

# EasyWheels 2010 Robot in Field Robot Event 2010

Timo Oksanen, Jari Kostamo

*Aalto University, School of Science and Technology, Helsinki/Espoo, Finland.*

*Otaniementie 17, FIN-00076 Aalto, Finland. email: timo.oksanen@tkk.fi*



## Abstract

EasyWheels 2010 is an improved version of EasyWheels robot that took part into Field Robot Event 2009 in the Netherlands. EasyWheels 2010 robot is capable of driving between corn rows, doing the turns and capable of following another robot in constant distance. The robot is pretty symmetric and it is able to drive the rows in both directions, and therefore the turnings to a next row are quick to do. The robot uses ultrasonic and infrared rangefinders in each corner of the robot, turning camera head on top of the mast and tilt-compensated compass. The algorithms estimate position in a row using both ultrasonic and infrared rangefinders, backwards odometry and machine vision together. In Field Robot Event 2010 EasyWheels 2010 took part into the basic tasks, but also with Helios robot in two robot co-operative challenge. In two robots co-operative task, Helios was driving in front and “shooting harvest backwards” and EasyWheels 2010 was equipped with a hopper that collected the harvest – this demonstration was awarded with the first prize.

**Keywords:** *field robots, agriculture, co-operative robotics, field robot event*

## 1. Introduction

EasyWheels robot participated in Field Robot Event 2009 in Wageningen, the Netherlands. The robot was built by students from Helsinki University of Technology (nowadays part of new Aalto University) and from University of Helsinki. The robot won the second prize in the event 2009.

However, the robot suffered some technological problems, like bad turning servo for camera head and painful problems in programming C both in Windows and Windows CE cross-connection. The latter

was considered to be too bad experience to be given any further and therefore some crucial change was needed, to give a bit better (programming) tool chain for next student team (that was to be Turtle Beetle). The problems in EasyWheels were reported in the Proceedings of 7<sup>th</sup> Field Robot Event 2009.

Pretty soon after 2009 competition we as teachers wanted to investigate possible fixes for the tool chain, as well as some other technological drawbacks, like DC motor drivers. The underlying tool chain both in EasyWheels and later on, has been Matlab + Simulink + code generation + Visual Studio + Windows CE, with remote debugging. To find proper fixes for the tool chain, we wanted to test it first with EasyWheels, by starting from the scratch (in software) and developing most of that with C#, not with C/C++. Only Simulink generated code requires some wrapping. This technology was utilized in Windows in “Wheels of Corntune” (Maksimow et al 2007) and victorious “Mean Maize Maze Machine – 4M” (Backman et al 2008). However, this was not possible to do in the same way in Windows CE Embedded. Also real-time capabilities of C# over Windows CE was a bit mystery. The main motivation was to *learn* how Windows CE + Simulink + CF.NET (C#) can be used together and to *teach* it for the student team.

So EasyWheels 2010 was born, and after discussions with FREDT / Helios team, we ended up developing a two-robot demonstration for Field Robot Event 2010 special co-operative challenge.



Figure 1: EasyWheels 2010

## 2. Mechanics and Mechatronics

### 2.1. Axle Module

For locomotion EasyWheels is equipped with two identical “axle modules” that encapsulate DC motor, reduction gears, differential, optical encoder in motor shaft, steering servo, feedback steering sensor (potentiometer), DC motor driver and microcontroller that does lower level servo control and offers a simple control interface for computer with RS-232 serial connection. Beside serial port the axle module requires only 12V power supply. Three identical axle modules were constructed for EasyWheels. These modules have proven out to be reliable, as only the DC motors have been changed to more powerful after last event. No further fixes were needed. (Kempainen et al 2009).

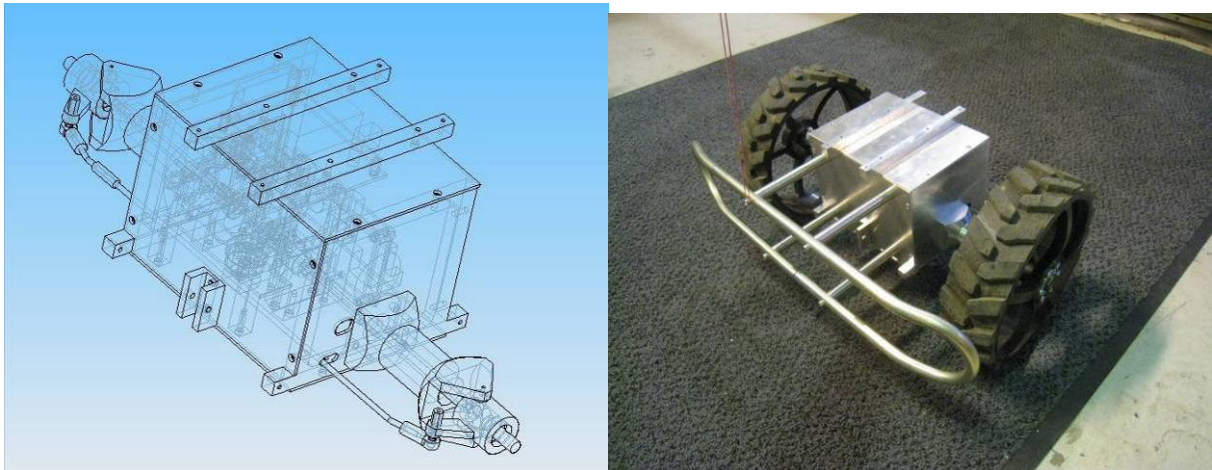


Figure 2: Axle module (Kempainen et al 2009)

### 2.2. Suspension

In EasyWheels suspension combined spring-damper with force balancing mechanism was used. The similar system was used in Mean Maize Maze Machine - 4M (Backman et al 2008), and in EasyWheels a bit more compact way. The suspension works very well in driving uneven terrain, but still the problem is to get enough roll stability to frame. This is still the problem in the chassis, as the upper body tends to roll on turns.

