Robot Brothers EasyWheels and ReD in Field Robot Event 2009

Teemu Kemppainen, Teemu Koski, Jaakko Hirvelä, Johan Lillhannus, Tomi Turunen, Jussi Lehto, Ville Koivisto, Marko Niskanen and advisors Timo Oksanen, Jari Kostamo, Petro Tamminen
Helsinki University of Technology (TKK), Department of Automation and Systems Technology
Department of Mechanical Engineering
University of Helsinki, Department of Agrotechnology
contact: Otaniementie 17, 02150 Espoo, Finland. timo.oksanen@tkk.fi

Abstract

Two robots were built to Field Robot Event 2009. The first robot, EasyWheels, was designed to carry out all the tasks and the second one, ReD, was created mainly for freestyle purposes, but it was also supposed to take part in Task1. The robots were built by students from Helsinki University of Technology and University of Helsinki. EasyWheels won the second prize in the Field Robot Event 2009 and took all in all four prizes.

1. Introduction

In Field Robot Event 2009 the tasks were: 1) basic navigation between curved maize rows, 2) advanced navigation in straight maize rows with missing plants and with turnings based on program, 3) detecting weeds (green golf balls) and spraying those, and 4) freestyle where each team had to demonstrate what the robot could do in real world.

Figure 1: EasyWheels and ReD

EasyWheels (Figure 1) is a continuum from earlier Finnish robots that participated in the Field Robot Event (Honkanen et al 2005, Telama et al 2006, Maksimow et al 2007, Backman et al 2008). However, EasyWheels is completely redesigned and remade from scratch. No parts or software was used from the previous robots. Some of the EasyWheels' ideas are adopted from the last year's robot (Backman et al 2008) but this time the main idea in designing EasyWheels was to make it as modular as possible.

The novel properties in the robot are: modular structure, advanced suspension system, two way driving, symmetric design, embedded computing with real time operating system, 3D CAD based design and CAM based manufacturing.

2. EasyWheels

2.1. Mechanics

The mechanics of the robot can be divided into four different areas as displayed in Figure 2. The area inside the red ellipse is the plate where the majority of the electric and electronic components are located. The frame and our advanced suspension system can be seen inside the blue circle and inside the yellow circle is the axel module. The cover which was designed and manufactured by our sponsor Valtra can be seen inside the green circle.

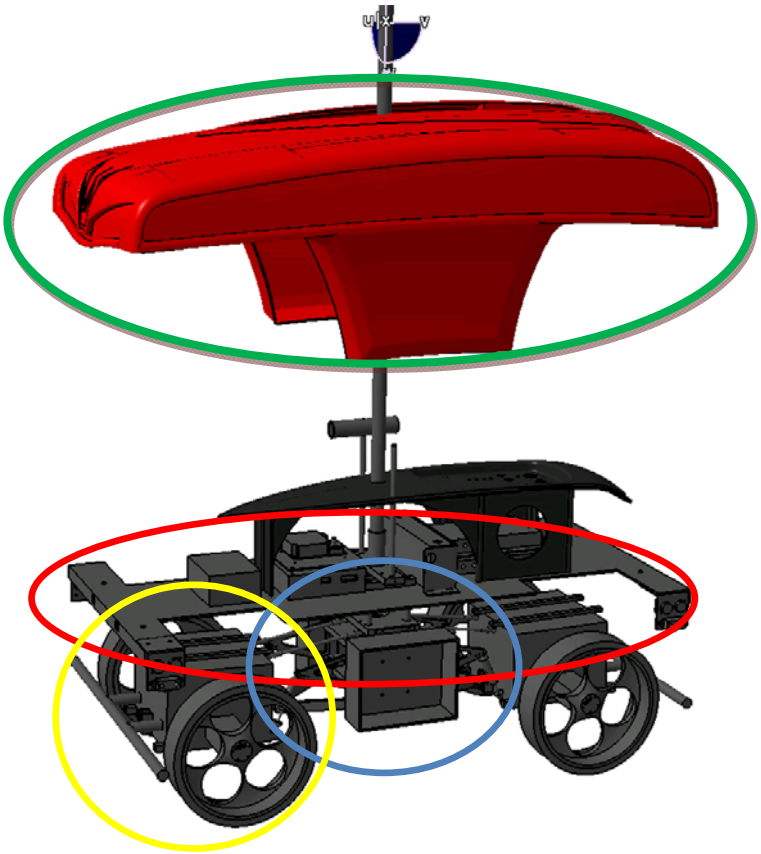

Figure 2: EasyWheels uncovered

2.1.1. Axel modules

The robot is designed and constructed of different modules. The modular design enables many ways for varying the construction and modular design could be adjusted for different tasks. This is an advantage in agriculture because the tasks depend on many variables and tasks vary depending on for example the crop and geological location. It is also an advantage that you can always have a spare module ready and in a case of a malfunction the module can easily and quickly be changed and the robot can continue its task while reparations are being done to the broken module.

The axel modules were designed and modelled using 3D-CAD software. All the components used in the axel module were modelled to ensure the parts would fit in a compact package. The model of the axel module is shown in Figure 3.

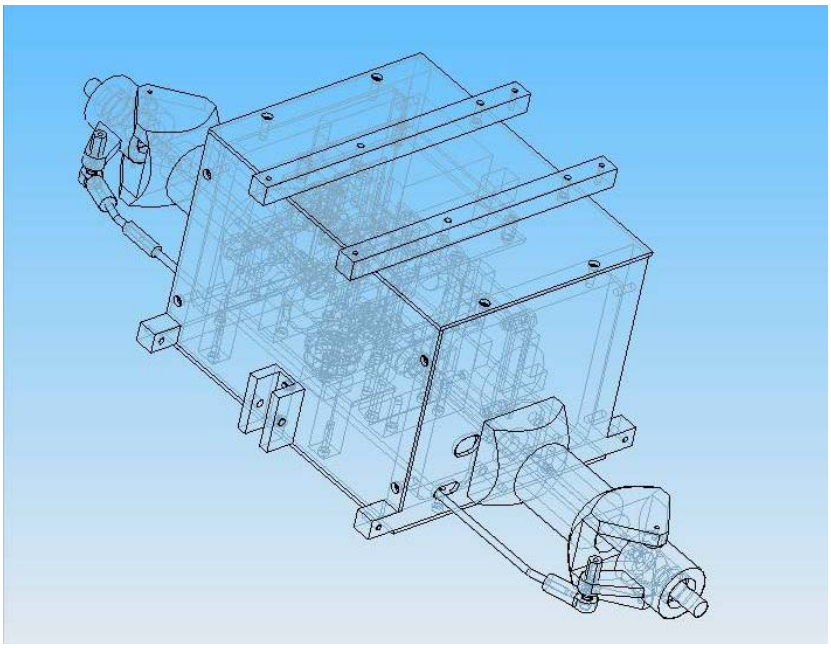

Figure 3: 3D CAD model of the axel module.

The main idea in the axel module was to include all the necessary parts needed for steering and driving the robot. The main components are: a) electric motor, b) reduction gears, c) differential, d) optical rotary encoder, e) steering servos including feedback, f) motor controller and g) microcontroller. These components were fitted into one box and the only interface to the module was a 12V power cable and a communication bus. The microcontroller in the module was designed to do the required feedback control inside the module. In the early stage different communication channels were considered but at the end RS-232 was chosen. It can be seen a picture of a complete axel module In Figure 4 and in Figure 5, the mechanically important parts are visualized and encircled.

Three identical axel modules were built: two axle modules were used in the robot and one was for spare (which proved to be a very good idea). As the team suffered from failures of the driving motors the spare module made it possible to replace the damaged module with a repaired one and the field tests could be continued with only short delays. While the field tests went on, the broken module was repaired by the other team members.

