Design and Control of the Quadruped Walking Robot ALDURO

Daniel Germann, Manfred Hiller, Dieter Schramm

***Abstract*—Alduro is a four-legged walking robot. Its main goal is to operate in rugged terrain, translating the cartesian operator commands (joystick) into actuator space by coordinating the legs according to the users wish. At the same time static stability has to be guaranteed, obstacles have to be avoided and the posture of the main body kept. This requires an elaborate motion coordination and controller software. One possible way to organise this is described here: by isolating the physical robot from the motion generation (hardware abstraction layer), by using concurrent behaviours for motion generation and by strict modularisation of the software. The selected tools and realtime operating system are described in the last sections.**

***Index Terms*—Walking Robot, Quadruped, Modular Controller Structure, Hardware Abstraction Layer, Free-Gait, Realtime**

I. Introduction

WALKING excavators have been around since the late 1960. Not only have they helped at many work sites in rugged terrain, rivers and mountains, but they also inspired the project of designing and building an automated four-legged robot described in this paper. Whereas in conventional excavators each degree of freedom is operated manually the goal of Alduro was to automate this task and relieve the user of coordinating 12 to 16 actuators.

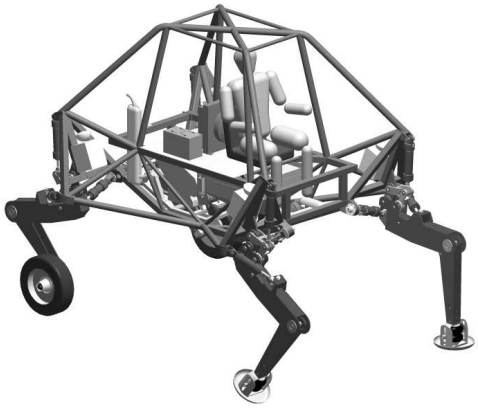

Fig. 1. CAD model of Alduro as a wheeled-legged vehicle.

But to replace the expert knowledge of an experienced operator by a computer is a daunting task. A first step in that direction is taken with this project and an overview given here. Starting with a design study of leg kinematics and a virtual prototype seven years ago the project has grown into the assembly of almost two tonnes of steel and periphery and its coordination by a central computing unit. An exemplary mechatronic task, involving mechanical and electrical engineers, computer scientists, structural analysis, and software design. This paper will focus on some of the subtasks like virtual and real prototypes, isolating the motion generation from the physical robot, modular software design, motion generation of leg and body and lastly realtime requirements of the controller and implementation.

This work was funded by the German Research Council (DFG) and supported by the Federal State of Northrhine-Westphalia.

Daniel Germann is with the Chair of Mechatronics, Department of Mechanical Engineering, University of Duisburg-Essen, Germany.

Prof. Manfred Hiller is the former head of the Chair of Mechatronics, Department of Mechanical Engineering, University of Duisburg-Essen, Germany.

Prof. Dieter Schramm is head of the Chair of Mechatronics, Department of Mechanical Engineering, University of Duisburg-Essen, Germany.

II. ALDURO

This chapter gives an overview of the history and the prototypes of the **A**nthropomorphically **L**egged and Wheeled **Du**isburg **Ro**bot Alduro.

A. Genealogy

The most evident advantage of legged machines against wheel driven vehicles is their ability to move in unstructured terrain. Possible areas of applications for legged working platforms are outdoor tasks as e.g. forestry or avalanche protection building. Smaller robots can be used for inspections of inaccessible, hazardous or contaminated areas.

In alpine regions many walking excavators (by different manufacturers) can be seen. They offer a high flexibility in operating terrain (slopes, quarries, forests, streams) and functionality (excavator, jackhammer, forester). But they need to be operated manually by an experienced driver.

