3 Testing of the Software

An important part of the work has been to test the software. The testing has been carried out through the entire project. As soon as a change has been done in the software, tests have followed to evaluate if everything worked as supposed to. I divide the potential errors into three groups:

1. Obvious errors, (for example, if a button doesn’t work).
2. Special case errors, (for example, if a button works most of the time but sometimes it doesn’t).
3. Minor errors, (for example, if a calculation of the speed is slightly wrong).

During the development of a specific software part, I first tested the new part by making debug print outs from it. When I considered the software part to be complete, it was necessary to test the entire system by remotely driving the tractor with the new software, and perhaps combine this with the debug print outs. Any obvious errors could be detected quite quickly in these tests, even special case errors could be detected here.

When all obvious and special case errors were handled, it was also good to collect some log data from a run with the tractor. The log data makes it possible to detect minor errors, for example, a minor error in speed calculation, which is hard to detect in any other way.
3.2 Equipment

Section 3.2 presents the equipment used for controlling the tractor.

Figure 6: Computer hardware setup.

Figure 6 shows the different computer devices used within the project, and how they are connected. The equipment in the tractor is mounted in a removable box placed behind the seat.

3.2.1 PC104

We have a PC104 on the tractor, which is a compact PC with a 700 MHz Pentium III processor, with 256 MB of RAM and a 40 GB hard drive. It has a network card, six serial ports and two USB ports. The Linux-based OS Xubuntu is installed on the PC104, it is POSIX compatible. Our controller program is running on the PC104.

3.2.2 Router

We use a D-Link 624 wireless router on the tractor, with an external Netgear antenna mounted on the roof of the tractor. The wireless network is WPA encrypted. The PC104 is assigned a static IP address, for convenience reasons.

3.2.3 ADAM -Modules

To control the speed, the steering, the lights and the wipers via remote interface, we need two kinds of devices which can be controlled by a computer [2]. First some controlled switches, and also a device with a controlled voltage as output.
We found those two devices in the ADAM-4000 series. They are devices communicating with a computer by a serial connection, providing different kinds of output. The following modules are used in the tractor:

- **4520 – RS232 / RS485 converter**
  This module is the link between a computer with a serial connection (RS232) and the other modules, communicating via a RS485 connection.

- **4060 – Relay output module**
  This module provides 4 controlled switches, 2 normally-open contacts, and 2 change-over contacts (one normally-open and one normally-closed with a common terminal). The two normally-open contacts are used for the lights and the wipers switches, and one of the change-over contacts is used to switch between the forward and reverse pedals.

- **4021 – Analog output module (x2)**
  This module provides a controlled voltage output. One module is used for the speed control, to send a voltage value to the controller. And another is used for the steering control, to send a voltage value to the steering valve.

The interconnection between the four modules and what they are controlling can be found in Attachment 1. The commands available for the modules can be found in [8].

### 3.2.4 DSP-3000 Gyro

We use the KVH DSP-3000 fiber optic gyro for measurement of the turning angular rate. It communicates via serial interface. The DSP-3000 is a single-axis interferometric fiber optic gyro [9]. The gyro measures angular rate of rotation, which can be integrated to allow the heading of the tractor to be measured.

### 3.2.5 GPS Receiver

We have a NovAtel SUPERSTAR II GPS mounted on the tractor. It can send binary or NMEA data, depending on the configuration. It sends data with a frequency of 1 message per second. The GPS receiver communicates via serial interface.

### 3.2.6 Speed Decoder

The speed decoder counts the pulses that are sent from the speed sensor on the tractor [2]. 400 pulses correspond to a movement of 1 m. The speed decoder is used for calculation of the tractor's speed. It is built with an ATmega162 microcontroller and communicates via serial interface. The commands available for the speed decoder can be found in Attachment 2.

### 3.2.7 Network Cameras

We use two D-Link DCS-900 network cameras for video feedback from the tractor [2]. They are stand alone devices, which are directly connected to the router on the tractor. The advantage with this is that the cameras don’t affect the performance of the PC104.

### 3.2.8 Safety Switch

The safety switch communicates via serial interface [2]. If the switch doesn’t receive any valid command it is open. The safety switch is connected to the ignition switch, the seat switch and the amber beacon on the tractor through a relay circuit. The safety switch is built
with an ATmega162 microcontroller and a small switch circuit connected to it, see Figure 7. The commands available for the safety switch can be found in Attachment 2.

**Figure 7:** Safety switch circuit.

### 3.2.9 Client Computer

Any modern PC with Windows installed on it and a wireless network card is suitable to serve as the client computer. To be able to view the GUI properly a screen resolution of 1400 x 1050 or above is required. We use a regular gaming joystick connected to the computer.
4 Result

4.1 Relay Logic

Parts of the relay logic on the tractor have been modified. The relay circuit schematics can be found in Attachment 3.

There are two switches mounted on the controller box; one on/off switch that turns on and off the equipment, and one auto/manual (a/m) switch.

- With the a/m switch in manual mode it’s possible to start and drive the tractor manually.

- With the a/m switch in auto mode it’s required that the controller program is started in order to start the tractor or to keep the engine on when one switches from manual to auto mode.