Figure 4: The axel module (with bumper bar attached)

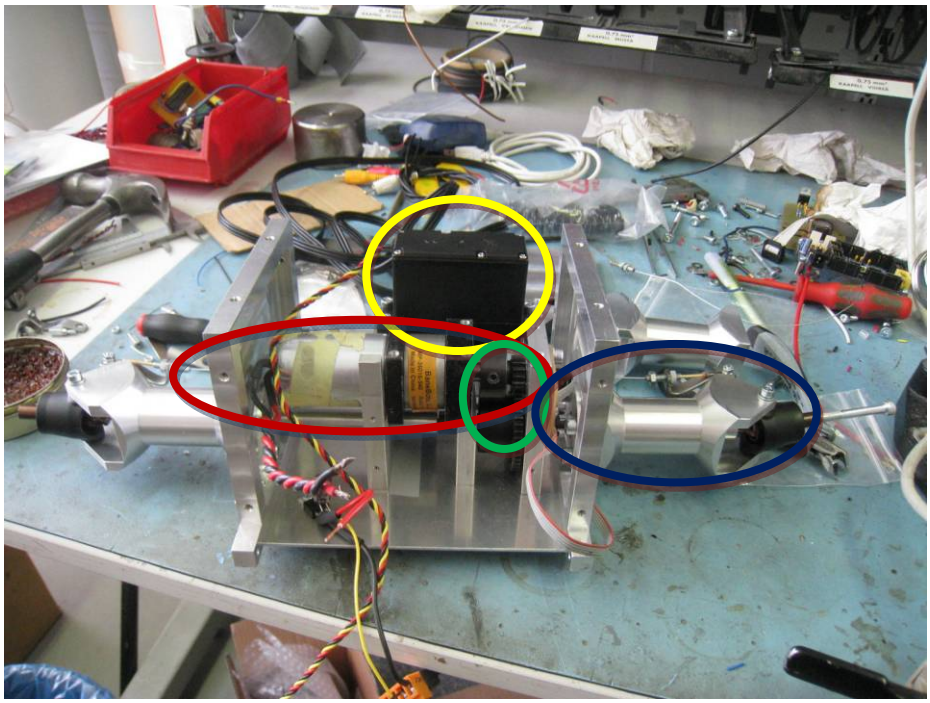

Figure 5: Main parts in axel module

In Figure 5, a Hitec HS-805 BB servo is shown in the yellow circle. The servo uses a wire system to turn the angle of the wheels. The wire starts from one steering block, circulates around the pulley of the servo and finally goes to the other steering block. The diameter of the pulley is optimised so that the full turning range of the servo is utilized and the servo delivers maximum torque to steering blocks. In order to improve the controllability of the steering angle, an external potentiometer was installed below the pulley wheel of the servo. The measurement of the steering angle was used to improve the precision of the steering angle and this was done by implementing a PI controller for the steering angle in the microcontroller. It was noticed that external measurement was required to compensate for

the steady state error of the selected HS-805BB+ servo and adding a PI controller improved the precision of steering significantly.

The red circle shows a Mabuchi RS-540 motor with a 16:1 transmission (Banebots) that deliver the torque to the wheels. Inside the green circle is the differential and inside the blue circle is the anchor and steering block.

The tires were cut from a Monstertruck 128/210-4.5 WhiteSpot tires and the wheels are CNC manufactured from a plastic bar. Most of the sheet metal parts seen in Figure 4 and Figure 5 were manufactured by our sponsor Laserle Oy and the remaining parts were manufactured at TKK. Laserle offered to cut all sheet metal parts for our robot according to our CAD drawings.

2.1.2. Frame and suspension system

The main idea for the design of the frame was adopted from the previous robot - 4M (Backman et al 2008). A four-wheel-driven and four-wheel-steered chassis was considered to be very functional as it was wanted to make sure the robot wouldn't get stuck on the field. The traction was improved by a balancing mechanism which ensured that each wheel applied the same contact force to the soil. In EasyWheels and 4M this was solved with a lever mechanism allowing the tires to adapt to the shapes of the surface. In addition to the balancing mechanism, a traditional suspension system was included in the frame of the robot. The 3D model of the chassis of the EasyWheels robot is shown in Figure 6.

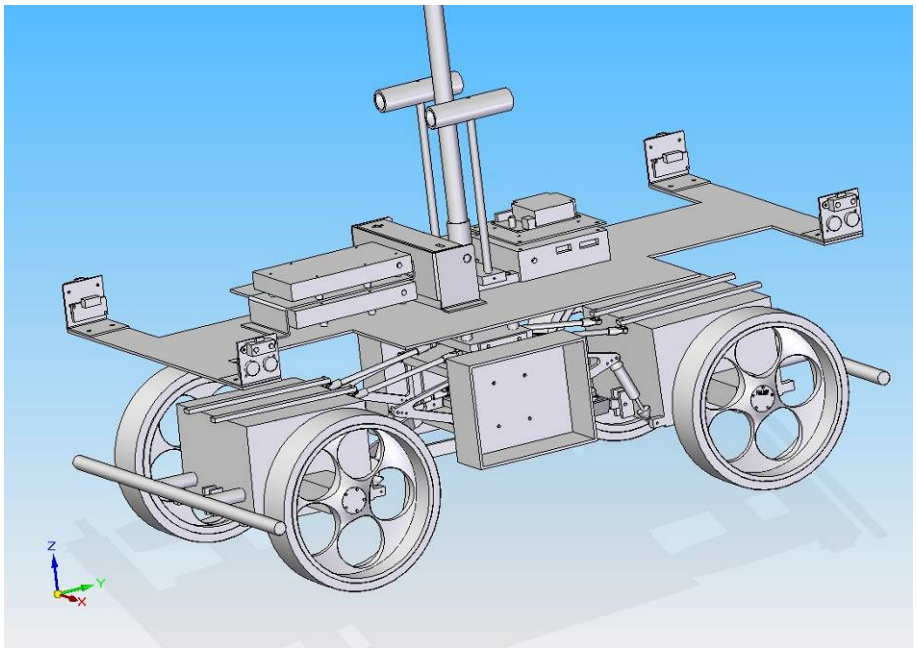

Figure 6: The chassis of the EasyWheels robot.

The idea was pretty much the same as last yeas but now that we had engines in the axel modules and not in the centre we needed to make some changes. One of them was to move the fulcrum from the top of the module to the side of the module. In that way it was also possible to relocate the part that controls the adapting in the lower frame (where 4M used to have its motor). The other reason was that EasyWheels was to be bigger than 4M.

The main idea was to connect all the shocks to a twisting plate that would rotate and guide all the four wheels to adjust the surface. Let's say that left front wheel would drop in a hole. Normally this would

mean that left front tire would have less load on it as well as the right rear tire. In a worse case scenario this would result to the robot being immobilized because the differential gear would direct all the thrust to the wheels that are in the air.

In our solution the idea is to even the load on the tires so that every one of them would also have the same thrust from the engines. If the left front tire of our robot would drop in a hole it would drop all the way to the bottom of the hole, because as the left front tire would be dropping, so would the right rear tire. At the same time the right front tire and left rear tire would rise towards the frame.

As it is bigger than its predecessor, it also was heavier even though a full sized laptop was not used onboard as was in 4M. That resulted in some trouble with the suspension. The original shocks that came with the shock absorber were obviously too light. They were designed for a 1.5 kg radio controlled car. With some searching and luck we did find suitable springs for the robot.

The main parts of the frame are milled or laser cut aluminium slabs and they are connected simply with screws. The most important rotating parts were attached with ball bearings to be sure that they operate as planned.

2.1.3. Plate

The idea was to have all the electronic systems (excluding those in axel modules) in one plate. The plate was designed to have enough room for systems. Despite of that requirement, at the end it was considered too small. One reason for that was that the plate had to be shrink down in order to make the cover to wrap also the batteries. Shrinking down included also some designed bends that was done to increase the plate's stiffness; though the bends were removed the stiffness of the assembly plate was still sufficient. The space required for cables was underestimated. After all modifications we had less space for electronics and wirings got messier than planned.

2.1.4. Cover

The cover for the robot was designed by Valtra and they have manufactured it using a 3D plastic printer. This was part of the sponsor agreement; the robot should not look like 80's video cassette recorder. Despite of technical limitations for designer, like free space in front of axel modules for tools, the design is very nice and functional for maintenance purposes. (Figure 7)

The cover was designed to be waterproof, it consists of two parts: one fixed part where control buttons are (in Figure 1, the black one), and the removable cover that allows easy access for maintenance. The local control buttons like for example start, stop, force turning and emergency stop were very handy when calibrations were made, because the robot did not always do what we expected it to do.

Figure 7: Cover design by Valtra.

2.2. Electronics

The electronics of EasyWheels can be split up into two areas: axel module electronics and plate electronics.

At the beginning it was decided to use only one kind of microcontrollers in this project. As the proto board used also in 4M was reported to be a good, the same model was used also in EasyWheels. The proto board is Futurlec ATMEGA which incorporates AVR ATmega128 microcontroller. One controller is inside every axel module and one is used to read sensors in the plate.

The software development for AVR was done using CodeVisionAVR by HP InfoTech.

2.2.1. Axel module

The electronics in the axel module are all related to the steering and driving of the wheels. The system is built up around an ATMega128 micro controller/ Futurlec ATMEGA proto board. The components that are connected to the board are the Hitec HS-805 BB servo explained in chapter 1.1 and the potentiometer that works as a feedback for its position. For motor feedback a Avago Technologies optical encoder (500 PPR) was used. With the info obtained from the encoder (motor rpm and direction) the motors speed is set with either a PWM value or rpm value that is obtained through a PI controller. PWM signal is directed to custom a build PWM amplifier, the amplifier is based on the same design as the one used in 4M. The PWM amplifier is shown in Figure 8.

