Development of an Autonomous Path Tracking Forest Machine - a status report -

F. Georgsson, T.Hellström, T.Johansson, K.Prorok, O.Ringdahl and U.Sandström

UMINF 05.08
ISSN-0348-0542
Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

February 28, 2005

Abstract

In many respects traditional automation in the forest-machine industry has reached an upper limit, since the driver already has to deal with an excess of information and take too many decisions at a very high pace. To further automation still, introduction of semi-autonomous and autonomous functions are expected and considered necessary. This paper describes an ongoing project along these ideas. We describe the development of the hardware and software of an unmanned shuttle that shifts timber from the area of felling to the main roads for further transportation. A new path-tracking algorithm is introduced, and demonstrated as being superior to standard techniques, such as Follow-the-Carrot and Pure-Pursuit. To facilitate the research and development, a comprehensive software architecture for sensor and actuator interfacing is developed. Obstacle avoidance is accomplished by a new kind of radar, developed for and by the automotive industry. Localization is accomplished by a Kalman filter combining data from a Real-Time Kinematic Differential GPS/GLONASS and a gyro/compass. Tests conducted on a simulator and a small-scale robot show promising results. Tests on the real forest machine are ongoing, and will be completed before the end of 2005.

1 Background and Introduction

This paper describes an ongoing project of the design and development of an autonomous path-tracking forest machine. This kind of product is part of a long-term vision in the forest industry [3], of developing an unmanned shuttle that transports timber from the felling area to the main roads for further transportation. The main advantages with such a solution are lower labor costs and less ground damages and emissions due to the lower weight of an unmanned vehicle (the cabin alone weighs several tons). The general requirements and conditions for the development of this kind of product are not addressed in this paper. It focuses instead on one of the necessary components: autonomous navigation, which involves sensing and moving safely along a user-defined path in a dynamic forest environment.

Unmanned vehicles in off-road use have been for long an active area of research and development. The mining company LKAB has been using vehicles in underground mines for many years, with reflective markers to aid the laser-based navigation system. Due to safety reasons combined with high demands on availability, these systems are no longer in full commercial operation. Stentz et.all develop optical techniques for localization of underground coal mining machines in [15]. Localization techniques of autonomous forest machines based on a combination of odometry and artificial visual landmarks are described in [9]. A development project for an autonomous Christmas tree weeder has been initiated with a feasibility report [4]. Requirements and system design for a robot performing selective cleaning in young forest stands is described in [17].

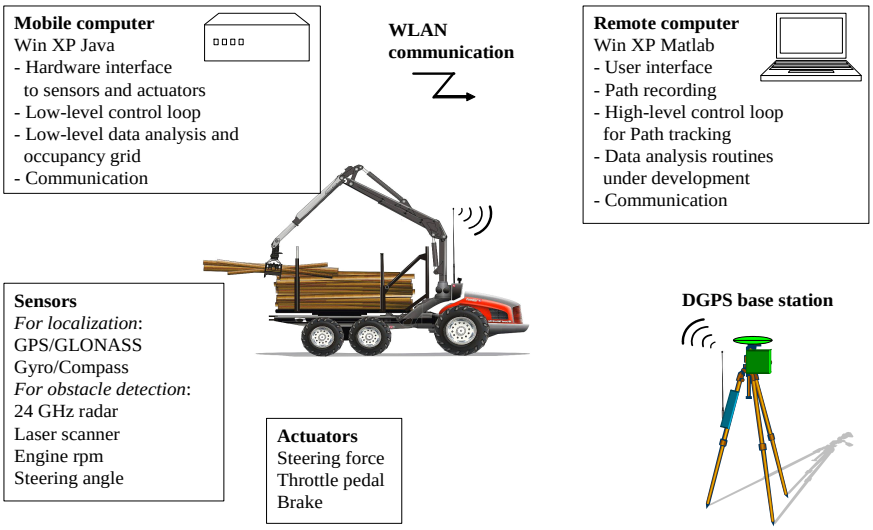


Figure 1: Overview of the architecture of the developed system. The high- and low-level parts are split between two computers. The shown forest machine is a vision of what a future autonomous vehicle would look like.

The initial design and ideas underlying the project reported in this paper are described in [5]. The resulting system has two modes of operation: *Path Recording*, in which the human operator drives or remote controls the vehicle along a selected path back and forth from the area of felling to the transportation road. In this phase, position, speed, heading and the operator's commands are recorded in the vehicle computer. When the vehicle has been loaded with timber (this subtask could also be done autonomously, but is not considered in this project) the operator activates *Path Tracking* mode, which means that the vehicle autonomously drives along the recorded path to the transportation road. In this paper, we describe the hardware and software developed and implemented for a pilot study on a Valmet 830 forest machine (forwarder) supplied by our industrial partner Komatsu Forest AB. The presented project started January 2003 and will end by July 2005. A continuation of the project is planned. Section 2 gives an overview of the developed hardware and software. The general design ideas behind a developed software system for sensor interfacing and actuator control are briefly described in Section 3. Section 4 describes how sensor fusion for position, heading, and obstacle detection is implemented. A novel algorithm for path tracking is described in Section 5. General experiences and directions for the future work finalizes the paper in Section 6.

2 System Overview

A block diagram of the developed system design is shown in Figure 1. The major building blocks are described in this section.

2.1 Computers and software

The system runs on two computers: the one placed on the vehicle is responsible for hardware interfacing and low-level data processing, such as data fusion in an occupancy grid (see Section 4.2). This computer communicates by a regular wireless local area network (WLAN) with the operator computer running the high level path tracking algorithms and the user interface. The two computers run with Windows XP at present, although it is designed so it can be moved to other OS:es, for instance UNIX or LINUX. This operating system independence comes from the choice of implementation language, Java and Matlab, as programs written in these languages are easily moved between different platforms. However, certain low-level drivers in C++ and the Java-Matlab interface will have to be converted. The Java version used has varied from 1.1.8 up to 1.4,

with little or no problems. The upgrading was needed partly for incorporating new features in Java, and partly to overcome compatibility problems with Matlab. In general, low-level processing and communication is implemented in Java, and high-level processing in Matlab. However, during development the Matlab environment has been used also for typical low-level operations, such as Kalman filtering. With a top-level cycle time of around 100 ms, it has not presented any problems, and it simplifies the research work considerably. In a final productified version most processing should be implemented on the mobile computer, with the operator computer in charge of the user interface only.