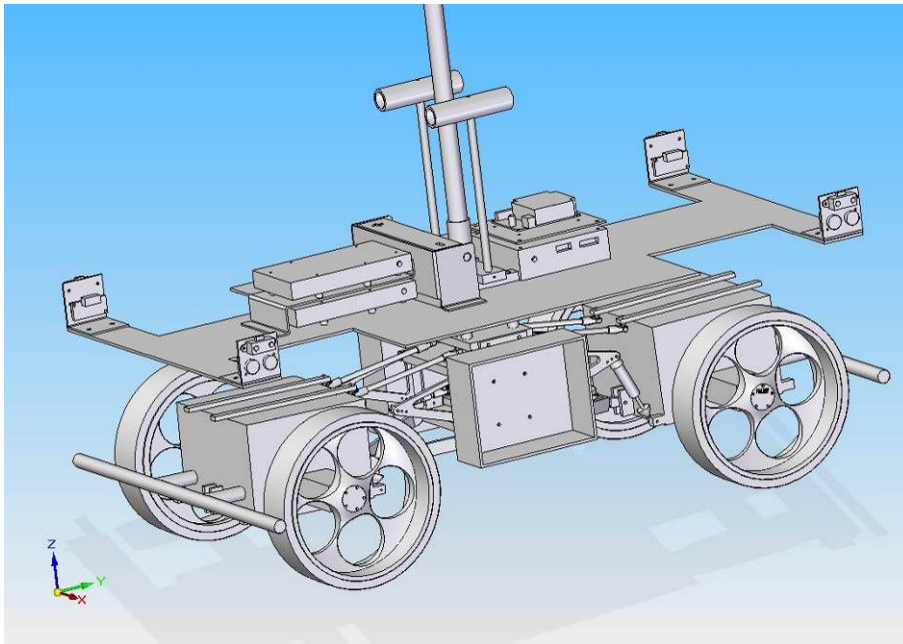


Figure 3: Suspension (Kemppainen et al 2009)

### 2.3. Camera head

One of the problems in original EasyWheels was the camera turning head. 360 degree operation was required for usage in two-way driving and headland operation. RC car servo with 360 degree rotation was selected. One of a few models capable of turning 360 degrees is GWS S125. However, the repeatability of this servo was rather bad (5 degree error) and therefore ordinary 180 degree servo was used in the competition 2009. (Kemppainen et al 2009)

For refurbished EasyWheels 2010 the RC servo was replaced with a stepper motor. A stepper motor is a bit heavier, but for this turning purpose good enough solution as there is no other counterforce than camera cables and vibration. A SparkFun stepper motor was selected for the purpose. The key numbers of the selected stepper motor are: step angle 1.8 degrees, 2 phase, rated voltage 12V, rated current 0.33A, holding torque 2.3kg\*cm. The stepper motor was installed on top of mast, in the place where servo used to be, and the new parts to connect the camera to motor were made with milling machine.



Figure 4: On the left: the stepper motor. On the right: the camera head with re-encapsulated Logitech 5000 with a sun glass.

### 3. Electrics and Electronics

#### 3.1. System

The electric system of the robot has completely been refurbished. One of the reasons was that the original electric wiring system, grounding plates etc. was a mess and hard to maintain. All the wires were taken out, components repositioned, some components added (like beeper, more ultrasonic range sensor to front, turning servo for front ultras, stepper motor for camera head, LCD display to local user interface, some new user interface buttons... ). Some new PCBs were milled to distribute power and to act as a star junction point for grounding (see Figure 5). I<sup>2</sup>C is utilized more and more to connect devices to microcontroller: LCD is working over I<sup>2</sup>C and all user interface LED's and buttons also through GPIO expanders.

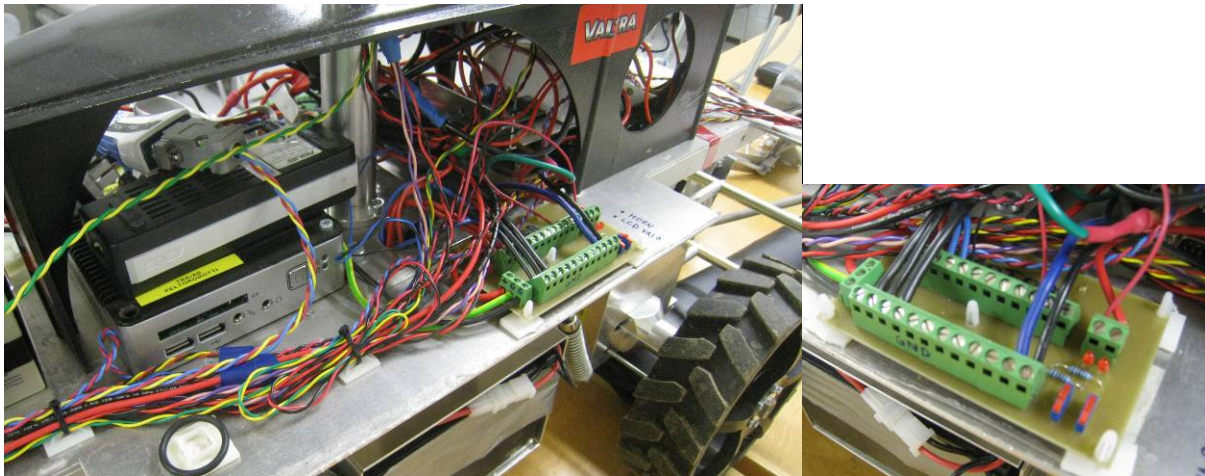


Figure 5: Wirings of the robot, one of the two junction boards on the right.

EasyWheels 2010 uses two parallel 12V lead acid battery systems to carry energy. One of the battery systems (2 times 6V/7Ah batteries in series) delivers power to DC motors and servos inside axle modules, and the other system (12V/2.1Ah + 12V/4.5Ah in parallel) supplies power to computers etc. The battery systems are split in two to eliminate risk of computer & controller reset during high current peaks when accelerating motors. The other battery was installed in parallel with an old 2.1Ah computer battery, as in FRE2009 the robot suffered battery problems. This resulted more weight, but was important for doing tests.

#### 3.2. Microcontrollers

Futurlec Atmega128 prototype boards were used in axle modules to do servo control for drive and steering, and the third one was used in upper body to connect all the sensors, stepper motor, front servo and horn to the computer. Five SRF08 ultrasonic ranglers, a SRF235 ranger, two CMPS3 compass modules, an LCD display (Batron LCD with Philips PCF2119 driver), and buttons and LEDs by using GPIO expander (Microchip MCP23008) in local user interface are all connected with I<sup>2</sup>C bus (Figure 6).