- If the a/m switch is in auto mode and the program is started, then the amber beacon on the roof of the tractor will be turned on, and the “seat switch”\(^1\) will be closed.

- When the a/m switch is in auto mode it’s not possible to drive the tractor manually since the regular steering and pedals is out of order then.

When the controller box on the tractor is removed there is no switch available to switch to auto mode. This is of great importance from a safety standpoint since the tractor from time to time is driven as a regular one, completely manual, in IceMakers daily activity.

---

\(^1\) The seat switch is a standard safety feature on the tractor. If no one is sitting on the seat, then any actuation on the accelerator pedal will have no effect.
4.2 Controller Program

The controller program, running on the PC104 in the tractor, is written in POSIX C. Figure 8 shows the programs different modules, and the connection between them. Each module except pc104_main consists of a .c and .h file. The .c file may contain thread (see section 2.2) functions, regular function or a combination of those. The .h file contains function prototypes, macros, structs and pthread_locks. The functionality of the modules is described in the sections that follow. Note that if a function name begins with the character *, then it’s a thread.

4.2.1 pc104_main

This module contains the main function of the program as the name implies.

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main()</td>
<td>Initiates ADAM Modules, open ports, creates the programs threads.</td>
</tr>
</tbody>
</table>

4.2.2 port_management

This module contains functions for handling of ports.

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int open_com(char port[], speed_t baud, char block)</td>
<td>Opens COM-port “port”, and sets the baud rate to “baud”. If “block” is “N”, then non blocking reading, else blocking. Returns the file descriptor on success, or -1 on error.</td>
</tr>
<tr>
<td>int open_socket(int port_nr)</td>
<td>Opens a socket on port number “port_nr” that allows a remote computer to connect to this program. Returns the file descriptor on success, or -1 on error.</td>
</tr>
</tbody>
</table>
4.2.3 current_status

This module can collect current status from most of the threads in the program, and send the information to a remote client. The available messages sent from this module can be found in Attachment 4.

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*joy_client_listener(void *arg)</td>
<td>Waits for a client to connect on tcp port 5000. If a client is connected, it starts a new thread of the type *send_joy_status().</td>
</tr>
<tr>
<td>*send_joy_status(void *arg)</td>
<td>Sends a string to the client with the current status of the joystick.</td>
</tr>
<tr>
<td>*dsp_client_listener(void *arg)</td>
<td>Waits for a client to connect on tcp port 5001. If a client is connected, it starts a new thread of the type *send_dsp_status().</td>
</tr>
<tr>
<td>*send_dsp_status(void *arg)</td>
<td>Sends a string to the client with the current status of the DSP3000 gyro.</td>
</tr>
<tr>
<td>*speed_client_listener(void *arg)</td>
<td>Waits for a client to connect on tcp port 5002. If a client is connected, it starts a new thread of the type *send_speed_status().</td>
</tr>
<tr>
<td>*send_speed_status(void *arg)</td>
<td>Sends a string to the client with the current status of the speed.</td>
</tr>
<tr>
<td>*regul_client_listener(void *arg)</td>
<td>Waits for a client to connect on tcp port 5003. If a client is connected, it starts a new thread of the type *send_regul_status().</td>
</tr>
<tr>
<td>*send_regul_status(void *arg)</td>
<td>Sends a string to the client with the current status of the steering regulator.</td>
</tr>
<tr>
<td>*gps_client_listener(void *arg)</td>
<td>Waits for a client to connect on tcp port 5004. If a client is connected, it starts a new thread of the type *send_gps_status().</td>
</tr>
<tr>
<td>*send_gps_status(void *arg)</td>
<td>Sends a string to the client with the current status of the GPS.</td>
</tr>
</tbody>
</table>

4.2.4 watchdog

This module checks that all threads in the program are running properly. This is partly done by checking that the parameter structure that the threads have are updated with an expected frequency. This test makes it is possible to detect if any unit is unplugged, or if a thread stops working for some other reason.

If the speed lever is above neutral for a short period of time and the speed given by the speed decoder module is zero, this is recognized as an error. With this test it is possible to detect if the connection from the “speed decoder” unit to the sensor on the tractor is broken, or if the sensor or “speed decoder” is malfunctioning.

If the watchdog detects any errors, it immediately starts a shutdown procedure. This procedure consists of three steps. The first is to stop the stream of messages to the “safety switch” in order to turn off the tractors engine. Next step in the procedure is to write a message to the program log (see Attachment 5) containing information of the type of error detected. The program log is very useful when performing error search. The last step is to reset
the ADAM modules, i.e. set their values to neutral. If the “safety switch” for some reason fails to stop the tractor, the last step will ensure that the tractor stops anyway. The shutdown procedure performed by the watchdog is used by all parts of the program. If, for example, another thread detects an error by itself, it will use the function close_program() in the watchdog that starts the shutdown procedure.

Table 6: watchdog functions.