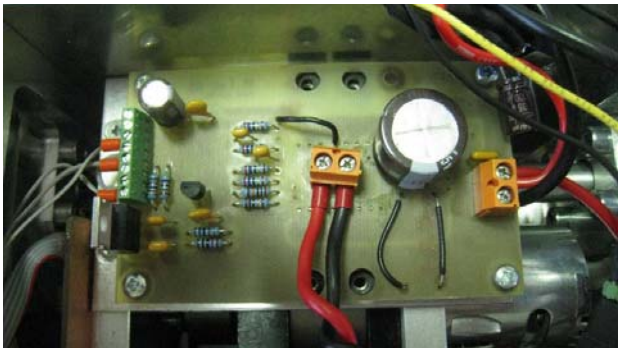

Figure 8: Motor PWM amplifier.

The controller is programmed so that it has 3 modes. If the module is in mode 0 no functions will work and the controllers are in standby. Mode 1 is for testing and as backup for normal drive. In mode 1 the servo and motor is given straight PWM values (open loop) so that it is easy to make tests. In mode 2 all the PI controllers start to work and this is the mode that the robot will have in the field. The microcontroller sends/receives information to/from the PC located at the plate.

2.2.2. Plate

The robot contains four ultra sonic sensors (SRF08) which are used for measuring distance from the robot to the corn plants in row. The measuring distance is set to about 60 cm and sampling time is 50ms. All the ultra sonic sensors are connected to I2c local network which is operated by Atmel 128 microcontroller. The ultrasonic sensors are programmed so that they first fire the front sensors and then read them one at the time and then the same procedure with the rear US sensors. Offset of 25ms between front and rear sensors and gain value optimisation is used to prevent false echoes. The same local net I2C contain also two 2D magnetometers that detects earth magnetic field in x and y directions. Sensors are installed in a 90 degree angle, so that measured data from the 2D sensors can be combined so that a 3D data can be obtained, it was cheaper to buy two 2D magnetometers than one 3D sensor.

The robot contains also four infrared distance sensors (SHARP GP2D12 4-20cm) which are located underneath the ultrasonic sensors and the practical measuring distance of the IR sensors is about 30 cm. The sensors are controlled by the same Atmel microcontroller that handles the ultrasonic's and magnetometers.

Due to tight time scale and lack of space on the robot we had to cancel the planned gyro sensors and the inclinometer. The gyros and the inclinometer was tested and controlled by Atmel microcontroller and connection was made via SPI ports.

Machine vision was made with Logitech webcam that was a little bit modified and re boxed. Because the robot needed to drive in both directions the webcam was installed on top of a 360 degrees turning servomotor. The night before the race we decided to chance servo motor because reliability problems with turning accuracy. On the Field Robot Event the servomotor was an ordinary 180 degrees servo and due to lack of test time it did not work as expected.

Computers, microcontrollers and other electronic devices are getting their electricity from a lead acid battery that is located on the assembly plate (12V / 2.1A). Also microcontrollers inside the axel modules are getting their electricity from this battery. The battery can run about one hour before the voltage drops below critical. Battery powers one main computer, two additional computers which runs image processing and machine vision, three Atmel 128 microprocessors and other gadgets like WLAN and sensors.

2.3. Machine vision system

Machine vision system consists of two different parts: the system that detects golf balls (weed) and the system that detects the center of the maize rows as well as the beginning and the end of the maize rows. Due to computing problems it was decided to have one processor for each part.

2.3.1. Row detection

This system is divided into adaptive RGB-color channel altering, simple color thresholding, detection of the centerline and into calculating the output.

The idea of the RGB-color channel altering is that because thresholding is based simply on color, we need an algorithm that would keep the RGB-colors the same in different daylight conditions. First the histogram for each color is calculated. Then the beginning and the end of each RGB-color channels are found so that we first start from intensity value 0 and going "upwards" until we have found 10% of the color channel that we have been looking for. The same idea is applied when finding the end of the color channel.

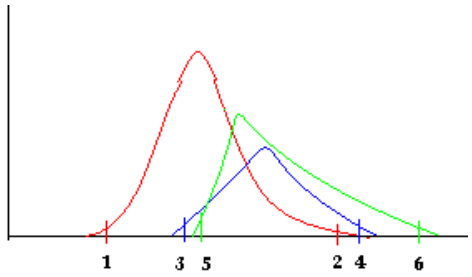

Figure 9: Histogram for each of the RGB-colors and their representative beginning and end points

In Figure 9 1 represents the beginning of the red-channel and 2 the end of the channel. Respectively 3 and 4 represent the blue-channel and 5 and 6 the green-channel. After this each of the color channels are expanded so that in a new RGB-color histogram each of the beginning points are 0 of intensity and end points 255. The formula for calculating a new intensity value for each pixel is (this formula needs to be calculated for each color):

$$I(x,y) = \frac{I(x0,y0) - Cmin}{Cmax - Cmin} * 255,$$

where I(x,y) is the new intensity value, I(x0,y0) the current intensity value, Cmax the end point of the current color channel and Cmin the beginning point of the current color channel.

After these calculations the RGB-color histogram should look something like this:

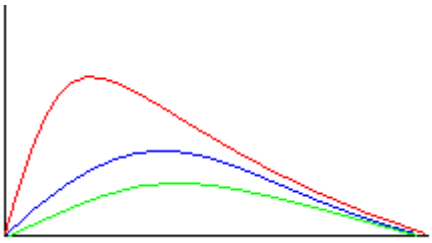

Figure 10: The histogram after RGB-color altering

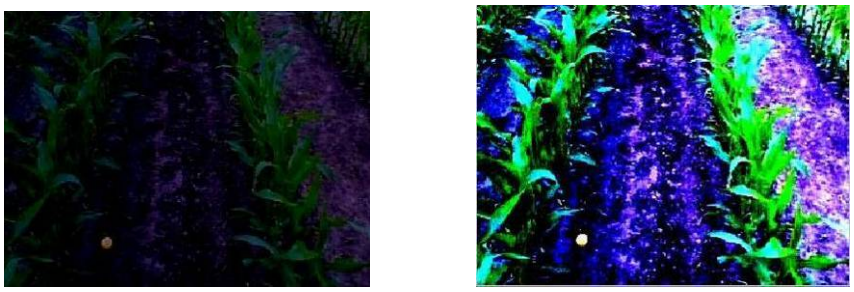

Figure 11: Original image (left), altered image (right)

However, this system was not used because it was too heavy.

The image is thresholded so that green vegetation can be seen from the thresholded image. This results in a black and white image, where white corresponds green colors and black other colors.

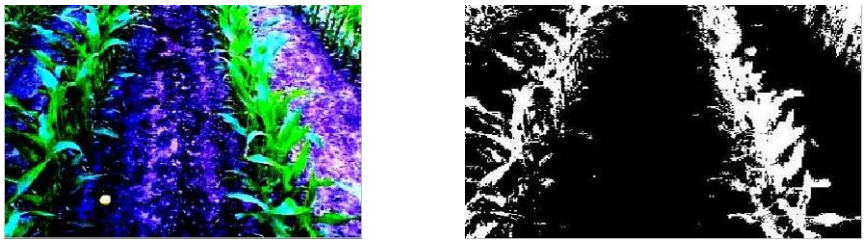

Figure 12: Altered image (left), thresholded image (right)

The used thresholding uses a following function:

$$f(x,y) = \begin{cases} 255, & if\ green\ channel > r + red\ channel\ and\ green\ channel > b + blue\ channel \\ 0, & else \end{cases}$$

where r and b are parameters which can be set.

After thresholding, the system tries locate the maize rows. The algorithm starts from the bottom center of the image and goes left and right until it has found enough white pixels in a 3x5 pixel region. Then a red circle is added to the resulting image. When it has found the maize rows from left and right it calculates the average of these x-coordinates and a new centerline point has been found. A blue circle represents this in the resulting image. When the whole image has been checked for maize rows, a straight line is plotted on the calculated centerpoints (green line). Blue line indicates the direction of the robot in Figure 13.