2.1.1 Communication

As mentioned in the previous section, the modules of the system may reside on different computers. The communication routines take care of the data routing, and make the actual location of each module transparent to the other modules. Communication between separate computers is done by Ethernet network, either directly through a cable or via a Wireless Local Area Network (WLAN). The WLAN equipment is the standard 10-to-54 Mbps hardware used for offices and homes. The WLAN is used for controlling the vehicle, but since the communication handling is transparent to the system, debug and in-office tests can be done by either a cable or direct communication within the computer.

The network communication uses datagrams (by the Internet UDP protocol), i.e. small packets of data transmitted with no control over their arrival, and therefore no acknowledge of received packets is obtained. The alternative would be TCP streams, which control the order, integrity, and completeness of the data. The reason for this choice is threefold:

1. Datagrams can be broadcast to more than one receiver; so several computers can 'listen in' on data from the sensors.

2. Datagrams delineate the data on the network. The data is conveniently seen as small units, as opposed to a data stream, where the software would have to find the beginning and end of each data packet.

3. With datagrams, lost packets are quickly replaced by new data. A streaming model would resend lost packets, but would also impose a variable delay, and with many resends there would be new data available by the time the original one arrives. TCP also uses bigger packets. Small packets have a greater chance of getting through the network, and the loss of one or several packets is tolerable. Up-to-date data is often more important than a complete data stream. For protection against a complete loss of control should the network fail, there are several time-outs built into the system. If control packets stop arriving at the vehicle for more than a preset time period, the vehicle automatically stops.

The amount of data that travels through the network is very small, currently the largest packets are about 1200 bytes, and are transmitted every 200 ms. (GPS packets contain a lot of position, velocity, and error data). Most packets contain only a few bytes of payload, usually one reading of a sensor. A conservative estimate on the communication bandwidth needed is about 200 Kbps. (20 different types of packets, 100 bytes per packet, 100 ms. period). This amounts to 2% of the bandwidth of a 10 Mbps. link.

The delay in the network is difficult to measure, since the two participating computers usually do not have synchronized clocks, but the estimate from preliminary tests is of less than 10 ms. Further development of a common time source will make these measurements more accurate. The WLAN range has not been tested separately, but tests with the forest machine found it to be at least 50 meters. This is too short a range for the intended use of the system, and further testing of different radio systems and antennae will be carried out.

2.2 Sensors

Sensors are primarily required for localization of the vehicle and obstacle detection. The performance of various sensor types and analysis algorithms is closely tied to the physical vehicle, on which the sensors are mounted. This means that the approach with multiple target machines during development (further described in Section 2.4) is of limited value for evaluation and development of sensor hardware/software. E.g., the range of sensors is normally fixed, and can not be

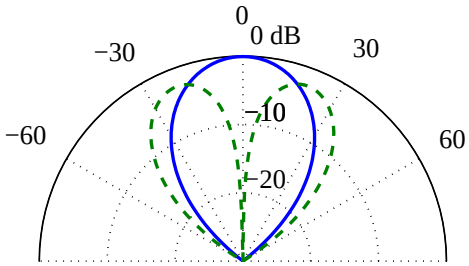


Figure 2: Switchable antenna characteristics for the sequential lobing radar. The relative strength between the two patterns S_Σ (solid line) and S_Δ (dashed line) is used to compute the bearing to the target.

scaled up in the same way as path tracking for example. Following is a description of the sensors chosen for the final target machine, the forest machine.

2.2.1 Radar

One main sensor type for obstacle detection on the forest machine is radar. The advantage of a radar (radio detection and ranging), compared to sensors based on ultrasound or light, is the ability to view obstacles reliably during bad weather conditions, like snow, fog or rain. One of the radar types used in the project is a pair of the Sequential Lobing Radar C1, produced by Tyco - M/A-COM, USA (primarily for the automotive industry). This radar works with 24 GHz frequency and is based on the monopulse theory [12]. Target range and bearing can be obtained by transmitting and receiving pulses. The range is estimated by matching the received pulse to an internally time-delayed one. An estimation of target-bearing can be obtained by using a receiver antenna with switchable lobe characteristics (Figure 2), S_Δ (slightly more sensitive diagonally) and S_Σ (mainly forward). The bearing is found by calculating the Additive Sensing Ratio, ASR, defined by

$$ASR = \frac{S_\Delta - S_\Sigma}{S_\Delta + S_\Sigma}, \tag{1}$$

and using a look-up table combined with phase information to distinguish between the two possible directions.

The transmitted beam width is ± 8 degrees vertically and ± 65 degrees horizontally, and the typical detection range is specified as $0.2 - 30$ meters, with a 15-cm resolution. Maximum range depends on weather conditions, the object's radar cross-section area, bearing, distance, and reflectivity. Objects containing metal or carbon are easiest to detect. The radar's physical size is 103x71x32 mm, and the transmitted power is less than 17 dBm EIRP. A Digital Signal Processor (DSP) for simultaneous tracking of up to 10 targets is included in the unit, and the results are presented via a CAN-bus every 30 ms. The target's relative amplitude is also given. The test results in Figure 3 show how a stationary radar located at the origin tracks a person walking in a zigzag pattern at various distances, between 1 and 15 meters. The person is detected at up to 9 meters away from the radar, when the bearing angle is within approximately ± 30 degrees. The results show good reliability and accurate positioning. In the final application, two radars will be mounted to cover the areas slightly to the left and right of the vehicle's front.

2.2.2 Satellite Navigation

An RTK DGPS (Real-Time Kinematics Differential GPS) satellite-navigation system, Javad Maxor, is the primary sensor for the vehicle's position. A stationary GPS receiver is connected by a radio link to a mobile GPS receiver (see Figure 1). Correction signals for timing, ionospheric and tropospheric errors are transmitted by radio from the stationary to the mobile GPS, resulting in a centimeter accuracy under ideal conditions. The Javad receiver is capable of receiving signals from both American GPS system and Russian GLONASS system. While providing a lower accuracy than the GPS, GLONASS provides important backup, especially at high latitudes (64 degrees north), at which the work has been conducted.