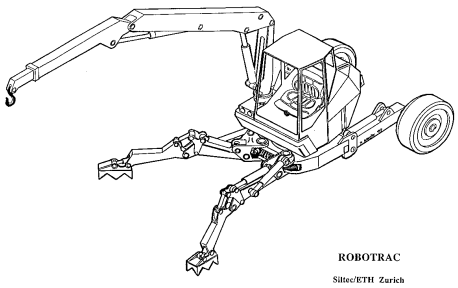

Fig. 2. Design study roboTRAC

With this limitation in mind, in 1992 a design study of the Swiss Federal Institute of Technology (ETHZ), the University of Duisburg and Siltec in Zürich led to the roboTRAC Fig. 2, a walking excavator with enhanced front legs. The additional knee would allow for a walking motion without use of the bucket.

Based on the theoretical works on roboTRAC on its kinematics [6] a new research project was started in 1998. Its main topics were

- better manoeuvrability by increasing the degrees of freedom of the legs,
- additional sensors for ground/obstacle detection, and
- a central computing unit coordinating the movement of the legs.

To achieve the desired higher flexibility the legs kinematics are modelled on the human leg (anthropomorphic). The hip joint is realised as a ball joint, the knee joint as a revolute joint. Three degrees of freedom (d.o.f.) are necessary to place the foot, the remaining redundant fourth d.o.f. can be used to minimise cylinder velocities and strain/stress in the structure of the leg.

Instead of using the anthropomorphic legs only as front legs all four legs are replaced by the new design. Thus we use four identical legs, and have the option to operate the rear legs with wheels as before (to increase velocity in less rugged terrain), or replace them with feet and so get a pure walking robot.

This led to the design of ALDURO. This time not only a case study was to be considered, but the goal was to build a full scale working prototype.

B. Virtual Prototype

Instead of building several small-scale models of the robot before tackling the fully scaled prototype a *virtual model* was used to development the robot and optimise its design. The virtual model consists of a physical model that covers the dynamics of hydraulics and mechanics. With help of the multibody systems simulations library MOBILE [9] the equations of motion and pressure build-up of the hydraulics are generated and integrated.

To validate the computer model a test stand for a single leg was built and compared, with satisfying results [13].

This co-simulation then was used to determine

- the necessary pump flow of the hydraulic system,
- the geometrical parameters of the four-bar mechanisms if the hip joint,
- the forces acting on the joints and actuators of the legs,
- the boundary conditions (forces/torques) of the structural analysis of legs and central body, and
- to test different controller algorithms, like PID and model-based ones.

Based on the results of simulation and FEM analysis a lattice framework design was chosen for the central platform and optimised for low weight.

C. Real Prototype

The prototype currently being built at our Institute (height 3.5 m, weight approx. 1800 kg, see Fig. 3) is equipped with four identical hydraulically actuated legs, with four degrees of freedom each. Each of the 16 joints is actuated by a hydraulic cylinder. To be independent of external power supply a combustion engine (a gasoline engine of Mercedes *smart* car with 45 kW) was chosen for powering the hydraulic pump. The operating pressure of the hydraulic system is 200 bar and the pump flow is 125 l/min.

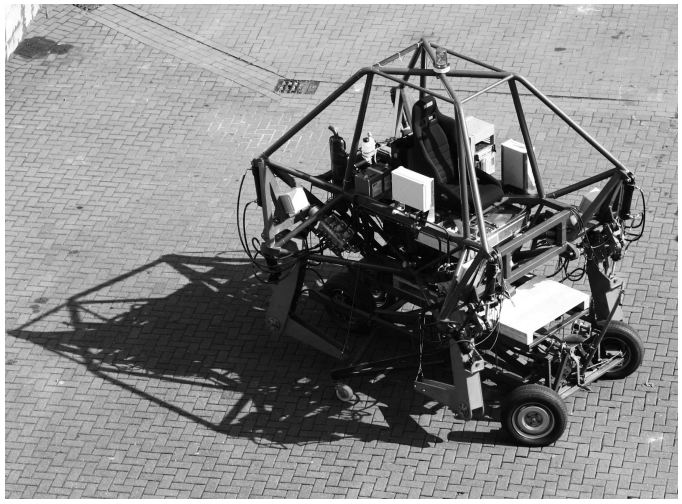

Fig. 3. ALDURO (with legs but no feet yet) sitting on the wheeled assembly platform

As it would be near to impossible to coordinate all 16 hydraulic cylinders manually a more user-friendly way of operation had to be found. Considering that also lorry and car manufacturers are conducting tests with steer-by-wire by means of joysticks it was apparent to use the same technique for ALDURO.