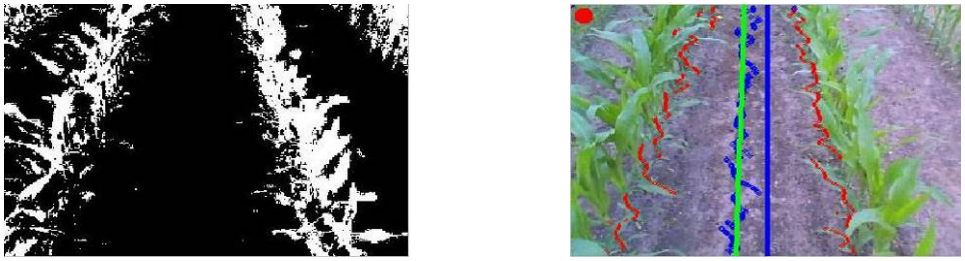

Figure 13: Thresholded image (left), resulting image (right)

Now the robot calculates the output using the robot's centerline and the calculated centerline from the analyzed image. Output consists of the distance between the robot's center and the calculated centerline and the angle (alfa) between these two lines.

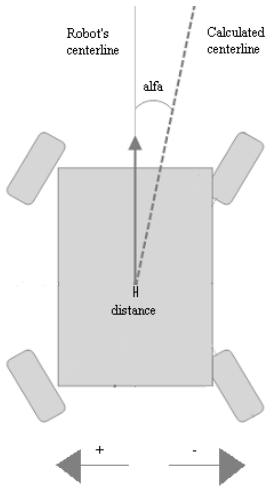

Figure 14: distance and alfa variables

Before the output can be calculated a few conversions have to be calculated. First the upper length (d1a) and lower length (d2a) of the camera image have to be calculated. This is done by:

$$d1x = 2 * \frac{4}{3} * \tan\frac{a}{2} * \sqrt{h^2 * (dix)^2}$$

where h is the height of the camera from the ground and dix the distance between the center of the robot and the camera's upper or lower view.

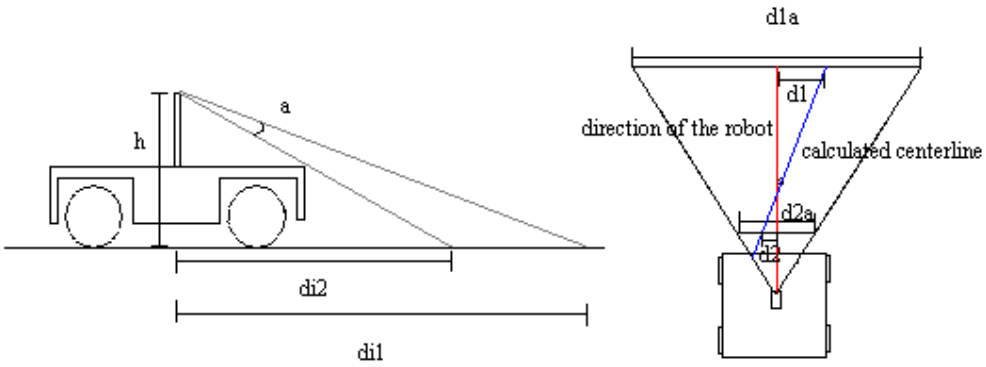

Figure 15

The alfa is then calculated as follows (radians):

$$alfa = \tan^{-1}\left(\frac{\frac{d1a * d1}{320} - \frac{d2a * d2}{320}}{d1 - d2}\right)$$

And the distance (meters):

$$distance = \frac{d2a * d2}{320} - dl2 * \tan^{-1}(alfa)$$

Now the detected centerline of corn row is in metric scale and ready for sensor fusion and navigation.

2.3.2. Golf ball detection

This system is divided into multiband thresholding and hough transform. The thresholding works by assigning for each of the RGB-colors a lower and upper threshold value. The resulted image is black and white image where white pixels correspond golfball green.

Figure 16: Original image (left), thresholded image (right)

But the resulted image has “false” white pixels which can be removed by averaging. This is done by checking a 3x3 pixel area and if there are enough white pixels the intensity of the middle pixel is left to 255. If the criteria is not met, the intensity is changed to 0.

Figure 17: Smoothed image

After this we use canny edge detection to find the edges of the thresholded image.

Figure 18: Canny edge detection applied to smoothed image

The next step is to use Hough transform to find objects that are circle of shape. Basic idea is that we draw a circle of radius r pixels in every white pixel in a completely new image. This circle just adds the intensity value by 1 in the new image.

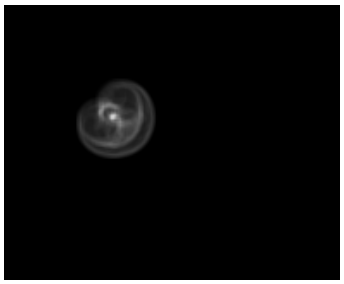

Figure 19: Hough transform applied to previous image

Next the algorithm scans the Hough transform picture and if a pixel has an intensity value greater than I (I can be set, but it was set to 200) the algorithm has found a ball and draws a circle in the resulting image.

Figure 20: Resulting image

After the system has found x times (x can be set, it was set to 3) a golf ball below a certain line, the system informs that the ball has now been passed. It gives 128 if the ball was on the left side of the image 1 if it was on the right 0 if there was no ball and 129 if there were balls on both side of the image.

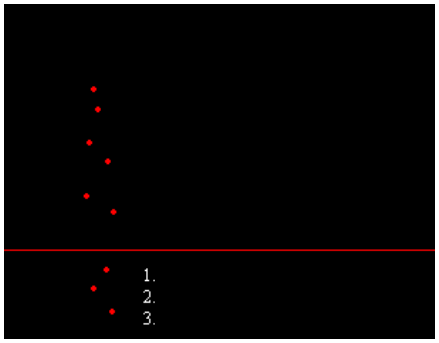

Figure 21: Image where the detected golfballs can be seen

2.4. Navigation

The navigation development started when the robot was still on the drawing board. Therefore a simulator was developed. The same ideas from previous robots (Backman et al 2008, Maksimow et al 2007, Telama et al 2006) was recirculated. The kinematic equations for 4 wheel steered robot were written and it was modelled using Simulink. All the distance sensor measurements (ultrasonic,

infrared) were simulated by using simple beam angle collision test by polygon algorithm. The image of camera on top of mast was created by modelling the field using Matlab Virtual Reality toolbox and the rendered image was captured from screen. Simulators' graphical output is shown in Figure 22.

Machine vision measurements were done by first creating a VRML file from field data by adding 1 plant to each maize location in data file. A picture was then made for each simulating step using its current location as input to Matlabs Virtual Reality toolbox. The picture from VRML block was then sent into MEX file, which is Matlabs way to run C-codes. This MEX file calls the machine vision codes that were the same as codes in real systems.

All the navigation algorithm are developed in Simulink and the controller part was converted to C-code using Simulink RTW. This was very useful, first the same algorithm can be developed, tested and tuned using simulator, and then the same algorithm can be run in the robots real time system.