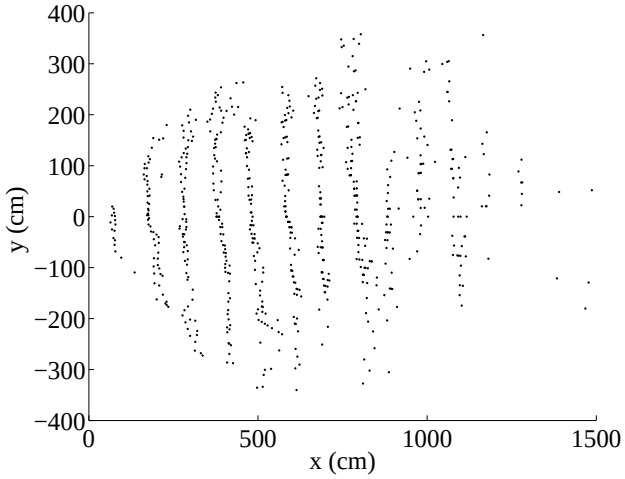


Figure 3: Detection of a person moving in parallel to the y-axis at distances 1, 2,..., 15 meters away from the radar positioned in the origin, facing along the x-axis. The radar detects the person within approximately ± 30 degrees between 1 and 9 meters.

2.2.3 Gyro/Compass

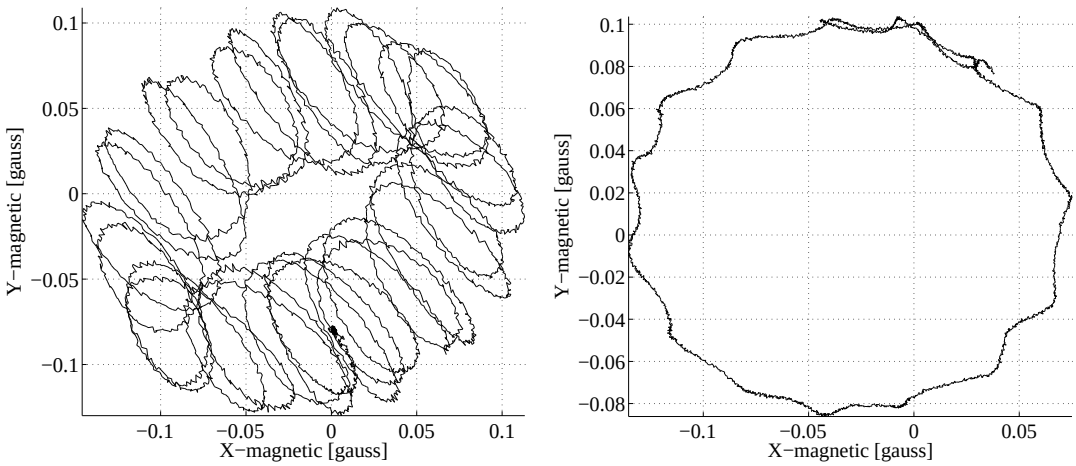
To estimate the pose in general and heading in particular an electronic compass/gyro of Crossbow Technology is used. The model, AHRS 400CC, is a nine-axis measurement system with linear accelerometers, rotational rate sensors, and magnetometers. Regarding the accelerometers, it is desirable to place the device as close to the point of the vehicle rotation as possible. For an aeroplane this means the center of gravity, and for a forwarder it is as close to the ground as possible. For this reason, the AHRS device was initially placed on the floor in the driver's cabin. After calibration, the magnetic field vectors in x and y directions from the magnetometers appeared as shown in Figure 4(a). In an ideal case, the figure should have been a circle centered at the origin. The tilted elliptical shape of the response is due to soft iron effects (i.e. the magnetic field lines are altered by magnetically soft material around the compass). The fact that the response is not centered at origin is due to hard iron effects (i.e. the magnetic field lines are set off by magnets and magnetized metal around the compass). The AHRS device is supposed to eliminate soft and hard iron effects by calibration, but due to the periodical disturbance that shows up as epicycles in the figure, the built-in calibration does not work. The epicyclic disturbance stems from the transmission of the forwarder, and since the AHRS device was placed on the floor of the driver's cabin, it was rather close to the rotating transmission. The frequency of the disturbance is consistent with the frequency of the transmission. Since the amount of magnetic metals is several magnitudes higher in a forwarder than in a small-sized aeroplane, for which the AHRS device was originally constructed, the conclusion is that great care has to be taken in the positioning of the device.

After moving the device about 2 meters vertically and placing it on the roof of the forwarder, the field vectors appear as in Figure 4(b). The disturbance is still visible but the amplitude has been drastically reduced. The gyro is combined with other sensors in a Kalman filter as described in Section 4.1.

2.3 Actuators

The forest machine is entirely controlled via an industrial communications bus, a CAN bus. Widely used in automotive systems, it requires only a single connection from our hardware to the machine control computer (actually several computers, connected by the CAN bus). Various hardware components enable the system to control engine throttle, turning rate, gear selection, etc. Some sensor data is also received through the bus, e.g. engine rpm., current turning angle, and actual throttle level.

The forest machine, a Valmet 830 as shown in Figure 5, is equipped with an articulated joint for steering. We have implemented a simple proportional integrating (PI) controller that takes care of controlling the steering angle by controlling the joint.



(a) cabin (b) roof

Figure 4: The measured magnetic field vector a) in the cabin of the forwarder, and b) on the roof of the forwarder. Both measurements are taken while completing at least a 360° turn.

The forward kinematics equations, which describe how the vehicle moves in response to changes in steering angle and wheel speed, have a known and simple form but may vary depending on varying load and ground conditions. E.g.: a forest machine with no load tends to move the light rear part more than the heavy front part when turning.

2.4 Development Strategy

Testing algorithms on the full-size forest machine is both impractical and inefficient. Therefore, the work has been conducted on four different target machines, each with increased complexity. As shown in Figure 5, the same main program can control any of the four target machines through a software switch board. Likewise, sensor data passes from the target machine to the main program. In this way, high-level routines like path tracking are easily developed and implemented by the use of a simple simulator [13]. The simulator implements no sophisticated sensor models, and has a simplified kinematics model for propulsion, but serves very well its purpose for debugging and testing the path-tracking algorithm described in Section 5. The user interface is also easily developed using the simulator as the target machine. The infrastructure for sensors or actuators (see Section 3), and the modules for communication between the two main computers are most conveniently developed on the small-size Pioneer AT2 robot [14]. Various types of sensors are also evaluated on this target machine. The third target machine, the Smart Pioneer AT2, has a dynamics (software determined) that more closely mimics a real forest machine, and is used for more realistic tests of the routines for turning, speed setting, and path tracking. In the current phase of the project, the system is moved to the real forest machine, and the routines for vehicle control are fine-tuned and tested. Also, reliable sensor tests are only possible using this final target machine.