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *watchdog(void *arg)</td>
<td>Checks that all threads are running properly, calls close_program() if an error is detected.</td>
</tr>
<tr>
<td>void close_program(char error_msg[])</td>
<td>Stops the flow of messages to the “safety switch” unit, writes a message to the program log, and requests resetting of modules.</td>
</tr>
<tr>
<td>void write_to_program_log(char error_msg[])</td>
<td>Writes a message to the program log.</td>
</tr>
<tr>
<td>void *safety_switch(void *arg)</td>
<td>Sends a message to the “safety switch” that keeps the switch closed for 100 ms. If the program stops sending the message, the switch will be opened, which turns off the engine and opens the “seat switch”.</td>
</tr>
</tbody>
</table>

4.2.5 joystick

This module handles the input from the remote client (the user). First it transforms the received message (the syntax of the message can be found in Attachment 5) into internal control parameters, such as drive speed voltage, steering angle, and active buttons. It then processes the commands according to the logic below.

To begin controlling the tractor, two criteria have to be fulfilled:

1. The speed lever has to be in zero position.
2. Button 1 has to be pushed.

If button 1 is released, the tractor stops and the criteria have to be fulfilled once again. Without these criteria it would be possible to start the tractor with an initial speed.

The speed is controlled with the lever. The steering is controlled by the stick. The lights and wipers can be turned on and off with Button 2 and Button 3 respectively. The steering regulator can be turned on and off with Button 9. Button 6 and Button 7 are used for changing the drive mode (forward or reverse). In order to change the drive mode the two criteria above have to be fulfilled.

Table 7: joystick functions.

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *joystick(void *arg)</td>
<td>Contains the handling of the users input. Communication via tcp port 4000.</td>
</tr>
<tr>
<td>int steering_reg_on()</td>
<td>Returns 1 if steering regulator is on, else 0.</td>
</tr>
</tbody>
</table>
### 4.2 Controller Program

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int get_drive_voltage()</td>
<td>Returns current drive voltage.</td>
</tr>
<tr>
<td>Double get_steering_angle()</td>
<td>Returns current steering angle.</td>
</tr>
<tr>
<td>int get_light_status()</td>
<td>Returns 1 if light is on, else 0.</td>
</tr>
<tr>
<td>int get_direction_status()</td>
<td>Returns 1 if direction mode is forward, else 0.</td>
</tr>
<tr>
<td>int get_joy_status(struct joymeas *joy_cpy)</td>
<td>Updates the arguments structure if there is new data. The structure contains current status information.</td>
</tr>
</tbody>
</table>

#### 4.2.6 adam_modules

This module controls the ADAM modules. It executes the actions requested by the joystick and steering_regulator modules.

**Table 8: adam_modules functions.**

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *adam_controller(void *arg)</td>
<td>Contains the handling of the ADAM modules.</td>
</tr>
<tr>
<td>void set_steering_speed(int speed, int port)</td>
<td>Sends the specified voltage (speed/10) to the voltage module 4021 with address 03 via serial port “port”.</td>
</tr>
<tr>
<td>void set_drive_speed(int speed, int port)</td>
<td>Sends the specified voltage (speed/10) to the voltage module 4021 with address 01 via serial port “port”.</td>
</tr>
<tr>
<td>void set_relay_output(char channel, char bit, int port)</td>
<td>Sets the channel “channel” to the value specified by “bit” (module 4060) via serial port “port”.</td>
</tr>
<tr>
<td>int check_module_stat(int port)</td>
<td>Returns 1 if all ADAM modules are connected, else 0.</td>
</tr>
</tbody>
</table>

#### 4.2.7 steering_regulator

This module contains the steering regulator. It uses the speed calculated by the speed_decoder module, and the yaw rate received by the DSP3000 module to calculate the regulator signal (see section 2.4), which in turn is used by the adam_module thread.

**Table 9: steering_regulator functions.**

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*steering_regul(void *arg)</td>
<td>Calculates the steering regulator signal, if the user has requested steering regulation.</td>
</tr>
<tr>
<td>void set_regul_signal(double r)</td>
<td>Sets the regulator signal to r.</td>
</tr>
<tr>
<td>int get_regul_status(struct regulmeas *rm_cpy)</td>
<td>Updates the arguments structure if there is new data. The structure contains current status information.</td>
</tr>
</tbody>
</table>
4.2.8 speed_decoder

This module calculates the vehicles speed by using the pulses received from the “speed decoder” unit. 400 pulses correspond to a movement of 1 m. The module consists of two threads; one that handles the communication with the “speed decoder” unit, and another that calculates the speed.

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *speed_sensor(void *arg)</td>
<td>Reads the data from the “speed decoder” and converts and store the part of it that contains the number of registered pulses.</td>
</tr>
<tr>
<td>void *calc_speed(void *arg)</td>
<td>Calculates the speed by using the number of registered pulses stored in the global variable current_pulses.</td>
</tr>
<tr>
<td>int get_speed_status(struct speedmeas *spd_cpy)</td>
<td>Updates the arguments structure if there is new data. The structure contains current status information.</td>
</tr>
</tbody>
</table>

4.2.9 DSP3000

This module receives data from the DSP3000 gyro via serial interface.

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *DSP3000(void *arg)</td>
<td>Receives data from the gyro and updates the status structure.</td>
</tr>
<tr>
<td>double get_angle_speed()</td>
<td>Returns the current yaw rate.</td>
</tr>
<tr>
<td>int get_gyro_status(struct gyromeas *oGyro_cpy)</td>
<td>Updates the arguments structure if there is new data. The structure contains current status information.</td>
</tr>
</tbody>
</table>

4.2.10 gps_receiver

This module receives data from the GPS via serial interface.