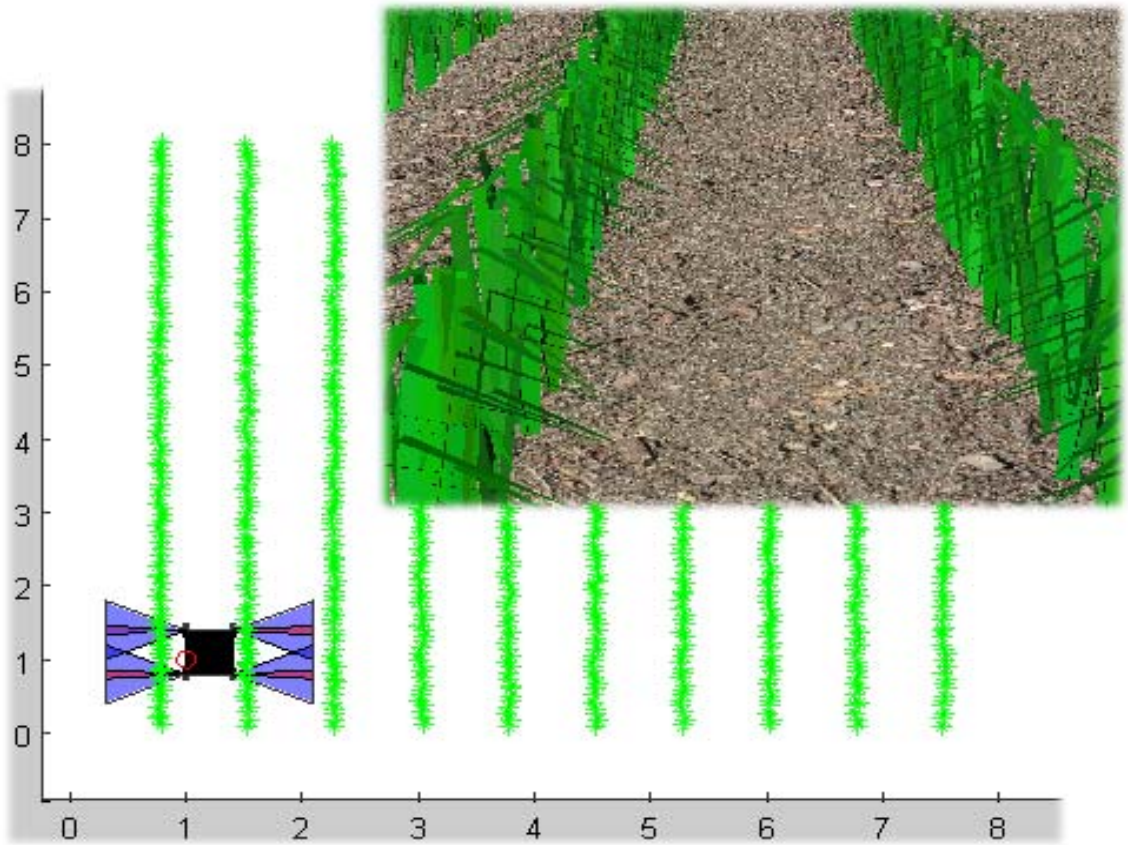

Figure 22: Simulator graphical outputs

The controller consists of blocks of which only one was active at the time and modes were used to control which block was in use at which time. Each task was divided into individual blocks such as normal driving in middle of row with row changes always to next row. For staying in the middle of the row, two different controllers were made: simple one and advanced navigation.

2.4.1. Row end operations

Row changing was implemented using Simulink Stateflow and two different state charts were done. The first for always changing to next row using "crab turns" (Figure 23) and the second for following

patterns which could skip rows when changing rows. Crab turn was defined by previous teams and it means that first the robot turns all wheels pointing to one angle (like a crab does), drives until travelled equal to half the row width, then stops, turns all the wheel to opposite angle and drives back without actually turning the robot.

For advanced pattern turns distances were used, which were calculated from control history. When state changes to out from row, the robot turns 90 degrees (which is measured by compass) after this the robot drives straight the distance that is calculated by width of row multiplied with the amount of turns need to be skipped, and reducing an amount of two times turning radius (equation 2.4.1). Also if turn amount is 1, "crab turn" is performed and if next row is same as earlier just the opposite direction, the robot drives out from row turns the camera around and then goes backward in the same row. Due to limited test time pattern turns couldn't be made working although pattern turning did work in simulator. During competition only "crab turns" worked well enough on the real system.

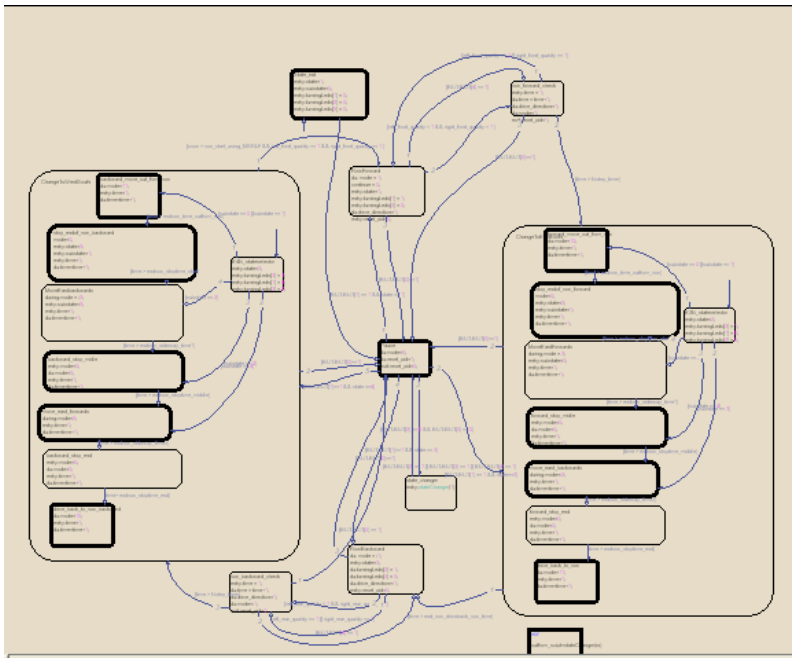

Figure 23: Simple row changing flow chart for doing "crap turn"

$$dist = (next\_row\_number \bullet row\_width) - 2 \bullet turning\_radius \qquad (2.4.1)$$

2.4.2. Advanced and simple navigation

Simple control strategy locked current rear wheels into middle position and uses front distance measurements difference to steer the front wheels by using a PID controller. After performing "crab turn" rear and front side are changed in scripts and the robot just locks its new rear wheels into middle position and start steering with its new front wheels using earlier rear ultrasound measurements which are now its front measurements.

Advanced mode consisted of weighted least square method that fits 1 line for latest 20 measurements from ultra sound-and infra red sensors for each side of robot. The locations of those measurements were placed into coordinates of its current location and calculating back to its previous location for each measurements using robots 20 latest controls. Outputs from this line fitting are one angle, distances to rows in robots left and right side and goodness number of current fit. These calculations

were then combined with machine vision outputs by using weights and goodness number. Result from these combination operations are difference from middle and angle difference from going straight. Combined angle- and distance difference were inputs of 2 PIDs, the first controls angle difference trying to set robot straight and the second one tries to keep the robot going middle of row. Both controllers operated as PI controllers because the derivative term made it unstable in simulations. Final wheel angles for front and rear axel modules were calculated using these controllers' outputs and simplified odometer model.

2.4.3. Real system implementation and log file simulator

Both control systems worked in simulator but due to lack of testing time advanced mode wasn't robust enough for using it in competition and only simplified control strategy was used. In both cases row ending was decided using two conditions: machine vision parameter that was "robot in row"; and 10 latest front side ultrasound measurements if all those were out of boundaries. Although there was a possibility of not using machine vision at all and only drive with measurements from ultra sound sensors, by only turning weight parameter to 0.

The system that was used in the real robot also included a front- and a rear block as shown in Figure 24 this changed the robots inputs which in most cases are pulses to engineering units and rear controllers turns controllers outputs back into corresponding pulse amounts. To be able to use this in the real robot it was generated into C-code by using Matlabs RTW C-code generator. In Matlab 2007b some problems with C-code generation occured. This caused lots of unnecessary problems such that it forgot some parameters from the C-code that were supposed to be changed in GUI and code optimization rounded some values without informing the user. Also Matlab 2007b seemed to have some sort of stability problems and crashed every now and then.

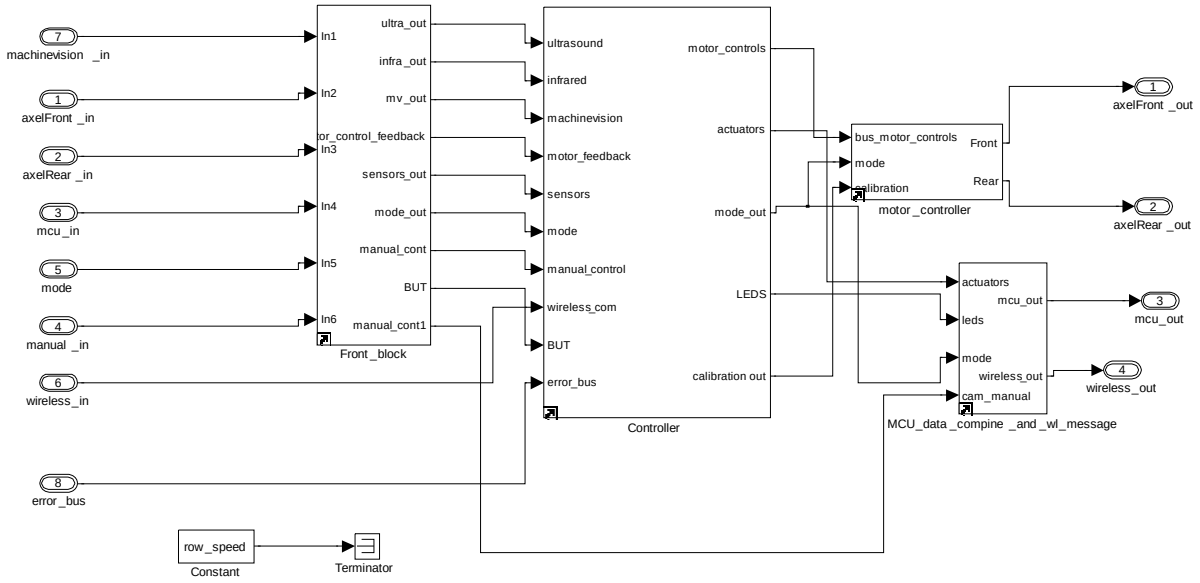

Figure 24: Simulink model of real robot control system, front and rear blocks

The simulator that was used to calculate controller states from log files was done in a hurry and it had some performance problems which occurred when the log file size increased and requirements for physical memory increased significantly in long log files.