The task of step generation, coordination of the cylinders, safety checks and controlling the cylinder positions is done by an on-board computer. The necessary realtime requirements are discussed in more detail in section IV.

III. Modular Controller Architecture

This chapter describes the modular structure of the controller software. To make the problem of gait generation more generic, a hardware abstraction layer between robot and *Motion Generator* is introduced. The latter consists of several parallel running non-interacting modules whose merged output is the input of the controller.

A. Hardware Abstraction Layer

The task of generating leg movements of a quadruped according to the user commands is complex. There were and are many other research projects that had gait generation as topic. In order to reuse their results and refine them – where necessary and possible – it would be more efficient to do that independently of the actual geometry, topology, and size of the robot. Thus a *Hardware Abstraction Layer* is inserted between *Motion Generator* and robot (Fig. 4).

The robot specific sensor data (e.g. cylinder lengths, hydraulic pressures, currents, feet forces) is preprocessed in the *Hardware Abstraction Layer* and mapped onto "higher", more abstract data, like

- feet positions and velocities,
- estimated central body height,
- which feet contact the ground, and

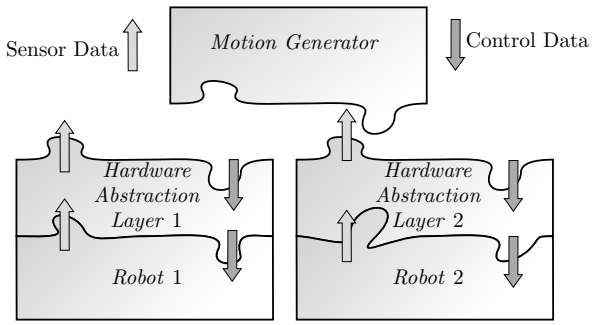

Fig. 4. The *Hardware Abstraction Layer* decouples the *Motion Generator* from the Robot.

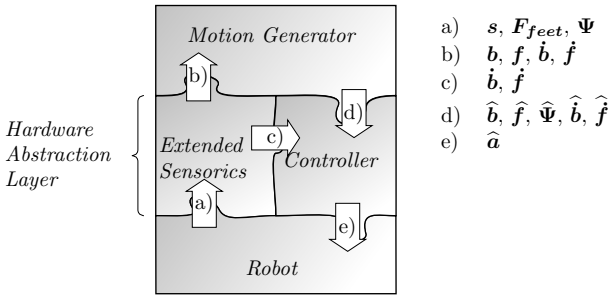

Fig. 5. The four main modules of the controller software. See Table I for the description of the transmitted information.

- proximity of feet (or actuators) to the limit of their workspace.

On the controller side, where commands are passed from the *Motion Generator* to the robot a similar mapping is done: from feet positions/velocities to actuator input values (e.g. currents, valve openings, forces).

The *Motion Generator* can rely on receiving identical input data no matter whether the robot walks on four electrically driven 3-d.o.f. legs or four hydraulically powered 4-d.o.f. legs. Its output, the feet motions, are translated to "robot space" by the *Hardware Abstraction Layer*.

B. Modules

The *Hardware Abstraction Layer* has two tasks: processing the robots sensor data and processing the control data of *Motion Generator*. The sensor data preprocessing tasks are grouped into the *Extended Sensorics* module, the rest into the *Controller* module. This leaves us with the four main modules, shown in Fig. 5. The description of the data is listed in Table **??**.

Data	Description	Ref.
$\boldsymbol{s}$	sensor data (measured joint values)	
$\boldsymbol{F_{feet}}$	measured feet forces	foot
$\boldsymbol{\Psi}$	orientation of body	ground
$\boldsymbol{f}$	position of feet	body
$\dot{\boldsymbol{b}}$	velocity of body	ground
$\dot{\boldsymbol{f}}$	velocities of feet	ground
$\widehat{\boldsymbol{b}}$	set-point position of body	
$\widehat{\boldsymbol{\Psi}}$	set-point orientation of body	body
$\widehat{\boldsymbol{f}}$	set-point position of feet	body
$\widehat{\dot{\boldsymbol{b}}}$	set-point velocity of body	ground
$\widehat{\dot{\boldsymbol{f}}}$	set-point velocities of feet	ground
$\widehat{\boldsymbol{a}}$	actuator set-point values	

TABLE I

DATA EXCHANGED BETWEEN THE FOUR MAIN MODULES. WHERE NECESSARY THEIR REFERENCE SYSTEM IS INDICATED.