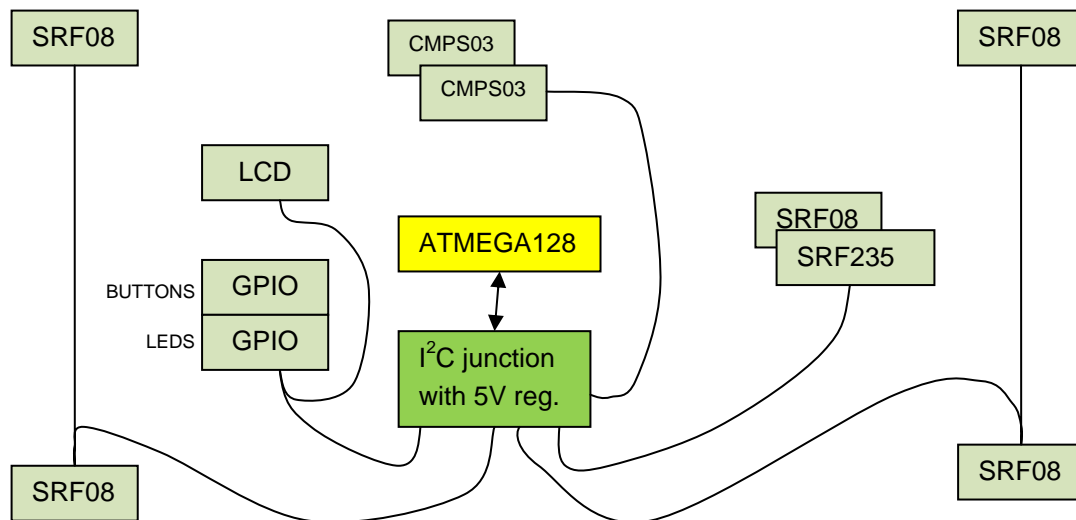


Figure 6: Devices connected with I<sup>2</sup>C bus

### 3.3. Stepper motor control

The stepper motor could be controlled directly from the microcontroller by using only some transistors between them. However, specific controllers are available that provide benefits like current control and microstepping to reach better control accuracy. A SparkFun EasyDriver v4 was selected for this purpose, Figure 7. The key features are: A3967 microstepping driver chip, resolution from full steps to 1/8 steps, adjustable current control from 150mA/phase to 750mA/phase and power supply from 7V to 30V.

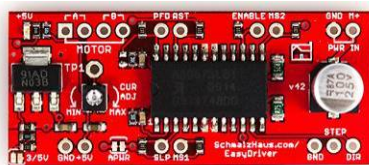


Figure 7: EasyDriver v4 – the stepper motor driver

The stepper motor control lacks feedback, so it is not possible to detect if the motor drops steps. In the power up sequence the motor is placed by hand to the zero position. This worked well in the short field test runs and also in the competition, but for longer working hours some feedback should be added.

### 3.4. Computers

The original EasyWheels was equipped with three Windows CE 6.0 running embedded computers, but in EasyWheels 2010 only two were used. The third one was used to detect weeds, and that was not objective this time. The other embedded computer is x86 miniPC and the other is XScale based.

EasyWheels 2010 uses embedded distributed computing instead of an on-board laptop. Earlier robots before EasyWheels couldn't reach strict real time, because of the non-real time operating system, Windows XP. EasyWheels' computers are running Microsoft Windows CE 6.0, which is a real time operating system. With a 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. The drawback of Windows CE is lack of flexibility, for instance relating new hardware added to computer – the compilation of operating system works only with the hardware for which it has been compiled, and if some required drivers are not compiled in, it may require recompiling the whole operating system. However, in the robot, only some USB-RS232 adapters and specific camera are needed to connect besides normal hardware, so this is not a repeating problem, only when setting up the system or changing the hardware.

Based on tests, eBox-4300 is about 4 times quicker to compute floating points (single) than Colibri. However, on the other hand Colibri is about 4 times quicker than eBox when it comes to 32-bit integers. For this reason eBox-4300 is used in navigational computation with Simulink generated code and Colibri used to compute machine vision algorithms that are integer optimized.

### 3.4.1. ICOP eBox-4300



Figure 8: Ebox 4300 (EmbeddedPC.NET)

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. (Kemppainen et al 2009)

ICOP eBox-4300 requires an external 5V supply. The original 230V adapter is rated to 3A, even if the computer does not take more than 10W. To produce 5V @ 3A from 12V lead-acid battery system, a DC/DC converter was installed (Texas Instruments TPS5430). The DC/DC converter was shared also to WLAN access point.

### 3.4.2. Toradex Colibri

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 it doesn't have a floating-point unit (FPU). It was selected for machine vision because of its computing power, the lack of a FPU was considered acceptable, as machine vision doesn't necessarily need floating-point computing as RGB images consist of 8bit integers. Naturally floating-point operations are possible, but through emulation they are very slow. (Kemppainen et al 2009)

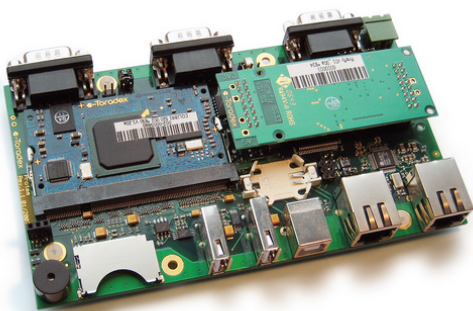


Figure 9: Toradex Protea carrier board with Colibri module (Toradex)

## 4. Sensors

### 4.1. Rangers

#### 4.1.1. Ultrasonic

The ordinary sensors in low budget robots are ultrasonic rangers. In this robot two kind of ultrasonic sensors were used: 5 pieces of Devantech SRF08 which operate in 40kHz sound frequency and one Devantech SRF235 which has 235kHz frequency. Both of the sensors are used over I<sup>2</sup>C bus. SRF08 is cheaper and has quite a wide beam, as opposite higher frequency produces narrow beam, but this sensor is more expensive. Four of the SRF08 were used in each corner of the robot, and the other two sensors were installed in front, in the middle of a bumper on top of a RC servo, and these sensors were used to measuring distance to the other robot running ahead. As it was not sure which of the sensors 40kHz or 235kHz is better for detecting the co-operative robot, both of them were installed.

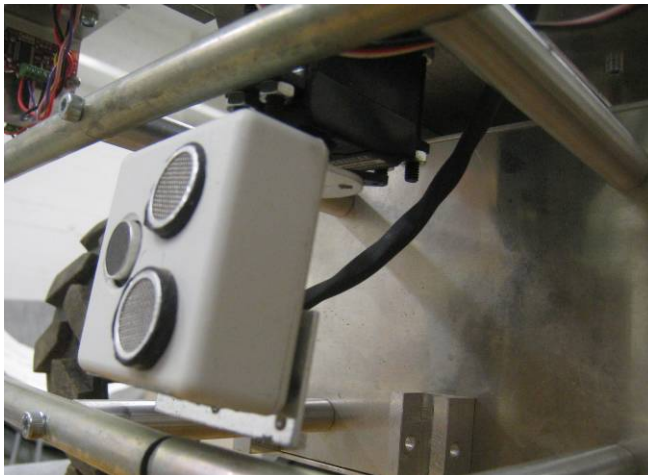


Figure 10: SRF08 and SRF235 installed in front of robot with servo head.

#### 4.1.2. Infrared

Four Sharp GP2D120 sensors were used in each corner to measure the distance to the maize row. The noise in these sensors is higher compared to SRF08 when driving in a maize row, but by using sensor fusion, these sensors still have additional information. The claimed range of sensors is 4cm to 20cm, but the sensor works quite OK from 3cm to 30cm. Under 3cm the signal is similar than in valid range, so under 3cm measurements should be restricted mechanically. Over 30cm range the signal is decreasing outside of valid range and therefore usable, but the sensitivity is low.