In the controller part many things could be been done better like not using multiple copies of the same element in different task modes by only doing rearrangement of blocks and their tasks. Also benefits of making libraries were understood by the controllers creator too late to take full benefits from these.

2.5. Computing system

In contrary to earlier robots, EasyWheels uses embedded distributed computing instead of an on-board laptop. Earlier robots couldn't reach strict real time, because of the non-real time operating system, Windows XP. EasyWheels' computers use Microsoft Windows CE 6.0, which is a real time operating system. With non-real time operating systems the problem usually was non-deterministic operating system's background processes that could freeze or interrupt critical navigation and machine vision algorithms. With Windows CE this kind of behaviour should not happen. In addition to real time operating system EasyWheels' computers are embedded. The goal was to achieve a modular computing platform with strict real time capabilities.

2.5.1. Hardware

EasyWheels has three embedded computers all running Windows CE 6.0. Two of the computers are for machine vision and one is for navigation. Machine vision computers are Toradex Colibri PXA320 embedded computers on Toradex Protea carrier board (Figure 25). The navigation computer is ICOP's eBox-4300 (Figure 26).

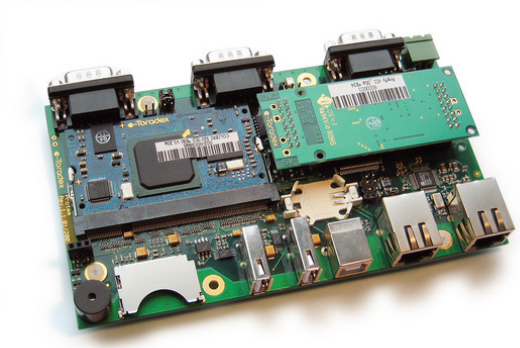

Figure 25: Toradex Protea carrier board with Colibri module (Toradex)

Figure 26: Ebox 4300 (EmbeddedPC.NET)

Colibri PXA320 has a very low power consumption with a great amount of computing power. Colibri has an ARM processor running at 806 MHz, but no floating-point unit (FPU). It was selected to be used for machine vision because of its computing power, the FPU was considered as an acceptable lack, as machine vision doesn't necessarily need floating-point computing as RGB images consists of 8bit integers. Naturally floating-point operations are possible, but through emulation they are very slow. One Colibri was used for row detection and the other for golf ball detection. As a navigation computer eBox is very efficient. EBox uses conventional x86 PC hardware and therefore has a FPU. With VIA Eden ULV 500 MHz processor eBox can easily run complex navigation algorithms which

can't be run on a Colibri. The three computers are networked via Ethernet, using a WLAN router. With networked design none of the computers (or algorithms) can freeze or slow down overall computing.

Of course one of the criteria was also a small size and both computers achieves this. Protea carrier board's dimensions are 86.5 mm x 117.4 mm with around 15 mm thickness. Ebox's dimensions are 115 mm x 115 mm x 35 mm.

2.5.2. Software

Modular design was also a goal with the software development. All code is self written in C++ except for the Matlab Real-Time Workshop generated navigation algorithms.

Machine vision software is designed to be as modular as possible. Even a less experienced developer can create a usable machine vision algorithm with the design. The machine vision itself is separated into several blocks, each computing a small part of the overall algorithms work, for example one block could be colour conversion, one threshold and so on. The algorithm is a "script" in a C++ -header so that the machine vision developer only needs to create necessary blocks and connect them in the script. Blocks are naturally designed to be reusable and as generic as possible.

Navigation software uses the same modular idea that machine vision, but it has no script or blocks. The machine vision algorithm itself is Matlab Simulink RTW generated code with C interface. Modularity is achieved so that navigation software is independent from the generated code. Only a small part of the interface needs to be rewritten if the algorithm changes drastically. If none of the inputs, outputs or tunable parameters change as the algorithm changes, no editing is needed for the software.

All of the software provides server and client interfaces. Every computer can be used and tested alone or together (Figure 27). Computers talk to each other via Ethernet, remote user interface is via WLAN.

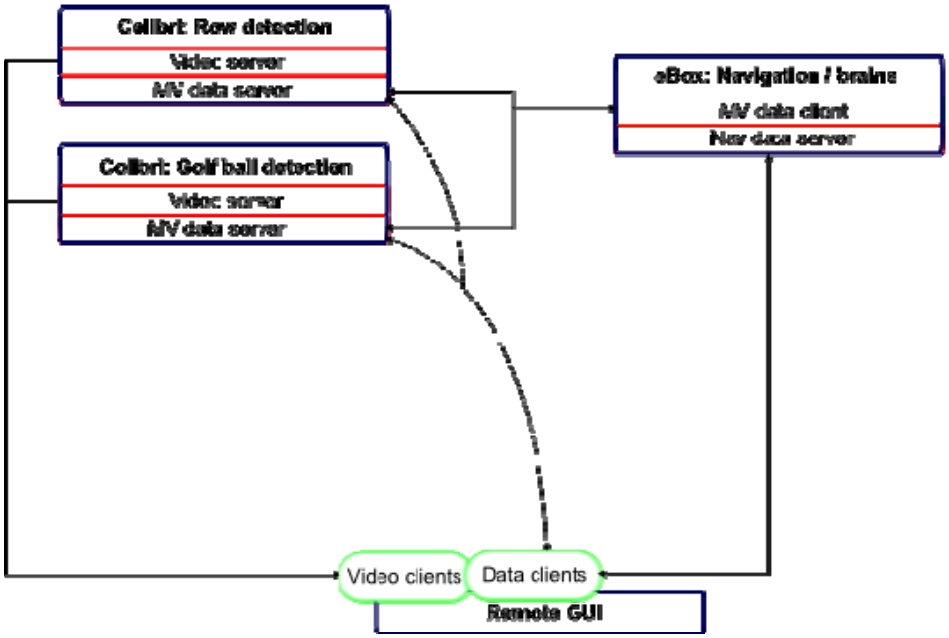

Figure 27: Software interfaces

Even though the interface is quite simple and easy to use for the algorithm developer, a huge amount of lower level C++ code had to be written. Most of the unexpected extra work was due to distributed computing and Windows CE. As none of the previous robots used distributed nor embedded computing, no one had any previous experience on the area. Distribution of computing power itself was expected to cause some extra work, but no one could predict the amount of headache Windows CE caused. Many parts of the code had to be rewritten several times as more and more “features” of Windows CE was revealed. Some of the features could have been avoided with previous experience with Windows CE. The biggest avoidable problem with CE was the lack of RTTI (Run-Time Type Information). This effectively prevented us from using Boost C++ libraries at all. The original idea was to use Boost.Asio with networking. After that deficiency MFC (Microsoft Foundation Class) was considered to be the choice for networking, but it was so buggy that it didn’t work correctly on CE. Finally all networking code was written with low-level Winsock 1.1. This forced us also to write all the serialization code from scratch.

It turned out that Windows CE lacks all the advanced features (available e.g. in Win32 environment) that enable the developer to write sophisticated code. Poor documentation of Microsoft’s code leaded to finding out all the features the hard way.

For machine vision OpenCV main libraries was ported to Windows CE, only some minor changes had to be done. This was considered a major breakthrough to use it in Colibri. Unfortunately it was learned that OpenCV utilizes floating point computing and typing internally even if the image type was integer. The result was that it was practically impossible to use any advanced OpenCV functions in real time computing, as the code had to be optimized all around.

3. ReD

As there was a considerable difference in knowledge of robotics, electronics and software within the group in the beginning of this robot project, it was decided that the students at the University of Helsinki should get acquainted with basic robotics first. The requirement was that a monster truck (Tamiya TXT-1) should be converted to a simple field robot being able to do the basic task of the competition indoors (driving between rows and making turnings to the next row). The idea was to make this with only two ultrasonic rangers, one steering servo, one speed feedback sensor and a AVR microcontroller.

The goal was to build the basic robot during autumn semester and the co-operation with other robot and "the tool" during spring semester. "The tool" had to be some real application in fields, and for the freestyle it was decided that two robots do interactive mechanical weed control, and this inspired the name of ReD (Remote Destroyer).

3.1. Mechanics

ReD is based on Tamiya TXT-1 RC-car chassis (Figure 28). The chassis has advanced suspension for both axles. Transmission consists of two Graupner Speed 500 motors mounted parallel in main gearbox, where also power is divaded for two cardan shafts, one for each axle. Front and rear axles are identical. Axles have differential gears. ReD has continuous 4-wheel drive, and as both axles are identical, it has also 4-wheel steering. In row it uses only front axle steering, but in headland it steers with both axles to improve turning.