3 Infrastructure for Software Development

As part of the development work, a general framework for development of software for robots and autonomous vehicles has been constructed. A modular structure was designed with a set of requirements in mind:

- Not having to change any code in the system when replacing one sensor (e.g. a speed sensor that gets data from the wheel encoders) by another sensor (e.g. a speed sensor that gets its data from a GPS receiver). Actually the 'user' of speed data may remain oblivious to the sensor in use.
- Not having to change any code when using different vehicles.

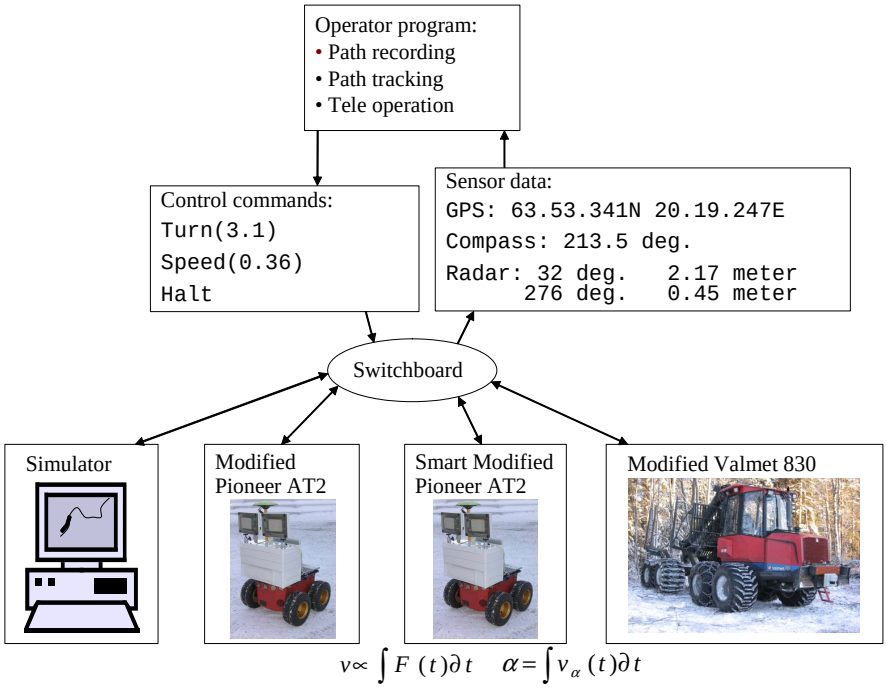


Figure 5: The work has been conducted on four different target machines, each with increased complexity. This approach greatly simplifies the research and development of both hardware and software.

- Operating the system on one or more computers, connected by network. The network should be invisible and the control algorithms should not have to be changed if some or all modules are moved to another computer. This feature makes it simple to just add another computer if some modules require more computing power.
- The analysis, design, and implementation should be object-oriented, to suit the modular design.

3.1 Modules

The forest machine system is made up of a number of software modules, some of which also represent real hardware devices. Each module can be independently loaded into the running system, and be logically connected to other modules. There are currently six major categories of modules:

- Sensors – represent hardware units that deliver sensor data, e.g. speed, heading, position.
- Actuators - represent hardware units that control external equipment, e.g. throttle, steering angle, and brakes.
- Vehicle – several implementations of real and virtual vehicles. See Figure 5.
- Control Panels – present data on the screen, or allow the vehicle to be manually controlled.
- Controllers – process sensor data and compute control signals for actuators.
- Proxies and Servers – facilitate transfer of sensor data and control commands over a network (Ethernet or WLAN). These modules hide the actual structures needed to use the network, so the system can have the same look and functionality whether used over a network or not.

Each module is implemented as one or more classes. The modules are connected primarily by an event-driven system. There is no central control loop running (except for the top-level Matlab path tracking loop described in Section 5). Instead the system reacts to changes in its environment.

For example, a sensor signal arrives and sets up a series of method calls that ultimately leads to a change in the state of the system. Some sensor signals might just update an internal map of the surroundings, while others might stop the vehicle immediately. Other events are timed events, i.e. some action is performed repetitiously. Operator input for pure tele-operation of the vehicle is handled in the same way: a push of a button in the GUI (Graphical User Interface) or a real joystick sets up a chain of events. The Matlab program responsible for path tracking runs on another computer, and is not part of this event-driven system. Instead it polls the sensor system, in a traditional fashion, in its main control loop.

A module usually executes in a separate thread, i.e. all the modules run in parallel. Most sensing, control, user interface, and behavior are defined in modules. They communicate with other modules through connections, either by sending commands directly to another module or by listening on other modules. They also have other properties that depend on their actual type, for instance update interval for sensors, and network addresses for servers and proxies.

An example: a Control Panel interested in getting speed data from a Speed Sensor gets a reference to the speed sensor, and installs itself as a listener on that sensor. When the sensor has new speed data, it informs all listeners on that sensor. A Control Panel might also send commands to other modules, for instance to a Vehicle. After getting a reference to the correct vehicle module it can send commands directly to it by calling some of the methods the vehicle has. Which methods are available for which type of module is controlled by an interface that describes the methods each module must implement. A Speed Sensor must have a method for installing speed listeners, for instance.

3.2 Configuration Manager

Every module has an associated configuration file with properties that control the module's behavior. For a sensor, this file may contain e.g. pose, network address, and filter constants. Most of these properties are set once and for all, while others are changed either by the user or by the module itself. The module can also save the changes so they take effect upon the next time the system is run. One example would be an experiment to find the most appropriate filter constant for a specific sensor; when a suitable value is found the sensor can store it in its configuration file. The next time the sensor is run, it automatically uses the saved value.

The forest machine system contains more than 60 different initialization files, and to facilitate changes, as well as for providing an overview, a graphical configuration manager is under development. The configuration manager gives the user a graphical picture of how every module is connected to other modules, and can also be used to modify individual properties for a module. It is also possible to reroute connections, add new modules, duplicate exiting ones, and remove them. One example showing a small part of the modules in the forest machine project is shown in Figure 6.

The files are stored in separate file folders, one per configuration. Together with the initialization files there is a configuration file that describes the system and which modules to load. Usually this start-up configuration file is stored together with the initialization files for the modules to load in that particular configuration. To start the system, a small boot-loader program reads a configuration from a file, and proceeds to load the appropriate modules. The user can select the configuration to load from a menu, or its name could be hard-coded into the boot loader. In this way, different versions (choices of sensors, filters, etc.) can be easily available during the development work.