### 4.2. Heading

The heading of the robot is estimated by using both magnetic field sensors and inclinometer. With an inclinometer the attitude (roll & pitch) are measured, and this information is used to rotate magnetic field vector to horizontal level. The heading angle is computed from rotated vector x- and y-components by using arcus tangent.

#### **4.2.1. Compass modules used as magnetic field sensor**

Like in the original EasyWheels, two CMPS03 compass modules were used. However, the directly computed heading angle from the sensor is not used at all, but only the raw values that are readable from “internal test registers”, in 8-9 and 10-11, respectively. The registers contain raw measurements from the two Philips KMZ51 one axis magnetic field sensors. So practically CMPS03 was used only “as a demo board” for KMZ51 chips. From two CMPS03 modules together four raw measurements are recorded every 50ms.

#### **4.2.2. Inclinometer**

Two VTI SCA610 sensors (1-axis) were used to measuring roll and pitch. The maximum slope is 30 degrees. The inclinometer works well in static conditions, but not that well in dynamic movement as there is no gyroscope to stabilize the angle. However, an internal 2Hz filtering provides stable signal and with slow driving speed the signal is good enough for magnetic field attitude compensation purposes.

#### **4.3. Camera**

Logitech QuickCam 5000 was used. The camera was re-encapsulated to a new box already in the original EasyWheels. The camera has problems in outdoors, in summer sunshine the pixels may saturate, and the colours are twisted. To correct this, a sunglass added (see Figure 4). Actually, this was the other “sunglass” from the cheap sunglasses bought as a part for 4M (Backman et al 2008).

### **5. Software**

#### **5.1. The main tool chain**

As described above, one of the main motivations was to learn a proper tool chain to connect Matlab, Simulink, Stateflow, Windows Embedded CE and .NET framework. Compared with EasyWheels (Kemppainen et al 2009), the main difference is to use C# to program runtime code for Windows Embedded CE. Runtime includes interoperation with all the sensors, doing communication with remote user interface, handling parameter storing and handling log files. .NET framework and C# provides more powerful tools for writing equally working code both for Windows Embedded and regular Windows. In C/C++ programming the biggest problem in cross-platform development are a number of very small hidden differences in bit level (like in WinSock) that causes a lot of headache. .NET framework resolves many of these problems. For mobile devices and Windows CE Microsoft provides a smaller version of “.NET framework” which is known as “.NET Compact Framework”. To make a difference, the Windows version is hereafter called “.NET Full Framework”. The basic namespaces, classes and methods in Compact Framework are just the same as in the Full Framework, but all the advanced features are not included. Also in some classes some methods may not be available. Generally it can be said, that if C# code works in Compact Framework, it will work also in Full Framework, and this has to be a way of development, to guarantee cross-platform usage.

The first problem was to find how to use C++ code and C# code together in .NET Compact Framework. C++ code comes from Simulink code generation (Real Time Workshop, Embedded Coder), and in order to use it efficiently with C# programming, a Multilanguage programming has to be used for integration. For Full framework this is straight forward as it supports so called “managed C++” code, run on .NET framework. This technology was utilized both in Wheels of Corntune (Maksimow et

al 2007) and 4M (Backman et al 2008). However, Windows Embedded CE and .NET Compact Framework is not supporting running “managed C++”, only regular C++ without .NET. Due this reason, for the original EasyWheels (Kemppainen et al 2009) the decision was to solely use C++ in programming – which was not a good solution as was concluded by Kemppainen et al (2009).

Luckily, a way to overcome the problem partially exists, it is known Platform Invoke (usually shortened as P/Invoke), which is a technology of .NET to make calls to native libraries (Windows internal libraries, or developed by a user). To make calls, a definition of library function has to be written in C# with some definite syntax, to make a connection. After definition, the function is called like any other function. Not to be too simple, .NET Compact Framework is a bit more limited in a sense of supporting the data types. Only simple basic data types (like integers and floating points) are supported, and fixed size structures (struct) – strings are not included for instance. Relating Simulink generated code, the biggest limitation is that (fixed size) arrays inside structures are not supported. This can be overcome by not using any other except scalar signals in Simulink model inputs and outputs (relevant only for C-interface). In other words, measurements from four ultrasonic sensors may not be defined as “US\_cm[4]”, but four variables “US\_cm1”, “US\_cm2”..., for instance.

The other limitation in .NET Compact Framework that was found, was lack of binary serialization classes and methods. This was needed when sending data structures between remote user interface (laptop running Windows 7) and embedded computer (eBox). The serialization converts runtime presentation of data to common binary stream, that can be deserialized on the other end. Luckily, this was not the first time in the world this limitation was found, and a good solution for the problem exists: a library called “CompactFormatterPlus”. This library does the same the .NET Full Framework would do, and more. A simple usage of the library for enough for this project.

So main cycle of developing runtime is presented in Figure 11.

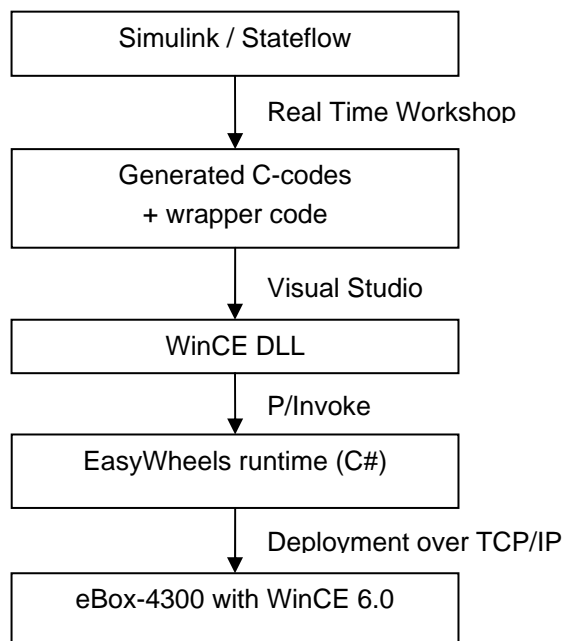


Figure 11: Main stream of software tool chain from Simulink to embedded computing.

One of the manual phases needed to take place is the definition of I/O Simulink interface to C# by hand. These structures in C# and C++ (in DLL) have to be exactly the same, in order to work. This is a

pretty quick phase to do in case of changing the interfaces, but just have to be remembered. The objective was not to have anything like this manual phases, but no solution has been found so far.