The interface between *Motion Generator* and the *Extended Sensorics / Controller* modules is fixed. Between *Extended Sensorics*, *Controller*, and *Robot* the interface changes depending on the physics of the robot. For ALDURO there are three implemented modules:

- an implementation of the interface to the input/output cards, communicating with the real robot,
- a simulation model of the dynamics, with ground contact elements, and
- a simplified kinematics-only simulation.

These modules have a common interface to *Extended Sensorics* and *Controller*, which makes them easily interchangeable without even recompiling the program (Fig. 6). Before trying *Motion Generator* or *Controller* on the real prototype they can be tested on the simulation without significantly changing to the software.

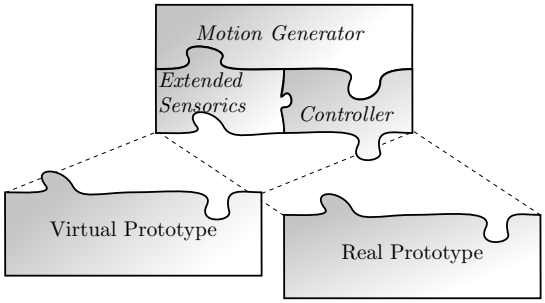

Fig. 6. Different robot modules for ALDURO (real and virtual) with identical interfaces

By further breaking down the tasks of the four main modules into smaller units the modular concept is extended to several layers. All modules can consist of other modules, thus representing a group of modules. This rigid structure (fixed interfaces, hierarchy) might interfere sometimes with simple solutions of a programming problem but makes the software more transparent. The abstraction through interfaces facilitates the substitution of modules with the same function but different implementation. As e.g. foot trajectory generator, that could base on a cycloid, a four-phase finite-state-machine (see section E) or some energy optimised trajectory (see Fig. 7). The choice which module to load is made at the initialisation of the program and remains constant until the program is stopped.

C. Concurrent Behaviours

To implement the complex task of co-ordinated motion generation it is divided into sub-tasks (behaviors) that are

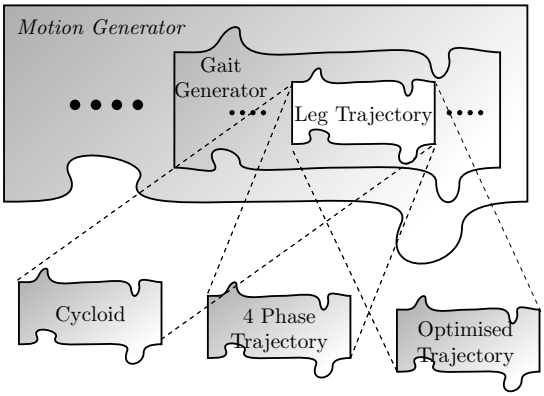

Fig. 7. Different modules for foot trajectory generation

put into modules. There are two principal ways to organise the modules: parallel and/or hierarchical. The well known "subsumption architecture" is based on the latter [1]. The modules are ordered vertically according to the priority of their task and can influence or block inferior modules. The advantages of this concept is a transparent and intuitively understandable layout of the dependencies among the modules. This cross-linking among the modules has the disadvantage that the modification, insertion or deletion of modules always affects the neighbouring modules and may cause a redesign of the whole architecture.

The approach chosen here is to organise the modules on one single level. Each module has full access to all (pre-processed) sensor data but not to any other module [18]. It generates an output according to its task and a corresponding weight. All outputs are weighted and fused and used as input of the controller. The only way for modules to communicate with each other is through the "world". The output produced by the modules influences the robot, this is sensed and interpreted by the *Extended Sensorics* and that data again is available to all.