4 Sensor Fusion

Fusion of different, simultaneously sampled, sensors typically works by estimating the relative accuracy, and then weighting or even completely disregarding sensors that, for the moment, are unreliable. One example of such sensor fusion is a Kalman filter for estimation of the vehicle's heading and position, based on simultaneously sampled data from GPS, gyro, compass, and the vehicle's steering angle and speed.

Fusion over time works with multiple read-outs from the same sensor (i.e. a time sequence) and results in one piece of information, normally some interesting property of the vehicle's environment.

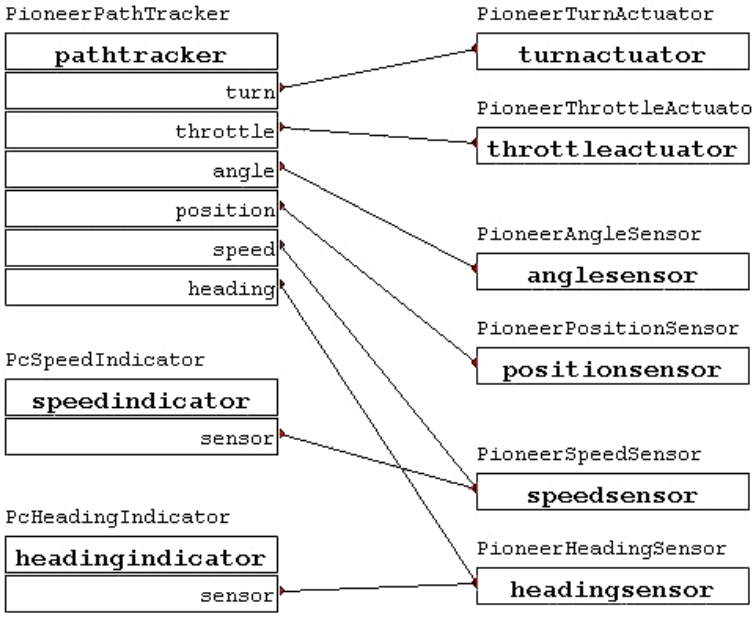


Figure 6: An example of the way Configuration Manager is used to declare the specific choice of sensors for an application. The properties of each module can be set by clicking the corresponding symbol.

A typical example is the creation of an occupancy grid, where each grid element contains the probability that the physical area corresponding to the grid element is occupied by an obstacle.

The implementation of the two examples given above is briefly described in the following two subsections.

4.1 Heading and Position

The heading and position of the forwarder plays an important role in the control of the vehicle. The position is obtained from the RTK-DGPS system (Section 2.2.2). The heading can be estimated by the AHRS (Section 2.2.3) and the RTK-DGPS system. However, the AHRS device yields a magnetic heading, while the RTK-DGPS equipment yields a geographic heading. There is a systematic difference (declination) between the two headings, but given the latitude and longitude, this difference can be calculated and accounted for. Unfortunately there is a substantial local deviation in the declination due to geomagnetic properties of Earth, and these also have to be accounted for. In Umeå, Sweden, where the prototype is developed, the declination is approximately 5.5 degrees eastwards. The declination-correction of the heading is macde in a preprocessing step.

To calculate the best possible estimate of the heading and position, we use a Kalman filter [8] with the following state equations:

$$
\begin{aligned}
x_{t+1} &= x_t + \frac{1}{2}\Delta t^2 a_T \cos\psi - \frac{1}{2}\Delta t^2 a_N \sin\psi \\
y_{t+1} &= y_t + \frac{1}{2}\Delta t^2 a_T \sin\psi + \frac{1}{2}\Delta t^2 a_N \cos\psi \\
\psi_{t+1} &= \psi_t + \Delta t \omega_t
\end{aligned}
$$

where a_T is the acceleration in the tangent-direction, and a_N is the acceleration in the normal direction. (x, y) denote the position, ψ the heading, and ω the rate of rotation. Δt is the time lapse between two consecutive updates of the Kalman filter. The different state variables are estimated as shown in the table below.

State variable	Estimated by
a_N and a_T	AHRS
x and y	Javad GPS-system
ω	AHRS, vehicle parameters
ψ	AHRS, Javad GPS-system, from Δx, Δy analysis

"Vehicle parameters" mean parameters that can be easily obtained from sensors mounted on the vehicle. ω is estimated with such vehicle parameters, namely α (turning angle) and v (velocity) according to

$$\omega = v\frac{a\cos\alpha + b}{\sin\alpha}$$

where $a \approx 1.6\ m$ and $b \approx 3.6\ m$ are the distances from the wheel axis to the articulated joint for the front- and rear-part of the vehicle respectively. In addition to the internal measurement of v, estimates of the velocity can be obtained from the GPS-system, and by integrating the accelerations obtained from the AHRS device.

Since the normal positions obtained from the RTK-DGPS system have centimeter accuracy it is possible to estimate ψ by calculating the direction of the vector $\begin{bmatrix} x_{t-\delta} - x_t \\ y_{t-\delta} - y_t \end{bmatrix}$, where $\delta \geq 1$. A larger value of δ improves the estimate, but reduces the response to quick changes in direction.

Further estimates of ω and v (that is Δx and Δy for a given Δt) are possible to obtain by analyzing other sensor data, for instance the optical flow in video sequences.

To solve the Kalman equations, information of the variance of the measurements and the system is needed. These are obtained by testing and simulation. Furthermore, the GPS-system supplies estimates of the current variance in the returned position used in solving the Kalman equations.

The theory behind the Kalman filter assumes normal distributed driving noise (i.e. white noise). This requirement is clearly violated when the GPS for instance, switches between various modes, primarily caused by occlusion of the satellite signals. To handle these situations, a different observer noise is set in a pre-processing stage depending on the run mode of the GPS-equipment.

4.2 Occupancy grid

The occupancy grid is 2-dimensional, and the obstacle detection subsystem currently works under a flat-ground assumption. Since it is only targeted at obstacle avoidance by design, and not at global map building, it can be made relatively small (e.g. 100x100 cells) and also move along with the vehicle. Although the system is working under a flat-ground assumption, the vehicle and range sensors are not assumed to be parallel to the flat ground. Their poses are fully 3-dimensional with 6 degrees of freedom, and the subsystem is designed to handle any pose set to a sensor. Although the pose of each range sensor has 6 degrees of freedom, a range sensor is approximated to have a 2-dimensional *sensing plane*, aligned with the xy-plane of its pose, where the x-axis equals the line-of-sight. The part of the sensing plane within the field of view and maximum range are referred to as the *sensor wedge*.