Parameters of the Simulink model can be tuned from C# code, as long as in the code generation phase the parameters are marked to be tunable. If marked as tunable, the code generation places the variables in to a separate struct, where they are easily available for a programmer. In EasyWheels 2010 the tunable parameters are stored in XML files (by using .NET framework XML Serialization), which was used to store different parameter sets for different tasks, just like in 4M (Backman et al 2008).

One of the nicest features that help debugging, is that the tool chain also supports the remote debugging of code by using Simulink as interface. Simulink code generation supports “external mode”, which means that the code generated contains certain additional code with which the Simulink running on one computer can get and set data for the C-code running on the other computer. Simulink also offers stack over TCP/IP to do the job, so practically this does not need any custom code. However, again the support for Windows Embedded CE was not available, but it was possible to customize Simulink provided C-code stack a bit, to support also Windows CE builds (Winsock was the difference). The main usage of the external mode is to see the signal values (either by using numeric display or Simulink scope), and by watching in which state Stateflow programmed state machines are, it shows the active states by highlighting them.

The output of this learning process, the main tool chain, was used also in Turtle Beetle (Pentikäinen et al 2010). The technologies and skills required to use and develop the tool chain were educated for Turtle Beetle at the beginning of the project.

## **5.2. Microcontrollers**

CodeVisionAVR was used to program microcontrollers. CodeVisionAVR is easy to use for beginners, as it helps user to select a proper register values with integrated wizard. For advanced user it does not make so much difference, but for beginners it saves time by not making it necessarily to go through datasheets and code examples.

## **5.3. Machine vision**

The machine vision was programmed in Toradex Colibri, running Windows Embedded CE 6.0. The machine vision component incorporates OpenCV library, but actually only the image data formats are utilized, as almost all algorithms suffer lack of FPU. The vision algorithm was written with C++, and compiled to CE compatible native DLL. The runtime was programmed with C#, and it uses P/Invoke to call the DLL functions, to set parameters, and read the outputs. With C# it was much easier to handle network data.

## 6. Algorithms

### 6.1. Machine vision

#### 6.1.1. Exposure control

A typical characteristic in cheap webcams is the optimization for indoor (human face) usage. The exposure control in Logitech QuickCam 5000 works so that it tries to minimize so called “gain” which increases brightness but amplifies noise, and therefore the internal exposure control tries to keep as long exposure as possible. However, this is bad for moving systems, as the images blur in low light conditions. In field robot, usually it does not make a difference if driving in clear sky sunlight, but has difference in cloudy conditions or driving on the evening. This problem was already tried to be solved in Wheels of Corntune (Maksimow et al 2007) and 4M (Backman et al 2008), but it was not possible to program your-own-algorithm, as Windows driver does not support control of exposure from the software. Luckily, the UVC driver for Windows Embedded CE supports direct control of these camera parameters, and now it was possible to do-it-yourself-exposure automation.

320x240 resolution was used. The automatic exposure works in three phases, listed from the brightest conditions to the darkest: 1) gain 0 & exposure > 1/500, 2) gain 0-255, exposure = 1/500, 3) gain 255, exposure < 1/500. The algorithm counts number of pixels and channels that are 255 (which is the maximum). This count number is regulated by a simple controller to 200 (of 320x240x3).

#### 6.1.2. Colour transform

To detect maize plants, a generalized EGRBI transform was used. The original EGRBI transform converts a colour from RGB space to another 3D space: Excessive Green (EG) – Red-Blue (RB) and Intensity. The goal is to detect the green, and the Excessive Green channel excludes intensity changes, so it is easier to threshold the value than in raw G channel. Red-Blue channel makes difference if EG channel says that the pixel is “not green”, to tell “is it more reddish or bluish”. The Intensity channel corresponds to greyscale. The problem of original EGRBI is the assumption that the colour in interest is exactly green-green. In the case of maize plants the colour is not green-green, but more yellow (red-green) than green-green or green-blue. One example of RGB colour of maize is 160-200-90, experienced by Logitech 5000.

In the Finnish series of field robots, EGRBI transform was tried in Demeter robot with a reasonably good experience. For Wheels of Corntune, the method was generalized the first time, and the generalized version was called as “ECCI” by Maksimow et al (2007). In the generalized EGRBI, or ECCI, the idea is that user may select any colour of interest, and the algorithm computes automatically the transformation to form “EG”, “RB”, and “I” channels. EC (Excessive Colour) channel marks how much the value is given in the direction of colour of interest (excluding intensity variation). Intensity is computed just like in the original version, and the CR (Cross) channel base vector is computed as a cross-vector of EC and I channels, to provide more information. If the colour of interest is 0-255-0 (pure green), the transform is exactly the same as EGRBI. The same method was used also in 4M (Backman et al 2008).

In EasyWheels 2010, the ECCI transform is used. The main difference in the previous editions is hidden in the reasoning: threshold is not used, but instead on each channel the probability is calculated by using on piecewise linear curves. For CR channel the symmetry is assumed, but on the other channels not, so the method provides 5 tuning parameters. The overall probability (of being

maize) is computed as a product of the three channels. The parameters are easy to tune by using test images, by first clicking the colour of interest and then finding appropriate values for the corner points of the curves.



Figure 12: Camera picture from top of mast.

### 6.1.3. Projection correction

One of the challenges in the project was to deal with the lack of floating-point unit (FPU) in Colibri. Computation of ECCI, correcting the projection and transferring all the data to navigational computer (eBox) was out of the question. The solution was to use pre-computation to form “blocks”, or regions-of-interest for the image that correspond certain metric area in the field. The angle of camera was set so that the top edge corresponds to 3.00 meters and in case the bottom edge corresponds about 0.70 meters. A “rasterization” of 12cm x 20cm was used in the preliminary tests, and this metric rasterization means that on top of image smaller number of pixels is used to form a block than in the bottom. By taking the height of camera and aspect ratio of image into consideration, the rasterization produces 268 blocks. For all of them ECCI transform and probability computation is done, based on average values in RGB. The resolution could have been even denser, as in Colibri this required only about 17% of processor power.



Figure 13: Multiresolution rasterization and ECCI transform, from the left: EC, CR, I, and total probability

### 6.1.4. Adaptive thresholding

This system also utilized adaptation to finding probabilities. Adaptive thresholding has also been used in Wheels of Corntune (Maksimow et al 2007) and 4M (Backman et al 2008). The assumption in adaptation is that when driving in rows, the average number of pixels which are interpreted as maize, stays (more or less) constant. For adaptation it is possible to set the assumption as a percentage, e.g. 30% has been used, even if this year 10% was a better guess. Here adaptive thresholding controlled upper corner point of EC channel piecewise curve. After the competition and playing with test data it can be said that this was not the best choice.