Since the modules are formally independent of each other the generation of each weight has to be co-ordinated, to guarantee that e.g. the "stability" module is able to overrule other modules in case of a critical state. .

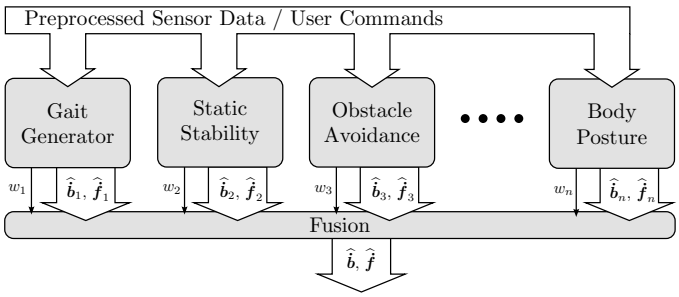

Fig. 8. Concurrent behaviour modules with outputs $\widehat{\boldsymbol{b}}$, $\widehat{\boldsymbol{f}}$ and merging weights w

To conform with the requirements of the *Hardware Abstraction Layer* (subsection A) the outputs have to be "robot independent", i.e. feet and central body related. Possible candidates are positions, velocities, accelerations, and forces. The latter three have in common that they are applied relative to the current position of the robot. Whereas a set-point position is an absolute value, that must not be too far away from the current position, or else the controller is bound to produce a "jump".

If we have only one motion generating module this could be solved by having one internal set of set-point positions that is constantly kept up-to-date with sensor values. But walking is only one possible mode of operation (see subsection F). When switching between operational modes this set-point values would have to be passed from one module to the next. But this violates the idea of concurrent operating modules without inter-modular communication.

Instead of using an internal representation of the robot and its surrounding world (i.e. set-point positions) [2] and [18] suggest to use the "real world" to represent itself. All modules use the pre-processed sensor data as a snapshot of the robots current state and act accordingly thereupon. So all possible outputs are relative to the measured state. Like velocities, accelerations or forces. The fusion of those entities can be very simple: a weighted summation of vectors. Doing the same with relative set-point positions is not feasible, as the scaling (weighing) of the position vectors would scale the absolute position, not only the (relative) difference.

For the reasons above the output of a motion generating module m are the velocities of the central body $\widehat{\dot{\boldsymbol{b}}}_m$ and the four feet $\widehat{\dot{\boldsymbol{f}}}_m$ are chosen. The fusion consists of the weighted (weight w_m) summation of n velocity vectors (1).

$$\left[\widehat{\dot{\boldsymbol{b}}}^{\mathrm{T}}, \widehat{\dot{\boldsymbol{f}}}^{\mathrm{T}}\right]^{\mathrm{T}} = \frac{1}{n}\sum_{m=1}^{n} w_m \left[\widehat{\dot{\boldsymbol{b}}}_m^{\mathrm{T}}, \widehat{\dot{\boldsymbol{f}}}_m^{\mathrm{T}}\right]^{\mathrm{T}} \tag{1}$$

D. FreeGait

There do exist several concepts to control the statically stable "walking" of machines. Basically there are two different perspectives to look at this problem. One is to focus on the legs and to search for an optimal pattern that coordinates the leg movements such that at the end the body moves in the desired direction. The other perspective is to focus on the body. The body is moved in the desired direction and the legs follow "on their own".

These two perspectives lead to two different concepts for generating gait patterns. In the first case sets of precalculated patterns for specific tasks, like "walk straight ahead" or "turn left", are optimised for some kind of criteria, as there are speed, stability and/or energy consumption [3]. The order in which the legs move is predefined as well as the parametrised foot trajectories. These gait patterns show a good performance on flat ground but are flexible only to a certain degree.

The second perspective leads to the so called *free gait* that represents an heuristic algorithm deciding after every single step what movement to do next. Based on the operator input and the momentary position of body and feet, the next set of movements is determined. This can be a step, a body movement or a combination of both. For walking

machines intended to work in unstructured terrain and to move in all directions this method is an ideal solution as the algorithm works with no direction of preference and is highly flexible.