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *read_nmea(void *arg)</td>
<td>Reads the NMEA messages from the gps and updates the status structure.</td>
</tr>
<tr>
<td>int get_gps_status(struct gpsmeas *gp_cpy)</td>
<td>Updates the arguments structure if there is new data. The structure contains current status information.</td>
</tr>
</tbody>
</table>
4.3 Graphical User Interface

The graphical user interface (GUI) is written in Java. Figure 9 shows the relationship between the different classes in the program. The frames with dotted lines indicate a class that contains graphical components. The hollow diamonds on the arrows indicate an aggregation relationship, and are known as “has-a” relationship.

4.3.1 GUIFrame

This class contains the main window of the graphical user interface. It coordinates the other parts of the program. It provides methods for updating status information, and handling of mouse clicks on buttons and menu items. It also contains condition variables that affect the behaviour of other program parts. The status of the condition variables can be read or changed by other classes via public methods in GUIFrame. The GUIFrame has two **WBrowser** objects that provides two web browsers, which are used to show video from the network cameras. The popup windows OptionFrame, GPSdata, and AboutPopup can be reached via the menu provided by GUIFrame.

Example 3 shows how the GUIFrame is related to the threads **ConnectToPC104Thread**, **JoystickClientThread**, and **SocketReadLogThread**, and also gives a brief description of these threads.

**Example 3**

**User action:** The connect button in **GUIFrame** is pushed.

**Program response:** An ActionPerformed method is activated when the user pushes the connect button. This method performs the following:

1. Deactivates the restart server button.
2. Reads the IP address in the configuration file with a **ReadProgramConfig** object.
3. Connects to the server on the IP address specified in the configuration file, and starts the controller program by creation of a **ConnectToPC104Thread**. The **ConnectToPC104Thread** will in turn perform the following:
   a. Start a new PuTTY SSH session with connection to the IP address specified in the constructor, and execute the run command for the controller program.

4. Starts a new **JoystickClientThread** object in order to take control of the tractor. The **JoystickClientThread** will in turn perform the following:
   a. Load the joystick.
   b. Try to connect to the server provided by the controller program up to 30 times with an interval of 1 sec. If the connection succeed, the global condition variable lostConnection is set to false to notify the other program parts that the client is connected. Then 5 **SocketReadLogThreads** of different types are created to collect current status information. The **SocketReadLogThreads** will in turn perform the following:
      i. Connect to the controller program, to the IP address and port specified in the constructor for the object.
      ii. Collect current status information, and update status labels in the frames GUIFrame and GPSdata.
   c. If the connection attempt in step b. succeeded, then the client starts to send the control command (see Attachment 5), based on the input from the joystick, to the controller program.

### 4.3.2 OptionFrame

This class contains a graphical interface for handling of program options. The user can with this popup window change settings for the network cameras and the GPS. See section 4.4.7 for details about the settings.

### 4.3.3 AboutPopup

Is a popup window that contains general information about the program, such as program version.

### 4.3.4 GPSdata

This class contains the graphical popup window that shows GPS data, and a local map with the vehicles position path shown. It also handles button and slider input from the user. GPSdata provides a public method for updating of GPS data, this method updates status labels, and also make calls to functions provided by the **GPSmapThread** class in order to update the local map.

The conversion from geodetic coordinates to ENU coordinates described in section 2.1.5 is used to draw the vehicles path. The reference point is in the middle of the coordinate system. The path is created by the drawing of lines from the vehicles previous position to its current. If the current position of the vehicle is more than 4 scale units from the reference point, then the path is redrawn with the current position as the new reference point. This is done to prevent that the position of the vehicle end up outside the map.

**GPSmapThread**

This class contains methods for handling of the local map. saveCurrentPath() saves the current driving path to a file. loadPath() loads a path from a file, and show it on the map. updateMap() updates the drive path, and draws a new map picture by creating a new **CoordinateSystem** object. The class also contains a thread function that is activated when the
GUI is disconnected from the controller program. The thread updates the map according to the users input.

**CoordinateSystem**

This class handles the drawing of the map. The paintComponent() method utilize the standard Java class Graphics2D\(^2\) to draw the map. The map is drawn with four GeneralPath\(^3\) objects:

1. A coordinate system, 20 squared units big, (gray).
2. Vehicles position path, (red).
4. Path that indicates an area in the coordinate system, (black).

---

\(^2\) A Java class in the java.awt package.

\(^3\) A Java class in the java.awt.geom package, represents a geometric path.
4.4 GUI User Guide

4.4.1 Installation/File Structure

The program files are distributed in a folder called “IceMaker_I”. The content in the folder is shown in Figure 10. The folder “run” contains the runnable file “GUI_IceMaker_I.jar”. The folder “IceMaker_I” also contains the development specific folders.

![Figure 10: Content of the IceMaker_I folder.](image)

The Java Runtime Environment (JRE) is required to run the program. Free for download at: http://java.sun.com/javase/downloads/index.jsp.