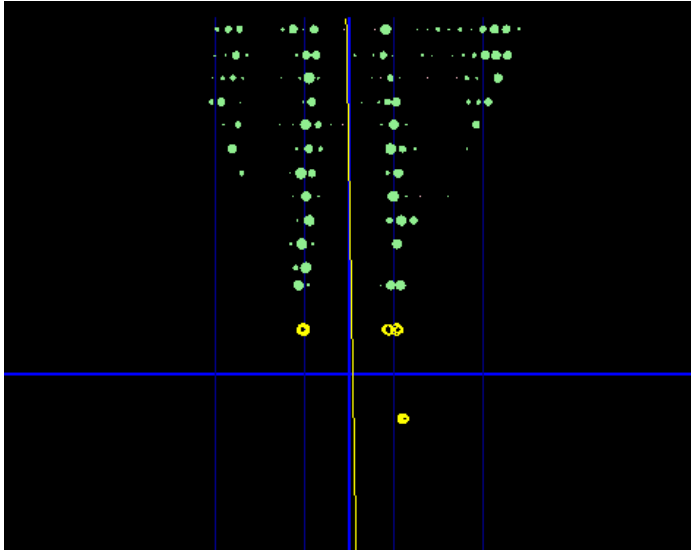
## 6.2. Position in row

The position estimation in row is separated from navigation, in order to tune one in time. The row position estimation means finding the two values: lateral position in row (error sideways) and angular position in row (error in heading). The third value is quality which indicates how well the sensor data fits a model. The row model is simply based on assumption on straight maize rows (in 3 meters ahead – 1 meter behind) and 0.75 meter row width (tunable parameter). As EasyWheels is pretty symmetric and the special feature is two-way driving, the natural selection for robot origin is the center point (where the mast is located).

Ultrasonic rangers, infrared rangers and information from machine vision are all projected to the same 2D metric plane. In case of ultrasonic and infrared sensors, the sensor reading is put to the correct geometric position in the robot coordinate system and perpendicular reflection is assumed. Furthermore, in case of ultrasonic and infrared rangers, also historic measurements from last 20 steps (corresponding 1000 ms) are projected, by using backwards odometry computation (backwards in time, historical steering angles, Euler backwards integration) – and historical projected measurements have a forgetting factor, so that the oldest measurement has only 0.2 weight of the newest. The same principle was used already in Wheels of Corn tune and 4M robots (Maksimow et al 2007, Backman et al 2008).

Compared with earlier editions, this time also the detected maize probabilities are projected in the same 2D space with ultrasonic and infrared measurements. In this way all the information is processed together, with no intermediate rounding as would happen if the same position in row indices were computed separately. However, this worked in principle, but the problems arise as all the information is not in the same scale. Using only the probability and positive detections was not good enough – also positive probability of “not being maize” has to be counted. Due to lack of time and test data this was not possible for this edition, but it will be investigated in the future.

In line fitting two stages were used to reach computational efficiency. In the first stage a rough estimate is found with a multi-resolution search algorithm: at first, 10 degree resolution is used and then the resolution is decreased to one degree – in the angle in question all the data is projected in the same axis and variation (0.75m cyclic) is counted. The higher variation – the more probable heading angle. In the second stage the data is first rotated to the angle given by rough search (which includes camera head turning), and recursive least squares method with 0.75m modulus is applied – to find a more accurate estimate. A visualization of the method is shown in Figure 14.



*Figure 14: Line fitting diagram: green spots are probabilities of being maize from the image processing, and yellow circles are projected maize plants from ultrasonic sensors in corners of the robot. Robot center position is in the center of blue axis. A yellow line is the result of estimated center driving line.*

### **6.3. Navigation in row**

Input to navigation in row is coming from row estimation (lateral error, angular error, and quality) and the output is driving speed, steering angle of front wheels and steering angle of rear wheels. EasyWheels 2010 is using the same navigational principle as its predecessors SmartWheels, Demeter, Wheels of CornTune and Mean Maize Maze Machine (Honkanen et al 2005, Telama et al 2006, Maksimow et al 2007, Backman et al 2008). The principle is simple: two separate PID controllers are used, one of them is handling lateral error control and the other is handling angular error. The output of the first controller is lateral speed (normal to robot heading) and the output of the second is angular velocity. The lateral speed requirement and angular velocity requirement together with set point for robot speed can be converted to steering angles, by using inverse kinematic model of the four wheel steered robot. In EasyWheels 2010 the angular error is also used as feed-forward to steering angles, which makes it easier to tune the controllers. The main idea of this navigation principle is to make the tuning easier: first the lateral control parameters can be tuned and afterwards the angular – 6 parameters to tune at the same time by hand is too much.

### **6.4. Navigation in the end of rows**

For turning at the end of a row, two methods were implemented: The first utilized two-way driving and works like-a-crab. The other is more common; at first it makes a 90 degree turning, then backwards or forwards some distance and finally completes the last 90 degree turn. In both cases, after detecting the end of row, the robot stops, drives backwards a bit (not to exceed a limit of 1.5 meters in the field), and stops again. Stops between the turning phases make it easier to tune the parameters, and for competition they were minimized.

An intention was to use the camera as a main sensor while the robot was driving in headland, like 4M did, to count how many rows are passed, and on the other hand to keep the orientation right. The angle of the camera was kept always towards to the rows and the angle was adjusted by the measurements obtained from the compass and/or odometry. However, we didn't have enough time to

implement camera based navigation at the end of the row, so the competition was done only by using odometry and compass heading.

For improved accuracy to drive a desired distance or a desired angle, a model based odometry was used. The kinematics and identified dynamics of the robot were used in the prediction model to estimate how much the robot would move after setting the driving speed to zero. In other words, the robot all the time thinks “where would I end up if stopping now?”, and this predicted position is used as threshold value in the navigation state machine.

### **6.5. Adaptive cruise control**

Each of the axle modules implements so called cruise control, or in the other words keeps the constant speed. For two-robot co-operative challenge, the other feature from the automobiles was to be implemented, so called “adaptive cruise control” or ACC. This system keeps the constant distance to the other car driving ahead, if the car is detected, and otherwise operates like normal cruise control.

In EasyWheels 2010 the adaptive cruise control is implemented by adding another PI control on top of normal cruise control, which adjusts the driving speed up or down from the desired speed. The other robot sent his speed to EasyWheels, and this was “the desired speed”. The output of the PI controller was limited to -0.2 to 0.2 m/s range, so the tuning up or down was done in a safe range.

Both SRF08 and SRF235 sensors were used. There was not enough time for the deep analysis of sensors in the competition, so the beforehand implemented “use minimum of the two sensors”. In the event demonstration the speed of 0.25 m/s was used and a set point for distance was 35cm.