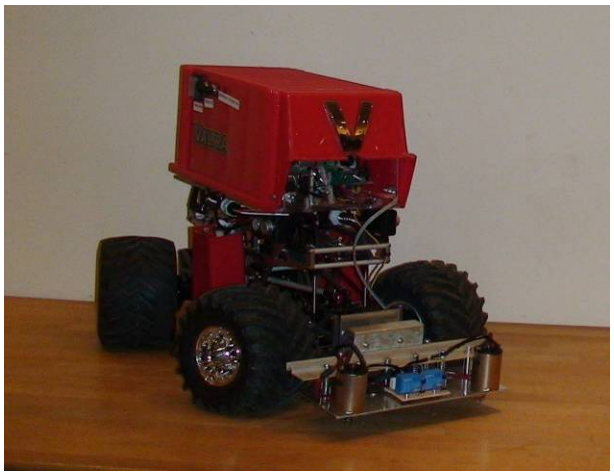

Figure 28: ReD

3.2. Electronics

ReD uses three voltage levels, +12V, +6V and +5V, which all share the common negative ground. Two 6V NiMh batteries are run in series to deliver the +12V main power to the “brains” of ReD, Futurlec ATMEGA microcontroller. +12V main power is also used by the self built DC-motor speed controller unit which controls the two Graupner Speed 500 DC-motors, CMUCAM3 machine vision camera and by the +5V regulator circuit on the self made power distributor board. Regulated +5V from this self made board is used by the Xbee Series 2 wireless serial communication device which is used for wireless serial communication between the ReD and EasyWheels. +5V output voltage directly from the ATMega AVR128 is used by two SRF04 ultra sonic sensors which are used for the navigation in the row, Honeywell hall-sensor which are used for odometry and for measuring the drive speed and also by CMPS03 digital compass which is used for turn angle detection at the end of the row. Two Acoms AS18-MG steering servos utilize +6V input voltage which is taken separately from one of the main batteries.

User interface includes three power switches: the main power switch and two separate switches for the AtMega 128AVR and CMUCAM3. User interface also has two push-buttons, one for compass calibration and one for switching the turn direction at the end of the row. Five LEDs on a custom made circuit board are also included in the user interface as indicators for different events. Two red LEDs show the current turn direction setting to be used at the end of the row, a green led indicates the power-on for the ATMega AVR128, one yellow led indicates the initiation of the starting sequence and one red led is reserved for error situations.

3.3. Machine vision

Machine vision system in the ReD is used to measure the distance between the ReD and EasyWheels robot, in freestyle. This distance information is used to calculate when the weed destroying unit has to be switched on and off. When information from the EasyWheels arrives about the location of the weed it has detected in the row, the distance between the ReD and EasyWheels is marked and based on to that distance information the weed destroying unit is switched on and off at appropriate time.

The base for the machine vision system is a CMUCAM3 camera unit which comes with the ready algorithms for image color detection. Ready algorithms are mainly used for image color detection, only slight modifications are made. The algorithm which is used divides the image to rows with the height of

1px and then processes each row separately while trying to find the specified color. After all the rows have been processed the information gathered during the process is assembled and information where the specified color in the image exists is shown. Machine vision system is utilized so that ReD is able to follow the EasyWheels robot and calculate the distance between the two robots. Basically the algorithm detects two red light sources (LED spot lights on the EasyWheels) from the image. After the detection the distance between the two objects is calculated. When this distance information is placed on to the pre-calculated formula, the distance between the two light sources and the camera can be calculated.

3.4. Navigation in row

Navigation in row is based on two SRF04 ultrasonic rangers. Measurements over 500 mm are filtered away to ensure that reflection from wrong rows won't be measured (e.g. if there are plants missing). If both measurements are usable, effective inter row spacing is calculated and recorded for further use. If only the other measurement is usable, hypothetical location of the other row is calculated based on recorded effective inter row spacing. This row estimation, as we call it, enables robot to follow only one wall. Default inter row spacing can be set in program. Distance between middle point of robot and middle point of the row is calculated either from both measurements (if both are usable), or from one measurement and calculated location of the other row. This distance is fed into the PID-controller that calculates new PWM-signal for steering servo. If both measurements are not usable during 500 mm driven distance, the robot starts headland turning. This distance can also be set in program.

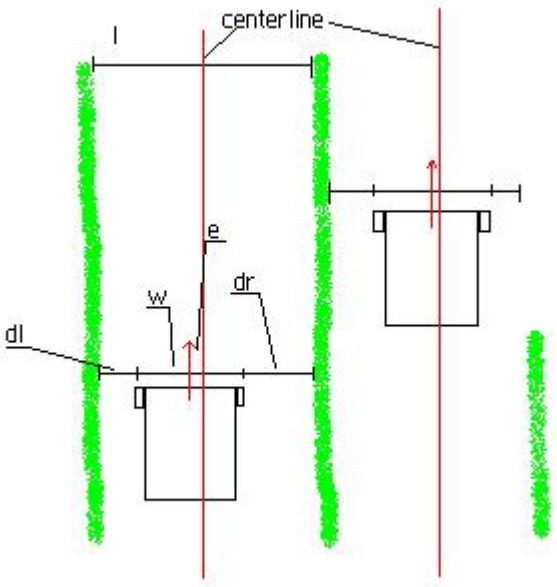

Figure 29: Navigation in row

In Figure 29 l = inter row spacing, w = distance between ultrasonic rangers, dl = left distance to row, dr = right distance to row, e = error (distance between centerlines of row and robot).

Error is calculated as follows: e = dl – dr
and missing measurement: dr = l – w – dl

3.5. Computing system

Programming is done with FlowCode flowchart programming environment (Figure 30), which generates C-code. All routines mentioned in this chapter are done in FlowCode macros (macros are generated as functions in C-code). Macros are used as much as possible, e.g. PID-controller, row estimations and headland turnings are in their own macros. This makes flowchart very easy to read. Interrupt routines are the main core of the program. Timer interrupts are used for pulse generation for servos and motor speed controller and one timer interrupt is used to measure approximately 16 ms time period to schedule trigging of ultrasonic rangers and data transmitting to EasyWheels. External interrupts are caused by hall-sensor, ultrasonic rangers and a push button for switching next headland turning direction. Data receiving causes USART-interrupt when the camera or EasyWheels sends data to ReD. Only very short routines are executed in interrupt macros. Longer routines are flagged to be executed in main loop right after interrupt. Some routines couldn't be done with FlowCode's flowchart, but it's possible to add C-codes into flowchart.

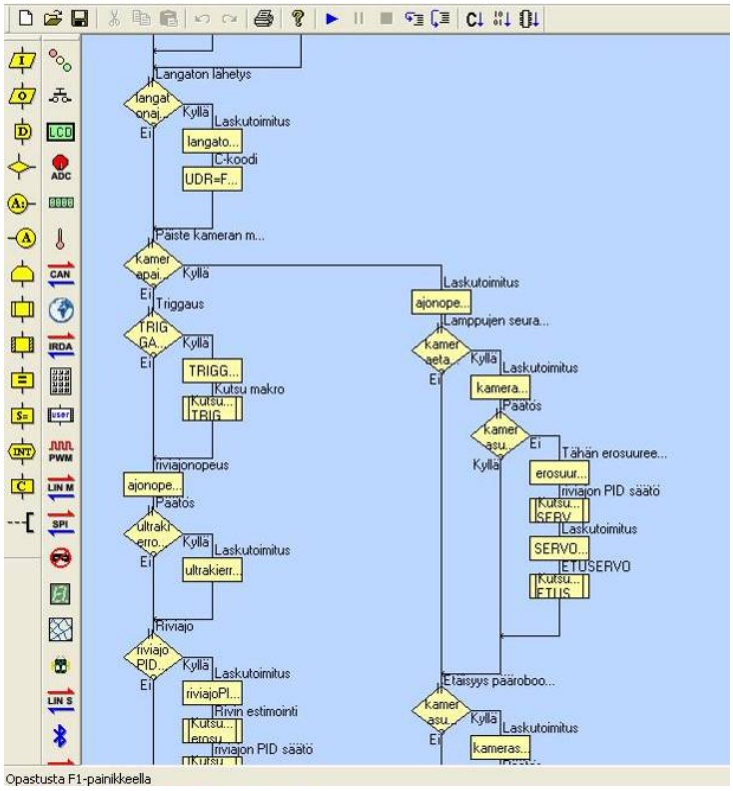

Figure 30: FlowCode

3.6. Destroyer unit

Destroying unit consists of simple chassis and two Mabuchi 540 motors. When EasyWheels tells ReD, that there is a weed in left or in right, camera is used to measure distance between ReD and EasyWheels. ReD measures distance to place where weed was found with hall-sensor. When weed is reached, ReD starts up its destroying motor in left or right depending on which side weed is found. Motors are controlled with transistors and relays.