The installation of the program is quite simple:
- Put the folder “IceMaker_I” containing the program files on the root of the C: drive.
- Use the shortcut “Run IceMaker_I GUI” found in the IceMaker_I directory to start the program.

4.4.2 Configuration File

![Figure 11: Configuration file “Config.txt”.](image)

Figure 11 shows the content of the configuration file. The “ServerIP” is the IP address that the client will connect to. “Map1” is the default map loaded into the “GPS Map” window (see section 4.4.5). It is possible to change the default map to “Map2” or “Map3” in the option window (see 4.4.7). The name of the maps should correspond to a file with map coordinates in the right format (see Attachment 6).
4.4.3 System Requirements

Intel Pentium 4 or equivalent processor
512 MB of RAM
Screen resolution of 1400 x 1050 or above recommended
Microsoft Windows XP
Microsoft Internet Explorer

Figure 12: Main window of the Graphical User Interface.

Figure 12 shows the main window of the graphical user interface, when the client is connected to the controller program on the tractor.

4.4.4 Component Description of the Main Window

Menu Items

Figure 13: Menu items.

Figure 13 shows the menu items available in the program.
The program is closed by clicking on “Exit” under “Menu”, this command can also be reached by the key combination Ctrl+Q.

By clicking on the “GPS Map” option under “View” the “GPS Map” window (see 4.4.5) is shown.

By clicking on the “Option” alternative under “Tools” the “Option” window (see 4.4.7) will appear.

By clicking on the “About” option under “Help”, a window containing current program version is shown.

**Buttons**

The “Connect” button seen in Figure 18 is used to connect to the controller program. The “R” button is used to restarts the controller computer. The button with the arrows is used to load the web cameras. The inactive button is used to abort a connection attempt.
The “Disconnect” button seen in Figure 19 is used to disconnect from the controller program. Note that the “R” button is inactive when the program is connected.

**Status Icons**

The icon shown in the bottom right corner indicates whether the program is connected to the controller program or not. Green icon indicates connected, and red disconnected.

![Status icons](image)

*Figure 20: Status icons.*

**Terminal Window**

<table>
<thead>
<tr>
<th>Joystick loaded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successfully connected to 192.168.0.102</td>
</tr>
</tbody>
</table>

*Figure 21: Terminal window.*

The terminal window seen in Figure 21 shows the program information for the user.

**Status Information**

<table>
<thead>
<tr>
<th>Speed: 0.0 Km/h</th>
<th>Regulator: On</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steering angle(R): 0.0 Deg</td>
<td>Wiper: Off</td>
</tr>
<tr>
<td>Steering angle(E): 0.0 Deg</td>
<td>Light: On</td>
</tr>
<tr>
<td>Drive mode: Forward</td>
<td></td>
</tr>
<tr>
<td>Pedal voltage: 0.8 V</td>
<td></td>
</tr>
<tr>
<td>Steering voltage: 5.8 V</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 22: Status information.*

Figure 22 shows the status information available. “Steering angle(R)” is the user’s reference angle, and “Steering angle(E)” is the estimated steering angle. “Regulator” is the steering regulator.
4.4.5 GPS Map

Figure 23: “GPS Map” windows.

Figure 23 shows two examples of the “GPS Map” window when the client is disconnected. The window to the left shows a map of a road outside Luleå University of Technology. This map is drawn with a number of GPS coordinates stored in the default map file (specified in the configuration file). The window to the right shows a path saved from a run with the tractor along the road.

4.4.6 Component Description of the GPS Map Window

Figure 24: Components of the “GPS Map”.

Status Information

The available status information is shown in Figure 24. The “A” after the status label indicates that there are valid data available at the moment. A “V” after the status label indicates that there are no valid data available. The format of the UTC time is hhmmss.s.
Buttons and Slider

Button available only when the client is connected:
- The “Ref. Point” button changes the reference point to current position of the vehicle and clears the old path.

Buttons available only when the client is disconnected:
- The “S” button saves the current path of the vehicle to the file “saved_path.txt” in the program directory.
- The “L” button loads a saved path from the file “saved_path.txt”.
- The arrow buttons are used to move the map around.

Components available no matter client connection status:
- The “Clear Map” button clears the vehicles path, the map loaded is left unchanged.
- The slider is used to change the scale of the map.

4.4.7 Option Window

Figure 25: “General Settings” tab.

Figure 25 shows the settings available in the “General Settings” tab. The default server IP address can be changed in this tab. The video from the network cameras can be turned on and off as well.

Figure 26: “GPS Map” tab.
Figure 26 shows the settings available in the “GPS Map” tab. The map shown in the “GPS Map” window can be changed in this tab. The three map options here can be changed in the configuration file “Config.txt”.

4.4.8 How to Connect to the Controller Program

- Connect to the router on the tractor via WLAN.
- Start the program by clicking on the shortcut described in 4.4.1.
- Click on the connect button in the bottom left corner of the main window. The program will then start a new PuTTY SSH session in the background, and try to connect to the controller program. Connection status is shown in the terminal window.

When a successful connection to the controller program has been established, then it’s possible to teleoperate the tractor with the joystick (see 4.4.10), provided that the tractor is started and the auto/manual switch inside the tractor is in auto position.