## **7. Two robot co-operation (with Helios)**

We were asked if we wanted to make a co-operative demo for a new Task in Field Robot Event 2010, with Helios. The first idea was to do it with Turtle Beetle, but quickly it was found out that time was running out of that project, and thus the authors ended up taking EasyWheels out of shelf and making the “EasyWheels 2010”. Helios has participated in Field Robot Events every time since 2007 (Knaupp et al 2007; Meinecke et al 2008) and they won the event 2007, were second 2008, and again winners in 2009; so the project was on solid ground – and it was decided to take as a challenge.

After a quick brainstorming, the representatives of the two teams agreed to have demonstration about “corn harvesting”, where one of the robots acts as a harvester and the other as a transporter. Helios team had some kind of “corn shooter” as a tool for Helios in their mind, so evidently Helios was to be the harvester and EasyWheels 2010 to be the transporter. The first idea was to do parallel driving so that one robot in one row and the other following in the next row. However, this was considered more challenging as there was no clear idea how long maize plants will be during the event, and which sensors could be used to detect the position of the other robot – therefore one driving behind another was considered a better choice.



*Figure 15: EasyWheels 2010 with a black hopper in Field Robot Event test field.*

To demonstrate the job, and taking into consideration a fact that we have only about a day to test the co-operation, the two teams had three aids in use: 1) a wiki-based common workplace, 2) a layered strategy for building up the technology, and 3) simulators to test the communication between the robots.

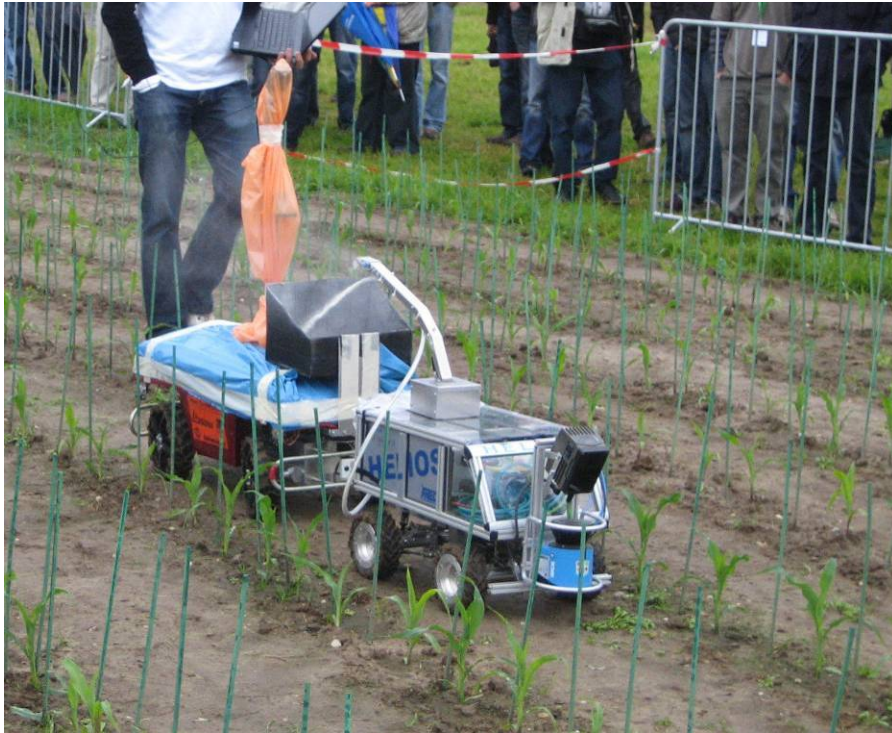
The layered strategy incorporated from the simplest to complex:

1. one driving behind the other, with fixed turnings, no online communication, manifold acting when Helios sees a row, the second robots starting a bit later than the first
2. online communication: EasyWheels2010 informs when "in contact" e.g. set point in gap reached and Helios informs when the headland starts
3. both robots start from headland, Helios first navigates some rows further, informs EasyWheels2010 of how many rows were counted, Helios drives slowly in row, EasyWheels2010 starts and finds the correct row, and catches up Helios, and informs about the "contact", Helios turns the manifold backwards and simulates blowing
4. some real matter is transferred between robots
5. EasyWheels2010 would go after two rows somewhere to "empty" the hopper...
6. ....and come back...
7. programmable turnings for both robots

In the Event, everything was not going as expected. Helios had some problem in their mechatronics (a front wheel steering problem, or something), and unfortunately they were not able to participate in any other task than the two robot co-operative challenge. Both the robots were communicating with the simulators, but in real life there were some problems, as it took for a while to tune the WLAN access point to right frequency band where no 2.4 GHz noise appeared. The starting from the headland and waiting in turnings by EasyWheels was tested in test field, and was proven to be working.

Due to Helios' problems with steering, it was agreed with the jury that Helios will be driven by remote control (Nintendo Wii controller) and EasyWheels 2010 has to act autonomously. Helios had problems with steering, but with remote control this problem was solved, and fortunately we had something to

show. A comparison to layered strategy presented above shows that 1) “following” and 4) “shooting a real matter” were working in the competition demonstration. 2) “communication” and 3) “starting position” were implemented and partially tested, but not demonstrated in the Event.



*Figure 16: Helios and “well protected” EasyWheels doing the demonstration in FRE 2010.*

An experience on two robot co-operative demonstration was positive, as we (the two teams) were able to do some real co-operation. The expected risks were encountered in Braunschweig, as a minor steering servo problem made the autonomous co-operation of the two robots impossible. It was again learned that a robot is a complex system and all the layers in the robot (from mechanics to most advanced algorithms) have to work perfectly to be operative.

## **8. Discussion**

In the Event, the field was different from what was expected. The test field had almost no maize plants, as they were transplanted to the competition field. The only way to do navigation tests in the competition area was to put 8mm diameter gray-green sticks into the soil and try to detect them (see Figure 15). These sticks were also put into the competition fields, to supplement 10-15 cm height maize plants. EasyWheels 2010 used ultrasonic rangers (4x), infrared rangers (4x) and machine vision to detect the rows (and the end of rows). Based on the first manual drive data in the test rows it came evident that infrared rangers were not able to detect any of the sticks (no use), with machine vision it was very hard to distinguish 8mm gray-green sticks from the ground (no use) and out of four ultrasonic rangers only front-left and rear-right sensors were in such condition that they were able to detect 8mm sticks. The other two ultrasonic sensors seemed to work only in range under 12 cm, as they had worn out somehow, maybe the dust from the field had gone inside the sensors. For future robot builders: be aware with SRF08 condition! So the conclusion was that only two of nine sensors were working, and especially when the most powerful sensor (machine vision) was out of the question, it seemed almost impossible to make it to work in the competition tasks.