The generic free gait algorithm can be described with the steps:

1. The pattern generator selects a leg to perform the next step.
2. The leg controller performs the step of the selected leg.
3. The body controller moves the remaining legs to achieve the desired body motion.

Free gait concepts differ in the criteria used to select the next leg to move. One possible free gait concept, called *reflective walk*, selects the leg that would increase the stability most when moved in walking direction [10]. [8], [4] use weight-ratios, describing the distribution of the robots weight on its feet, to plan the next move. Another criteria – the one chosen here – is to select the leg with the smallest geometrical margin, i.e. the next one likely to be in a stretched position [17]. In case the leg cannot be lifted because the robot would become unstable a corrective movement of the body (centre of gravity) is superimposed on the desired motion to relieve the selected leg [14].

Fig. 9 shows the basic principle of the *free gait* implemented for ALDURO. The main criteria for choosing the next leg is its proximity to its limit of workspace. As it is too time consuming to compute the cartesian workspace of a foot online the workspace of the actuators is used here. Instead of a three dimensional space we get a four dimensional space, the four cylinder lengths being the four coordinates x_i, $i = 1..4$. In *Extended Sensorics* the proximity of each actuator to its end-of-travel (*eot*) is computed as

$$eot_i = -ln(x_{min,i} - x_i) - ln(x_{max,i} - x_i)\ , \qquad (2)$$

with $x_{min,i}$ and $x_{max,i}$ as minimal and maximal length of the i^{th} cylinder.

To decide whether a leg will soon reach the end of its workspace its not the distance only that is of interest. The question is also how fast it is moving towards this limit. This can be expressed as $v_{eot,i}$

$$\delta_{eot,i} = \frac{d}{dx_i} eot_i = \frac{1}{x_{min,i} - x_i} + \frac{1}{x_{max,i} - x_i} \qquad (3)$$

$$v_{eot,i} = \frac{d}{dt}(eot_i) = \frac{d}{dx_i}(eot_i)\frac{dx_i}{dt} = \delta_{eot,i}\frac{dx_i}{dt} = \delta_{eot,i} v_{x,i} \qquad (4)$$

The value *proximity* used in Fig. 9 is the maximal value of $\delta_{eot,i} * sign(v_{eot,i})$ of all four cylinders.

Simulations with a simplified two-dimensional model [5] have shown that deadlocking of the algorithm is a problem that has to be tackled. A first addition that showed good results is to check whether the leg to move is the front one of the two rear legs. If so it is not to be moved next.

E. Leg Trajectory Generation

To realise the generation of leg trajectories without precise kinematic knowledge of workspace, step length, and