4.4.9 How to Disconnect from the Controller Program

Disconnection from the controller program can of course be forced by generating errors, for example, close the WLAN connection on the computer or unplug the joystick. The soft way to disconnect is by clicking on the Disconnect button, then a dialog window (see Figure 27) will appear, just click on the Yes button to disconnect.

![Disconnect dialog](image)

Figure 27: Disconnect dialog.

4.4.10 Joystick

To begin controlling the tractor, two criteria have to be fulfilled:

1. The speed lever has to be in zero position.
2. Button 1 has to be pushed.

If button 1 is released, the tractor stops and the criteria have to be fulfilled once again. Without these criteria it would be possible to start the tractor with an initial speed.

The speed is controlled with the lever. The steering is controlled by the stick. The lights and wipers can be turned on and off with Button 2 and Button 3 respectively. The steering regulator can be turned on and off with Button 9. Button 6 and Button 7 are used for changing the drive mode (forward or reverse). In order to change the drive mode the two criteria above have to be fulfilled.
5 Discussion

One part of the purpose was to examine the possibility to provide support for a number of new signals in the system, for example, water temperature and oil pressure, and consider the possibility to do this within this thesis. There has not been enough time to implement these kinds of signals. Although the division of the controller program into different modules has made it easier to add more devices and signals to the system in the future.

Safety is an essential part of the development of the control system. The watchdog implemented in the controller program has contributed to increased safety. Loose cables, connection errors, malfunctioning threads or devices should all be detected by the watchdog and lead to an immediate halt of the tractor. Even though the system safety has increased immensely, there is certainly a need to continually think about safety issues. A control system of a big and potential life threatening machine can never be too safe. The safety requirements of the system become even greater when autonomous control algorithms are implemented. Then sensors for detection of obstacles around the tractor are needed, to make it possible for the tractor to stop if something is in its way. Worth to mention here is that the tractor is supposed to work in a somewhat secluded area. But still, a control system like this should be prepared for worst case scenarios.

The graphical user interface has performed well during testing and shows all relevant information for the user. Due to the demanding processing of the video streams from the network cameras, I recommend to run the GUI on a fairly powerful computer in order to get stable real-time response.

5.1 Future Development

A future goal of the development of the control system is to make the tractor IceMaker I more autonomous. For example, make it possible to let the tractor drive on its own along a path drawn with GPS coordinates. Development and implementation of autonomous algorithms are needed to reach the goal.

- Sensors for detection of obstacles around the tractor have to be implemented in the system for both increased safety and better steering algorithms.
- Higher precision of the GPS position is needed; this can be achieved by implementing a Differential Global Positioning System (DGPS).
- An electronic map of the area where the tractor is aimed to work on has to be created. Methods for defining the tractors drive course on the map have to be developed.
- Equipment for control of the snow thrower in the back of the tractor may be needed.
- Equipment for control of the steering brakes on the tractor might be needed to be able to get full control of the tractor when driving on low friction surfaces.
- The WLAN has to be built to give an accurate coverage.
- More video feedback from the tractor may be needed.
- To increase the safety another “safety switch” could be mounted and implemented in parallel with the existing.
- A simple radio controlled switch should be mounted to the ignition switch for the possibility to stop the tractor if the regular system starts to malfunction.
References

[1] General Information about Arjeplog. URL:
http://www.arjeplog.se/index.php?option=com_content&task=view&id=60&Itemid=96

Luleå University of Technology, 2007.


[6] Håkan Fredriksson, Peter Danielsson, Sven Rönnbäck, Kalevi Hyypä. SNOWBOTS:
Gyro Guided Steering of a Teleoperated Tractor During Winter Conditions. Luleå
University of Technology, 2008.


2005.

Attachment 1: Interconnection Between ADAM Modules

- Pedal ground
- Pedal Relays
- Reference ground
- Signal voltage to the steering valve

- 4021 Speed Control
- 4021 Steering Control
- 4060
  - 0
  - 9
  - 8
- 4520
- PC104

12V
Attachment 2: Safety Switch & Speed Decoder Commands

Safety Switch Specification:

Serial settings: N, 57600, 8, N, 1

Safety Switch Command Table

<table>
<thead>
<tr>
<th>Command</th>
<th>Response</th>
<th>Switch closed for</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘1’</td>
<td>‘&lt;’&lt;CR&gt;</td>
<td>0.1 sec</td>
</tr>
<tr>
<td>‘2’</td>
<td>‘&lt;’&lt;CR&gt;</td>
<td>0.2 sec</td>
</tr>
<tr>
<td>‘3’</td>
<td>‘&lt;’&lt;CR&gt;</td>
<td>0.3 sec</td>
</tr>
<tr>
<td>‘4’</td>
<td>‘&lt;’&lt;CR&gt;</td>
<td>0.4 sec</td>
</tr>
<tr>
<td>‘5’</td>
<td>‘&lt;’&lt;CR&gt;</td>
<td>0.5 sec</td>
</tr>
<tr>
<td>‘6’</td>
<td>‘&lt;’&lt;CR&gt;</td>
<td>0.6 sec</td>
</tr>
<tr>
<td>‘7’</td>
<td>‘&lt;’&lt;CR&gt;</td>
<td>0.7 sec</td>
</tr>
<tr>
<td>‘8’</td>
<td>‘&lt;’&lt;CR&gt;</td>
<td>0.8 sec</td>
</tr>
<tr>
<td>‘9’</td>
<td>‘&lt;’&lt;CR&gt;</td>
<td>0.9 sec</td>
</tr>
</tbody>
</table>

Speed Decoder Specification:

Serial settings: N, 57600, 8, N, 1

The unit sends a message at every registered pulse and every second. All data is in hexchar format. Message layout: Data:Index,Time,CheckSum

Index – Represents the number of pulses received, 16 bit integer.