However, the robot was capable of driving between stick rows surprisingly well, only by using front-left and rear-right ultrasonic sensors, thanks to sophisticated line detection algorithm. We had only one spare ultrasonic ranger for both EasyWheels and Turtle Beetle, and this was replaced in front-right. With that replacement also row end detection was improved a bit, but still the row navigation in backward driving was a bit too waving – which was seen in Task 1.

The next day after the competition it was possible to do test drives in the competition fields, and finally at that time it was possible to use machine vision as there was enough maize plant green. After short calibration EasyWheels was driving pretty well in the rows, and the driving speed could be increased.

Co-operative challenge proved to be a real challenge – as we co-operated with a team from another country and we had only about one day to do (all) tests before the competition. Luckily, we had prepared well for this task, by developing a clear layered strategy how to do the demo – objectives were set high, but still we had in mind that at least some movement had to be done in case of e.g. wireless communication is not working. Pretty standard WLAN communication between the robots was surprisingly one of the problems as in some phase the communication did not work anymore in channel 8 (mysterious noise?), but after an hour or two investigation it started to work again when changing the channel to 9 – a lot of test time was lost for solving this problem. Helios robot had some mechanical and other problems and they were not able to participate in the other tasks, but luckily we were still able to do the two robot co-operative challenge in a bit eased way: Helios was driven in manual control, and EasyWheels followed autonomously in a constant distance.

## 9. Conclusions

A good field robot has to be able to adapt to any kind of field, whether it is in good condition or not. To be good in row driving in any conditions, multiple sensors have to be used as one may work better in some conditions and some in the other. EasyWheels 2010 was equipped only with ultrasonic rangers, infrared rangers and machine vision with turning head – the best detection is achieved if all three types of sensors can detect the row. The test field was very challenging, and two of three sensors were not working, but still some row navigation was possible.

The two robot co-operative challenge was considered being challenging. One robot is a pretty complex system and it can be said that if one piece in the system is not working, the whole robot might not work. The risk in the two robot challenge is about on power three, as there are two robots that have to work by themselves, and also the communication and other interaction between them have to work. This risk realized in our two robot demonstration, but luckily a way to do the demo was found – after a lot of work this was better than nothing – and the jury appreciated it with the first prize in co-operative challenge task.

## Acknowledgments

At first, we would like to thank all the sponsors (Valtra, Koneviesti, Laserle, HP Compaq, HP Infotech and Suomen Kulttuurirahasto) that supported EasyWheels project in 2009. Secondly we would like to thank all the EasyWheels team members in FRE 2009 – the robot was made with good quality and it provided a good starting point for learning new software technology. Especially the axle modules and the suspension system appeared to be bullet-proof technology. We would also like to thank all the Turtle Beetle team (FRE 2010) members for inspiring co-operation and giving a permission to test our robot in the “orange test field”. Gratitude is also given to staff members of Department of Engineering

Design and Production for letting us to play with the robots in the facilities. Finally, we would like to thank all the 44 students participated on Finnish Field Robot Event teams during 2005-2010, to give us motivation to improve education and technology, and to learn more and more about field robots.



## References

- Backman, J., Hyyti, H., Kalmari, J., Kinnari, J., Hakala, A., Poutiainen, V., Tamminen, P., Väättäinen, H., Oksanen, T., Kostamo, J. and Tiusanen, J. 2008. **4M – Mean Maize Maze Machine**. In 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](http://www.fieldrobot.nl/downloads/Proceedings_FRE2008.pdf)
- Honkanen, M., Kannas, K., Suna, H., Syväne, 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**. In Proceedings of the 3rd Field Robot Event 2005. ISBN 90-6754-969-X. pp. 73-90. [http://www.fieldrobot.nl/downloads/Proceedings\\_FRE2005.pdf](http://www.fieldrobot.nl/downloads/Proceedings_FRE2005.pdf)
- Kempainen, T., Koski, T., Hirvelä, J., Lillhannus J., Turunen, T., Lehto J., Koivisto, V., Niskanen, M., Oksanen, T., Kostamo, J. and Tamminen, P. 2009. **Robot Brothers EasyWheels and ReD in Field Robot Event 2009**. In Proceedings of 7th Field Robot Event 2009, Wageningen, the Netherlands. ISBN 978-90-8585-744-0. pp. 37-64. [http://www.fieldrobot.nl/downloads/Proceedings\\_FRE2009.pdf](http://www.fieldrobot.nl/downloads/Proceedings_FRE2009.pdf)
- Knaupp, J., Meyer, R., Meinecke, M., Robert, M., Schattenberg, J. and Schlott, J. 2007. **Autonomous Field Robot System “Helios”**. In Proceedings of the 5th Field Robot Event 2007. pp. 35-44. [http://www.fieldrobot.nl/downloads/Proceedings\\_FRE2007.pdf](http://www.fieldrobot.nl/downloads/Proceedings_FRE2007.pdf)
- Maksimow, T., Hölttä, J., Junkkala, J., Koskela, P., Lämsä, E.J., Posio, M., Oksanen, T. and Tiusanen, J. 2007. **Wheels of CornTune**. In Proceedings of the 5th Field Robot Event 2007. pp. 75-87. [http://www.fieldrobot.nl/downloads/Proceedings\\_FRE2007.pdf](http://www.fieldrobot.nl/downloads/Proceedings_FRE2007.pdf)
- Meinecke, M., Robert, M., Roesler, J., Roos, L., Schattenberg, J., Schwerter, M. and Göres, T. 2008. **Hard- and Software concept of the autonomous Field-Robot Helios**. In Proceedings of the 6th Field Robot Event 2008. ISBN 978-3-00-027341-4. pp. 65-74. [http://www.fieldrobot.nl/downloads/Proceedings\\_FRE2008.pdf](http://www.fieldrobot.nl/downloads/Proceedings_FRE2008.pdf)
- Pentikäinen, J., Pihlanko, M., Pihlanko, P., Helenius, J., Tuhkanen, E., Matilainen, M., Sairanen, M., Oksanen, T., Kostamo, J., and Tamminen P. 2010. **Turtle Beetle**. In Proceedings of the 8th Field Robot Event 2010. (in print).
- Telama, M., Turtiainen, J., Viinanen, P., Kostamo, J., Mussalo, V., Virtanen, T., Oksanen, T. and Tiusanen, J. 2006. **DEMETER – Autonomous Field Robot**. In 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](http://www.fieldrobot.nl/downloads/Proceedings_FRE2006.pdf)
- BaneBots Robot Parts, <http://www.banebots.com>
- EmbeddedPC.NET, <http://www.embeddedpc.net>
- Futurlec, <http://www.futurlec.com>
- ICOP, <http://www.icoptech.com>
- Laserle Oy, <http://www.laserle.fi>
- OpenCV, Open Source Computer Vision Library, <http://sourceforge.net/projects/opencvlibrary/>
- Toradex, <http://www.toradex.com>