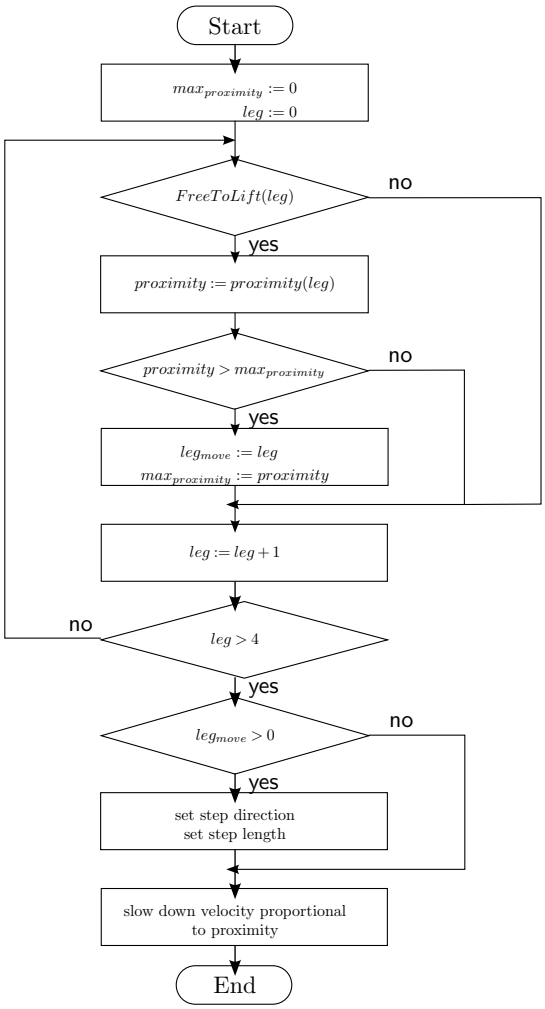

Fig. 9. *free gait* pattern generator

step height the modular concept is used here as well. The stepping motion is divided into four phases: lift the foot, swing it, lower it to the ground, and stance. Once the step is triggered by the pattern generator the *lift* module reuses the memorised step height to lift the leg off the ground and to stop as soon as the height (or end of workspace) is reached. Simultaneously the *swing* module starts as soon as the sensors detect the foot's take off, to slowly increase the speed in the desired direction. The closer the leg gets to its kinematic margin – or if the pattern generator stops the step – the more it is decelerated. At this point the *lower* phase starts and moves towards the assumed ground surface (based on the memorised height from the last step). The process is continued at a constant low speed until ground contact is detected or the whole manoeuvre is cancelled. When the ground is reached the *stance* phase generates a small but constant velocity pointing downwards to keep the foot firmly on the ground. This would result in a upward motion of the platform, were it not for the *body posture* module that generates a corresponding downward velocity for the central body, cancelling out the *stance* velocities in the fusion module.

Parallel to the foot trajectory generator there is the mod-

ules for the *obstacle avoidance* reflex [12]. If a leg touches an obstacle during its trajectory it has the option to cross the obstacle or to retract the foot, depending on the height of the obstacle and the position of the leg. The same applies to the situation that a foot is lowered without finding ground contact. It has to be retracted.

Whereas in pre-computed patterns stability is already considered (and even maximised) the free gait described above so far ignores it. As static stability solely is a question of keeping the centre of gravity inside the supporting polygon it can be delegated to the independent module *static stability* that superimposes a velocity on the body to keep the stability margin above a set minimum.

F. Operational Modes

Walking is only one mode (*Operate*) of a walking robot, albeit the main one. But before starting to walk, the robot has to prepare itself. In the case of ALDURO this means getting up from the mobile assembly and parking vehicle (mode *Straighten*). After standing on its own legs and the parking vehicle is moved away the robot is ready to operate. Before parking again the four legs have to be spaced evenly and the central body levelled and lifted to a predefined height (mode *PrepareToPark*). When the parking vehicle is safely under the robot, it can lower itself onto the vehicle and retract the legs from the ground (mode *Park*).

It is the human operator that initiates the switches between those four modes. But it also needs the controller software to check the validity of the operator commands. When parked, a *Straighten* has to be completed before the robot is allowed to start to *Operate*. For all modes there are conditions to be satisfied before switching to them. An natural approach to this problem is using a finite state machine. The four modes above correspond to four states. The state transitions are in response to input changes, i.e. operator commands and sensor inputs. The output events are associated with states, which makes it a Moore Machine [11].

For each state (except *Operate*) there is a corresponding "finished" state, that is reached automatically when the sensor inputs tell a state that it has reached its goal. For *Straighten* this means that after all four feet have touched ground, and the central body has been lifted by a defined height the state *Straightened* is reached.

Fig. 10 shows the states and the state transitions. Connections with arrowheads depict externally triggered state changes, i.e. user commands. The connections ending with a circle are reached "automatically", depending on the sensor inputs, i.e. usually when a state has finished its task.

As ALDURO is an experimental system the operator should at any moment be able to pause the operation of whatever the robot is doing. Therefore, for each of the four main states an additional "pause" state is added. At the pausing command of the operator the current state is set to its "paused" version, and back, see Fig. 11.