Time – Is the internal clock, 16 bit integer, cleared at 28800. The time has a scale of 1/28800 and overflows at 1s.

Checksum – Is calculated with: (index>>8)&0xff+index&0xff+(t>>8)&0xff+t&0xff

Speed Decoder Command Table

<table>
<thead>
<tr>
<th>Command</th>
<th>Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘Z’&lt;CR&gt;</td>
<td></td>
<td>Timer is set to zero.</td>
</tr>
<tr>
<td>‘R’&lt;CR&gt;</td>
<td></td>
<td>The unit does a soft reset.</td>
</tr>
<tr>
<td>‘P’&lt;CR&gt;</td>
<td></td>
<td>Shows device manual, clear index and time, and is polled.</td>
</tr>
<tr>
<td>“ECHOHelloWorld”&lt;CR&gt;</td>
<td>“HelloWorld”&lt;CR&gt;</td>
<td>Synchronize the communication with the unit. During ECHO the transmission of other messages are stopped.</td>
</tr>
</tbody>
</table>
Attachment 3: Relay Circuit Schematics

12 V from the tractors battery

Seat Switch

GND via safety switch

12 V to Box

Man/Auto 12 V

Auto On 12 V

Steering Valve On/Off (12 V)

To Ignition Relay

GND

GND to Box

Extra Light

12V

Module 4060 (Relay output)

Module 4021

Forward Pedal

Reverse Pedal

Front wiper

Rear wiper

12V
Attachment 4: Current Status Messages

Current status sent on port 5000:

<table>
<thead>
<tr>
<th>Name in message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAM Index</td>
<td>The index of the message.</td>
</tr>
<tr>
<td>Time</td>
<td>The time stamp of the message.</td>
</tr>
<tr>
<td>Pedal</td>
<td>Pedal voltage times 10 in Volt.</td>
</tr>
<tr>
<td>Steering</td>
<td>Steering valve voltage times 10 in Volt.</td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of the vehicle, 1 (forward) or 0 (reverse).</td>
</tr>
<tr>
<td>Wiper</td>
<td>Wiper status, 1 (on) or 0 (off).</td>
</tr>
<tr>
<td>Light</td>
<td>Light status, 1 (on) or 0 (off).</td>
</tr>
<tr>
<td>END</td>
<td>Indicates the end of the message.</td>
</tr>
</tbody>
</table>

Example:
ADAM Index: 6592 Time: 1212495645.636940 Pedal: 8 Steering: 58 Direction: 1 Wiper: 0 Light: 0 END

Current status sent on port 5001:

<table>
<thead>
<tr>
<th>Name in message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP3000 Index</td>
<td>The index of the message.</td>
</tr>
<tr>
<td>Time</td>
<td>The time stamp of the message.</td>
</tr>
<tr>
<td>Rate</td>
<td>The angular rate of rotation in degrees.</td>
</tr>
<tr>
<td>Heading</td>
<td>The vehicles direction in degrees.</td>
</tr>
<tr>
<td>END</td>
<td>Indicates the end of the message.</td>
</tr>
</tbody>
</table>

Example:
DSP3000 Index: 10815 Time: 1212495645.667428 Rate: -0.000022 Heading: -97.091704 END

Current status sent on port 5002:

<table>
<thead>
<tr>
<th>Name in message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEED Index</td>
<td>The index of the message.</td>
</tr>
<tr>
<td>Time</td>
<td>The time stamp of the message.</td>
</tr>
<tr>
<td>CalculatedSpeed</td>
<td>The vehicles speed in m/s.</td>
</tr>
<tr>
<td>CurrentPulses</td>
<td>The number of pulses registered by the “speed decoder”</td>
</tr>
<tr>
<td>END</td>
<td>Indicates the end of the message.</td>
</tr>
</tbody>
</table>

Example:
SPEED Index: 14067 Time: 1212495852.552298 CalculatedSpeed: 1.175 CurrentPulses: 11636 END
Current status sent on port 5003:

<table>
<thead>
<tr>
<th>Name in message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGULATOR Index</td>
<td>The index of the message.</td>
</tr>
<tr>
<td>Time</td>
<td>The time stamp of the message.</td>
</tr>
<tr>
<td>RefAngle</td>
<td>The vehicles reference angle.</td>
</tr>
<tr>
<td>EstAngle</td>
<td>The vehicles estimated angle.</td>
</tr>
<tr>
<td>RegulVal</td>
<td>The steering valve voltage times 10 in Volt.</td>
</tr>
<tr>
<td>RegulAct</td>
<td>Regulator active status, 1 (on) or 0 (off).</td>
</tr>
<tr>
<td>END</td>
<td>Indicates the end of the message.</td>
</tr>
</tbody>
</table>