A pose expresses a frame of reference in the coordinates of another frame of reference. It represents a directed relation between the two frames, and consists of a 3D rotation and a 3D translation. In every pose relation, it is also possible to determine a parent and a child. For example, consider the range sensor pose (in relation to its mounting plate). In this case, the mounting plate is parent and the range sensor is child. These relations finally form a tree structure of poses, where the root object is the ground, and the leaves are all the range sensors.

In the current implementation, all poses, except the vehicle pose, are assumed to be static. Since the ground → vehicle → mounting plate → sensor form a chain of relations, there are situations that require to collapse them into a shorter chain, for efficiency reasons. This is done in two separate phases. First, at system start-up, all static poses are collapsed as far as possible, and then the remaining dynamic poses are frequently collapsed into a collection of poses between the ground and each sensor, as the vehicle moves onward. Since poses are internally represented by ordinary 4-by-4 homogenous matrices, pose collapsing is carried out by simply multiplying the matrices together, as explained below.

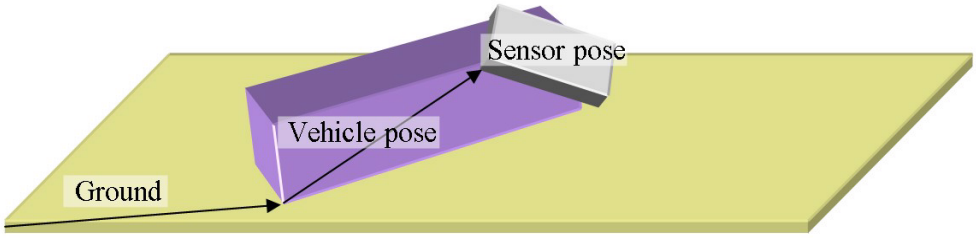


Figure 7: A visual example of a sensor pose expressed in relation to a vehicle pose. The arrows symbolizes the translation vectors that are one of the two components in a pose definition.

$$
\begin{aligned}
r &: \text{Roll} \\
p &: \text{Pitch} \\
h &: \text{Heading (aka Yaw)} \\
x, y, z &: \text{Translation vector components} \\
R &: \text{Rotational matrix} \\
T &: \text{Translational vector} \\
P_{Parent \to Child} &: \text{Pose represented by an ordinary homogenous matrix}
\end{aligned}
$$

Given the definitions above, a pose chain collapse is carried out according to the formulae below.

$$
\begin{aligned}
R &= \begin{bmatrix} \cos p \cos h & -\cos r \sin h - \cos h \sin p \sin r & \sin r \sin h - \cos r \cos h \sin p \\ \cos p \sin h & \cos r \cos h - \sin p \sin r \sin h & -\cos r \sin p \sin h - \cos h \sin r \\ \sin p & \cos p \sin r & \cos p \cos r \end{bmatrix} \\
T &= \begin{bmatrix} x \\ y \\ z \end{bmatrix} \\
P &= \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \\
P_{Ground \to Sensor} &= \underbrace{P_{Ground \to Vehicle}}_{Dynamic} \cdot \underbrace{P_{Vehicle \to Sensor}}_{Static}
\end{aligned}
$$

The implementation of the occupancy grid is inspired by the implementation by Novick [11], where the vehicle is always positioned somewhere inside the mid cell of the map. When the vehicle crosses a borderline to a neighboring cell, the contents of the map is shifted one cell, and the vehicle can remain in the center cell of the grid. The data that gets shifted out of the map is discarded. It is noteworthy that the map is never rotated. Instead the vehicle rotates and moves within the center cell, and data is always shifted in whole cell steps. In this way blurring caused by the vehicle movement is completely avoided.

The sensor fusion is based on a common (and naive) Bayesian approach for updating, as described by Movarec [10], where all individual sensor readings are assumed to be mutually independent, between sensors and over time.

For each cell, the ratio of the probabilities for being occupied (o) and not being occupied ($\neg o$) is updated after a new sensor reading B according to

$$
\frac{p(o|A \wedge B)}{p(\neg o|A \wedge B)} = \frac{p(B|o)}{p(B|\neg o)} \cdot \frac{p(o|A)}{p(\neg o|A)}
$$

where A denotes the set of all previous range sensor readings. The usage of this *odds form* can be further simplified by taking the logarithm:

$$\underbrace{\log \frac{p(o|A \wedge B)}{p(\neg o|A \wedge B)}}_{\text{Updated cell content}} = \underbrace{\log \frac{p(B|o)}{p(B|\neg o)}}_{\text{Sensor model}} + \underbrace{\log \frac{p(o|A)}{p(\neg o|A)}}_{\text{Cell content}}.$$

In this way, the grid cell updating operation is reduced to simply adding a real number, where a positive value suggests an obstacle, and vice versa. By using binary logarithms it is possible to make highly efficient implementations in integer arithmetic (however, the current implementation uses floating-point arithmetic in Java).

In this implementation, each range sensor software module must supply a Bayesian sensor model that can yield the probability odds $\frac{p(B|o)}{p(B|\neg o)}$ for a given range value and position on the range sensor wedge. Movarec also discusses and evaluates a theoretically more correct technique, where previous readings are also considered in the sensor model, $\frac{p(B|o \wedge A)}{p(B|\neg o \wedge A)}$, but concludes that the cost is significant and the practical benefit seems to be of minor importance. The possible success of Bayesian updating depends to a high degree on the accuracy of the sensor models.

When it comes to performance improvement, the primary task is to reduce the number of cells to update. Optimally, only the cells lying directly under each sensor wedge should be updated. In the current implementation, this is accomplished as follows: to determine which cells are directly under the sensor wedge, each sensor model gives a boundary point list to the grid module at system startup. The boundary point list, which outlines the sensor wedge, is represented in local sensor coordinates. Before each grid update takes place for a given range sensor, the boundary points are transformed into grid coordinates, and a grid scan table is then constructed, which guarantees correctness and efficiency (one important assumption is that the outline of each sensor wedge must be strictly convex). The scan table has a row for each row in the grid, and two columns to define the grid column interval that should be traversed (for each grid row). When a range sensor delivers a new range value, the occupancy grid implementation then traverses the grid cells that are directly below the sensor wedge, and asks the Bayesian sensor model for the probability odds (for the mid-point of the cell) and subsequently updates each grid cell.

The map updating frequency typically falls somewhere between 5 and 50 Hz depending on the grid and sensor configuration at hand. The grid is configured so the cell size is in the same order of magnitude as the smallest obstacle to be detected.