4. Freestyle

In freestyle task, both EasyWheels and ReD would have worked together. Unfortunately ReD had problems with electronics, and this freestyle had to be skipped. The following will explain how it would have worked.

4.1. Idea

The idea of freestyle was to destroy weeds between rows. Shortly, EasyWheels was meant to detect weeds (golfballs), and tell ReD where to turn on destroying its (Figure 31). It was meant to use machine vision as a virtual drawbar so that ReD would have followed EasyWheels right behind. Data transmission would have been wireless. Machine vision and data transmission as well as destroying itself are described in chapters below.

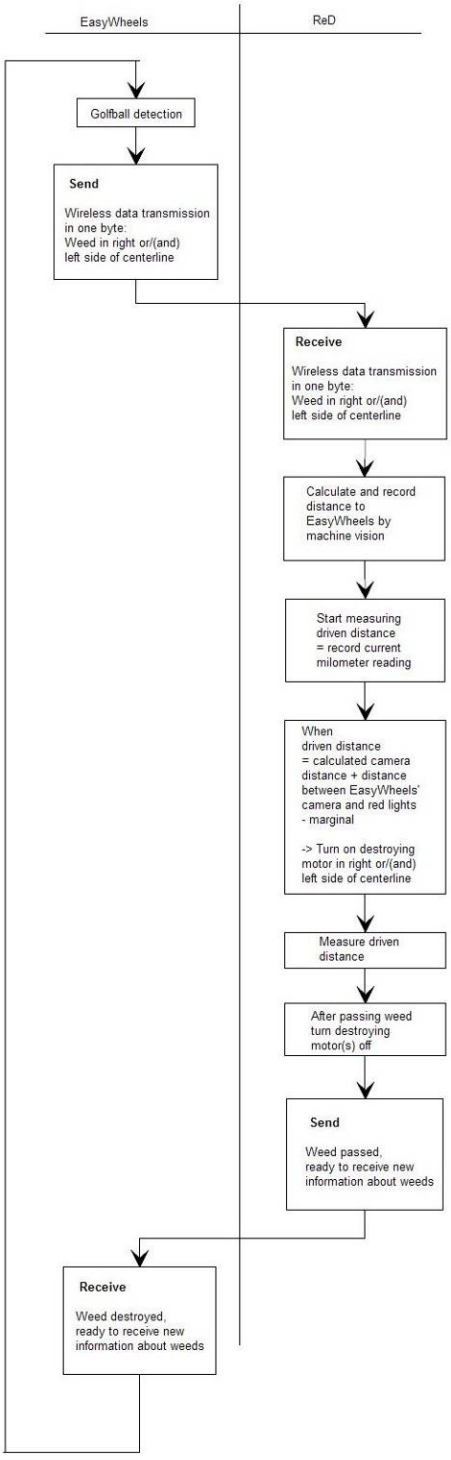

Figure 31: Simplified flowchart of planned freestyle

4.2. Virtual drawbar

Machine vision would have been used to make ReD follow EasyWheels. The idea of the virtual drawbar is simple: The rear end of EasyWheels was equipped with two red lights. In the front of ReD is a camera which would have been used to detect those two lights. As the distance between the two lights was known, the distance from ReD's camera to EasyWheels rear lights could have been calculated, as those distances are approximately proportional (Figure 31).

4.3. Destroying

When ReD would have been told by EasyWheels, that there was weed in either right or left or both side of centerline, ReD would have taken care of destroying incident during the rest of time. At the moment when information about weed would have reached ReD, it would have calculated distance from EasyWheels based on machine vision as described above, and started counting driven distance based on the hall-sensor. When driven distance would have been equal to calculated distance between ReD and EasyWheels, ReD would have turned on destroying motor in the side of centerline, where weed was detected (Figure 31). These distances and marginals, like distance from EasyWheels' weed detection camera to back lights, and marginal needed to turn on and off destroying units before and after weeds, were not measured.

5. Discussion

It turned out that Colibri's are not the right choice for machine vision computers as they lack a FPU. The missing FPU had to be compensated for some parts by creating algorithms for presenting the lack in mind. For some image processing operations however that was not possible. Colibri’s computing power didn’t seem to be sufficient for complex image processing. As a navigation computer eBox was a good choice. eBox could run all the navigation algorithms and other secondary processing (networking, parameter management etc.) without any problem.

Windows CE came out only passable. Lots and lots of unnecessary work was caused by Windows CE. Many of its features are poorly documented and many things that a C++ programmer could expect for an operating system to have, were missing. Windows CE provides support for .NET Compact Framework C# version only, no .NET C++. With C# many things probably would have worked better, but no one in the team had previous experience in developing software in C#. Generally .NET as a real time language is a little questionable, although this argument has no proven basis. One choice would be to use a Linux based operating system which come in various flavours, also in real time versions.

There was a lot to do to build the robot and make it to work. The project had repeating problems to keep in schedule. The chassis and axel modules were completed not earlier than 1.5 weeks before leaving to competition and there was too little time to test the robot algorithms on field. Without the simulator it may not have worked at all.

6. Conclusions

Even though EasyWheels used a novel design, it didn’t perform as well as planned. The design is excellent, but due to lack of work and testing time only a small part of the robots potential was used. Modular design definitely was a great improvement to the previous robots; every part (mechanical, electrical or software) could be separately tested.

The amount of work to create this kind of robot is huge. To create a robot like EasyWheels it definitely needs motivated members. It is not possible to master all the areas so it is a must to have people who design the mechanics and people who program and develop algorithms. A year for 6 member team would be barely enough to build EasyWheels, we had 6 members only for 5 months and 4 members for 5 months. EasyWheels was built entirely afresh, but with following robots that is not necessarily reasonable. Nearly every part of the lower level C++ code is reusable with minor modifications, so it would be wise to benefit from that.

Acknowledgments

Building of the robots and participation to the competition in Netherlands was possible thanks to our sponsors: Valtra, Koneviesti, Laserle, HP Finland, HP InfoTech and Suomen Kulttuurirahasto.

References

Backman, J., Hyyti, H., Kalmari, J., Kinnari, J., Hakala, A., Poutiainen, V., Tamminen, P., Väätäinen, H., Oksanen, T., Kostamo, J. and Tiusanen, J. 2008. 4M – Mean Maize Maze Machine. Proceedings of the 6th Field Robot Event 2008. ISBN 978-3-00-027341-4. pp. 9-41. http://www.fieldrobot.nl/downloads/Proceedings_FRE2008.pdf

Honkanen, M., Kannas, K., Suna, H., Syvänne, J., Oksanen, T., Gröhn, H., Hakojärvi, M., Kyrö, A., Selinheimo, M., Säteri, J. and Tiusanen, J. 2005. The development of an Autonomous Robot for Outdoor Conditions. Proceedings of the 3rd Field Robot Event 2005. ISBN 90-6754-969-X. pp. 73-90. http://www.fieldrobot.nl/downloads/Proceedings_FRE2005.pdf

Maksimow, T., Hölttä, J., Junkkala, J., Koskela, P., Lämsä, E.J., Posio, M., Oksanen, T. and Tiusanen, J. 2007. Wheels of Corntune. Proceedings of the 5th Field Robot Event 2005. pp. 75-87. http://www.fieldrobot.nl/downloads/Proceedings_FRE2007.pdf

Telama, M., Turtiainen, J., Viinanen, P., Kostamo, J., Mussalo, V., Virtanen, T., Oksanen, T. and Tiusanen, J. 2006. DEMETER – Autonomous Field Robot. Proceedings of the 4th Field Robot Event 2006, ISBN 978-90-8585-480-7. pp. 27-38.
http://www.fieldrobot.nl/downloads/Proceedings_FRE2006.pdf

BaneBots Robot Parts, http://www.banebots.com

CMUcam3, http://www.cmucam.org/

EmbeddedPC.NET, http://www.embeddedpc.net

FlowCode, http://www.matrixmultimedia.com/flowcode.php

Futurlec, http://www.futurlec.com

ICOP, http://www.icoptech.com

Laserle Oy, http://www.laserle.fi

MSDN, Microsoft Developer Network, http://msdn.microsoft.com

OpenCV, Open Source Computer Vision Library, http://sourceforge.net/projects/opencvlibrary/

Toradex, http://www.toradex.com