Example:

Current status sent on port 5004:

<table>
<thead>
<tr>
<th>Name in message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS Index</td>
<td>The index of the message.</td>
</tr>
<tr>
<td>Time</td>
<td>The time stamp of the message.</td>
</tr>
<tr>
<td>UTC</td>
<td>The UTC time of the message, (h)mmss.sss format.</td>
</tr>
<tr>
<td>Date</td>
<td>GPS date, (d)mmmyy format.</td>
</tr>
<tr>
<td>Status</td>
<td>GPS data status, valid data (A) or invalid data (V).</td>
</tr>
<tr>
<td>Lat</td>
<td>The latitude in degrees.</td>
</tr>
<tr>
<td>latHem</td>
<td>The latitude hemisphere, N (north) or S (south).</td>
</tr>
<tr>
<td>Long</td>
<td>The longitude in degrees.</td>
</tr>
<tr>
<td>longHem</td>
<td>The longitude hemisphere, E (east) or W (west).</td>
</tr>
<tr>
<td>WGS84Alt</td>
<td>The altitude in meter.</td>
</tr>
<tr>
<td>Speed</td>
<td>The vehicles speed in m/s.</td>
</tr>
<tr>
<td>Heading</td>
<td>The vehicles direction in degrees, 0.0 to 359.9 degrees.</td>
</tr>
<tr>
<td>ActSat</td>
<td>Number of active satellites, 0 to 12.</td>
</tr>
<tr>
<td>PDOP</td>
<td>Position dilution of precision, 0.5 to 99.9.</td>
</tr>
<tr>
<td>HDOP</td>
<td>Horizontal dilution of precision, 0.5 to 99.9.</td>
</tr>
<tr>
<td>VDOP</td>
<td>Vertical dilution of precision, 0.5 to 99.9.</td>
</tr>
<tr>
<td>END</td>
<td>Indicates the end of the message.</td>
</tr>
</tbody>
</table>

Example:
GPS Index: 412 Time: 1212495644.807411 UTC: 121852.000 Date: 30608 Status: A Lat: 65.616921667 latHem: N Long: 22.137405000 longHem: E WGS84Alt: 66.5 Speed: 0.0 Heading: 0.0 ActSat: 7 PDOP: 3.8 HDOP: 2.4 VDOP: 3.0 END
Attachment 5: Program Log & Control Command Syntax

The program log is found in the file PROGRAM_data.log on the controller computer. An example of the program log is shown below.

```plaintext
### Fri Apr 11 16:34:37 2008
Client successfully connected

### Fri Apr 11 16:44:45 2008
Client request to close the program

### Fri Apr 11 16:44:45 2008
WATCHDOG CLOSE: Resetting of modules succeeded
```

The available error messages in the program log are shown in the table below.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Potential reasons for the error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEED SENSOR ERROR: Old data from speed sensor</td>
<td>The power cable or the serial cable to the “speed decoder” is unplugged.</td>
</tr>
<tr>
<td>SPEED SENSOR ERROR: Invalid signal from speed sensor</td>
<td>The “speed decoder” is disconnected from the speed sensor on the tractor. Or the speed sensor on the tractor is malfunctioning.</td>
</tr>
<tr>
<td>SPEED ERROR: “speed decoder” read error</td>
<td>The serial port on the computer is malfunctioning.</td>
</tr>
<tr>
<td>GYRO ERROR: Old data from gyro</td>
<td>The power cable or the serial cable to the gyro is unplugged.</td>
</tr>
<tr>
<td>DSP3000 ERROR: Port read error</td>
<td>The serial port on the computer is malfunctioning.</td>
</tr>
<tr>
<td>JOYSTICK ERROR: Old data from joystick</td>
<td>The client is disconnected.</td>
</tr>
<tr>
<td>JOYSTICK ERROR: Socket read error</td>
<td>The socket is malfunctioning.</td>
</tr>
<tr>
<td>ADAM ERROR: Lost connection to one or more ADAM modules</td>
<td>The power cable or the serial cable to the ADAM modules is unplugged.</td>
</tr>
<tr>
<td>WATCHDOG CLOSE FATAL ERROR: Failed to reset modules</td>
<td>The controller program is malfunctioning.</td>
</tr>
</tbody>
</table>

The syntax of the control command to the controller program is:

Data: AAA,BBB,CC<NULL>

AAA – Percentage of full drive speed.

BBB – Percentage of steering angle, where 50% is neutral, below 50% left turning angle and above 50% right turning angle.

CC – A 32 bit integer that contains information of which buttons that are activated. Button 1 is represented by bit 1 in the integer, if this is high then button 1 is activated. The same apply to button 2 to 32.
Attachment 6: Format of Coordinates in Map Files

The first row in the file has the following syntax:

Scale: (1 to 500) ReferencePoint: $X_{ECEF}$ $Y_{ECEF}$ $Z_{ECEF}$

Row 2 to n has the following syntax:

Coord: (latitude in degrees) (longitude in degrees) (altitude in meter, h)

Example:
Scale: 1.0 ReferencePoint: 2445837.9865538427 995004.202748357 5786498.142095433 Coord: 65.616333333 22.137283 58