Sudden change in terrain elevation is a well known problem, where the ground may falsely appear as an obstacle. That is just one example of problems that are not easily solved in non-deterministic outdoor environments, when using a 2-dimensional grid. The above mentioned problem is partly solved by adjusting the maximum range of the sensors in such a way that the probability of getting a ground echo (within maximum range) is sufficiently small. Furthermore, some heuristic techniques may be used to determine if a ground echo has been received. Another similar problem is negative obstacles, like ditches and ravines, which is not yet embraced by the obstacle detection subsystem. Therefore, a 3-dimensional occupancy grid, with embedded terrain analysis, is considered a natural next step.

5 Path Tracking and Obstacle Avoidance

To navigate safely through the forest, a new path-tracking algorithm named *Follow-the-Past* has been developed. Traditional algorithms like Follow-the-Carrot [1] and Pure-Pursuit [2] use position information only, and sometimes run into problems that can be avoided by taking into account additional recorded information from a human driver. If the vehicle deviates from the recorded path, for example as a result of avoiding an obstacle, or because of noise in the positioning sensors, the Follow-the-Past algorithm steers like the driver, plus an additional angle, based on the distance to the path. The algorithm is described in the following section. More details and test results are found in [6, 7].

5.1 Follow-the-Past Algorithm

While manually driving along the path, the orientation and steering angles are recorded together with the position at every moment. The recorded position (x', y'), the recorded orientation θ' and the recorded steering angle ϕ' are used by three independent behaviors:

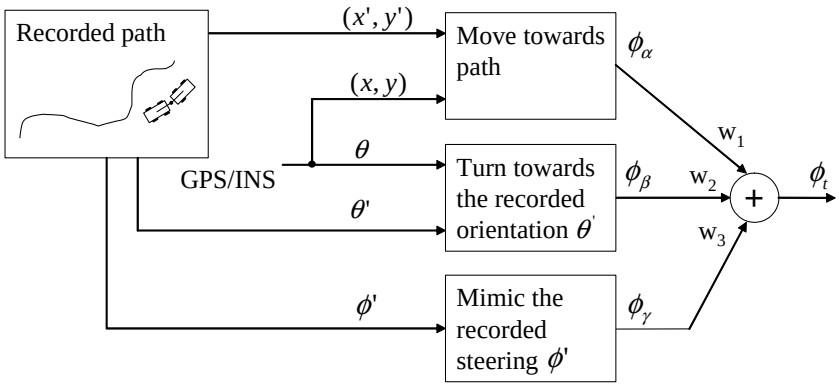


Figure 8: Path tracking with reactive control of steering angle ϕ_t.

ϕ_β: Turn towards the recorded orientation θ'
ϕ_γ: Mimic the recorded steering angle ϕ'
ϕ_α: Move towards the path

Each behavior suggests a steering angle and is reactive, i.e. operates on the current input values; orientation, steering angle, and shortest distance to the path. ϕ_α uses recorded positions (x', y') and actual position (x, y) as inputs. ϕ_β uses recorded orientation θ' and actual orientation θ as inputs. ϕ_γ uses the recorded steering angle ϕ' as input. The three behaviors are fused into one action, the commanded steering angle ϕ_t, as shown in Fig. 8. The three independent behaviors ϕ_α, ϕ_β, and ϕ_γ operate in the following fashion:

ϕ_β: Turn towards the recorded orientation The angle θ' is defined as the recorded orientation at the closest point on the recorded path. This point is called the path point. ϕ_β is computed as the difference between the current orientation θ and the recorded orientation θ':

$$\phi_\beta = \theta' - \theta. \tag{2}$$

ϕ_γ: Mimic the recorded steering angle This behavior simply returns the recorded steering angle ϕ' at the path point:

$$\phi_\gamma = \phi'. \tag{3}$$

By using the recorded steering angle, the curvature of the path is automatically included in the final steering command. This is a great advantage compared to methods like Pure-Pursuit [2] and Follow-the-Carrot [1].

ϕ_α: Move towards the path This behavior is responsible for bringing the vehicle back to the path, if the vehicle deviates for some reason from the path. Such a deviation can be caused by noise in the position signal, or by the obstacle-avoidance system. ϕ_α can be computed in many ways, e.g. by the following algorithm (refer to Figure 9:

1. Determine the closest point on the recorded path (denoted Path Point).
2. Compute a Look Ahead Point at a Look Ahead Distance ℓ from the Path Point, in a direction δ, defined as the sum of the recorded orientation θ' and the recorded steering angle ϕ' at the Path Point, i.e.:$\delta = \phi' + \theta'$.
3. Calculate a Look Ahead Angle ψ, defined as the polar angular coordinate for the vector between the vehicle's current coordinates and the Look Ahead Point.
4. Compute ϕ_α as the difference between ψ and the angle δ., i.e.: $\phi_\alpha = \psi - \delta$.

An alternative method to compute ϕ_α can be found in [6].

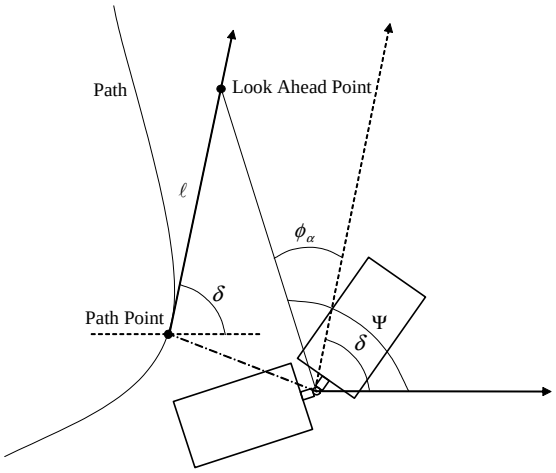


Figure 9: Calculation of ϕ_α, which is responsible for keeping the vehicle on the track.

5.1.1 Command Fusion

The three behaviors ϕ_α, ϕ_β, and ϕ_γ all return a suggested steering angle, aiming at fulfilling the goals of the respective behaviors. These three values are fused into one value ϕ_t by a weighted addition as shown in Fig. 8. In our tests, all weights have been set to 1. I.e.:

$$\phi_t = \phi_\beta + \phi_\gamma + \phi_\alpha. \tag{4}$$

The expression for the fused ϕ_t is:

$$\begin{aligned} \phi_t &= \psi - \delta + \phi_\beta + \phi_\gamma \\ &= \psi - (\phi' + \theta') + (\theta' - \theta) + \phi' \\ &= \psi - \theta. \end{aligned} \tag{5}$$

5.1.2 Testing and Results

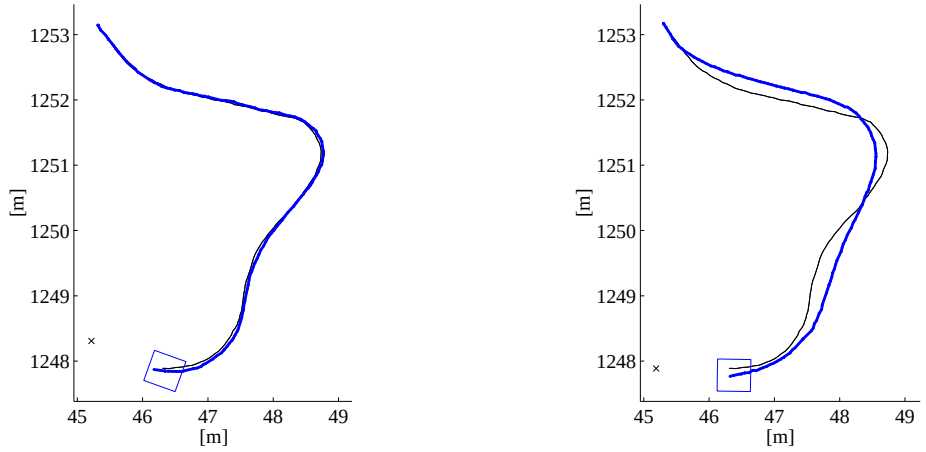
The developed algorithm has been tested both in a simulator for forest machines [13] and on a Pioneer robot. In this report we present only a test done with the Pioneer robot and compare the results to an implementation of the Pure-Pursuit [2] method. In this test, a Look-Ahead Distance $\ell = 1.2$ meter is used. Figure 10(a) shows results for path tracking with the Follow-the-Past method. The vehicle (thick line) is capable of following a recorded path (thin line) with almost no deviation from the path. As a reference, Figure 10(b) shows how the vehicle behaves when using the Pure-Pursuit method under the same conditions. This path has some sharp turns, which makes it difficult for both Follow-the-Carrot and Pure-Pursuit, as they tend to "cut corners" instead of following a highly curved path. Under normal conditions, this problem is avoided by the Follow-the-Past algorithm.

5.2 Obstacle Avoidance

To function in the forest machine application, the path-tracking behavior has been combined with VFH+ [16] for obstacle avoidance. The HIMM grid used in the original VFH+ algorithm is replaced by an occupancy grid with Bayesian updating as described in Section 4.2.

6 Status, Experiences and Future Work

The system has been successfully implemented on the simulator and on the Pioneer robot. The developed algorithm for path tracking performs very well, and the general tools for robot software architectures have been shown to be both powerful and flexible. Sensor tests and system adjustments for the forest machine are in progress and planned to be completed during 2005.



(a) Follow-the-Past (b) Pure-Pursuit

Figure 10: Comparison between the Follow-the-Past and the Pure-Pursuit path tracking algorithms. Follow-the-Past is able to follow the path almost perfectly, while Pure-Pursuit tends to "cut corners". The examples above are from tests with the Pioneer robot.

The continuation of the project will deal with the sensing problems specific for forest environments. Techniques to distinguish obstacles from ground in uneven and hilly environment have to be developed. The specific problem with detection of human beings is also of the highest priority for a future productification of an unmanned forest machine. The future work will also involve development of ways to achieve accurate localization and estimation of heading without expensive GPS and gyro equipment.

Acknowledgements

This work was financed by The Kempe Foundations, VINNOVA, Land Systems Hägglunds, Carl Tryggers stiftelse, LKAB and Komatsu Forest AB. We acknowledge their support gratefully.

References

[1] M. J. Barton. *Controller Development and Implementation for Path Planning and Following in an Autonomous Urban Vehicle*. Undergraduate thesis, University of Sydney, Nov. 2001.

[2] R. C. Coulter. Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 1992.

[3] U. Hallonborg. Förarlösa skogsmaskiner kan bli lönsamma. *Skogforsk RESULTAT*, (9), 2003.

[4] H. Have, S. Blackmore, B. Keller, F. Thielby, S. Fountas, and H. Nielsen. Autonomous weeders for christmas tree plantations - a feasibility study. Technical Report Pesticides Research no. 59, 221 Danish Environmental Protection Agency, 2002.

[5] T. Hellström. Autonomous navigation for forest machines. Technical Report UMINF 02.13, Department of Computing Science, University of Umeå aug 2002.

[6] T. Hellström and O. Ringdahl. Follow the past - a path tracking algorithm for autonomous forest vehicles. Technical Report UMINF 04.11, Department of Computing Science, University of Umeå 2004.

[7] T. Hellström and O. Ringdahl. Follow the past - a path tracking algorithm for autonomous vehicles. *Submitted for publication*, 2005.

[8] R. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, 82:35–45, 1960.

[9] H. Mäkelä and K. Koskinen. Navigation of outdoor mobile robots using dead reckoning and visually detected landmarks. In *ICAR'91 Fifth International Conference on Advanced Robotics*, pages 1151 – 1156, June 1991.

[10] H. P. Movarec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9:61–74, 1988.

[11] D. Novick. *Implementation of a Sensor Fusion-Based Obstacle-Detection Component for an Autonomous Outdoor Vehicle*. PhD thesis, University of Florida, 2002.

[12] P. Z. Peebles Jr. *Radar Principles*. John Wiley & Sons, 1998.

[13] O. Ringdahl. Path tracking and obstacle avoidance for forest machines. Master's Thesis UMNAD 454/03, Department of Computing Science, University of Umeå april 2003.

[14] A. Robotics. Robots, AGV's & robotic sensing. *http://www.activmedia.com/*, 27 Jan. 2005.

[15] A. T. Stentz, M. Ollis, S. Scheding, H. Herman, C. Fromme, J. Pedersen, T. Hegadorn, R. McCall, J. Bares, and R. Moore. Position measurement for automated mining machinery. In *Proceedings of the 1999 International Conference on Field and Service Robotics*, pages 299 – 304, August 1999.

[16] I. Ulrich and J. Borenstein. VFH+: Reliable obstacle avoidance for fast mobile robots. *IEEE Int. Conf. on Robotics and Automation*, pages 1572–1577, May 1998.

[17] K. Westlund and T. Hellström. Requirements and system design for a robot performing selective cleaning in young forest stands. *Submitted for publication*, 2005.