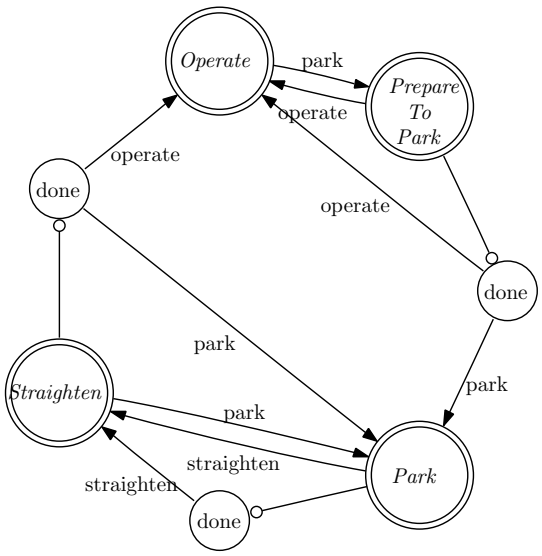

Fig. 10. Operational modes as states of a finite state machine

Product	Soft	Hard	License
RTLinux		x	GPL/comm.
RTAI Linux		x	GPL
ElinOs (using RTAI)		x	free/comm.
vxWorks		x	commercial
QNX		x	commercial
LynxOS		x	commercial
Windows CE	x		commercial
Embedded Linux	x		GPL
manually trimmed Linux	x		GPL

TABLE II
A LIST OF REALTIME OPERATING SYSTEMS

IV. IMPLEMENTATION

The problem of realtime requirements is discussed and the software framework MCA introduced.

A. Realtime requirements

There are several definitions for "realtime", "hard realtime" and "soft realtime". Most of them are contradictory. The definition from IEEE [7] is: *Realtime in operating systems: the ability of the operating system to provide a required level of service in a bounded response time.* The distinction between "soft" and "hard" seen from a task perspective is that the missing of a guaranteed response time in a "hard realtime" task is catastrophic (airplane), whereas in a "soft realtime" task only unpleasant (video/audio applications). Some possible corresponding operating systems (RTOS) are listed in Table II

As ALDURO is able to harm humans during operation, being in close contact to them, timing delays in the controller are dangerous and therefore not acceptable. The envisaged control cycle is 1 ms, with a scatter of ±5%. This requires one of the "hard" RTOS in Table II. Whether the chosen cycle time and the computer are fast enough

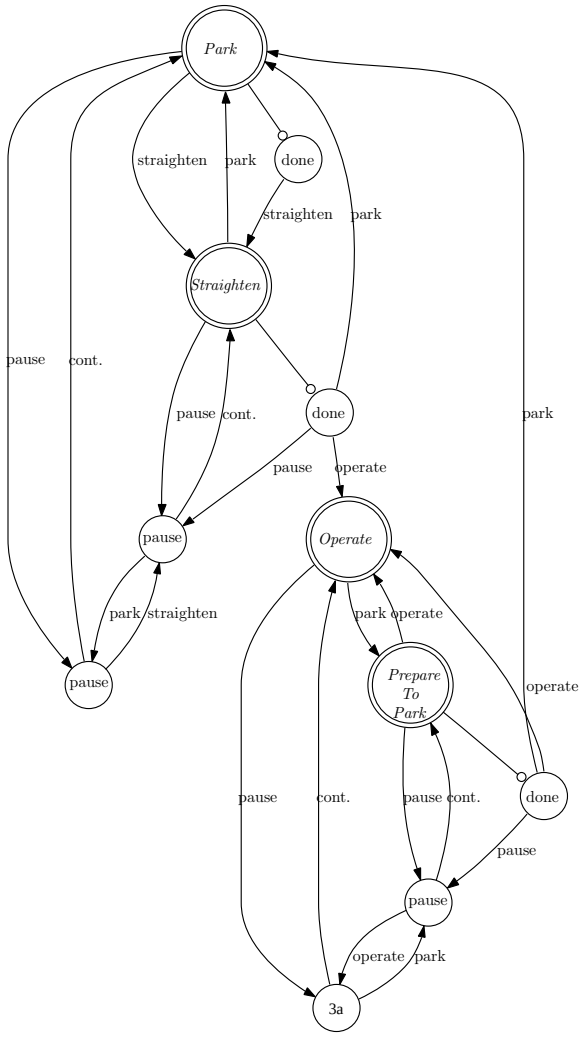

Fig. 11. Operational states with pausing

preliminary tests will have to show.

The commercial systems are stable, well tested and supported, but expensive. Besides they bring their own sets of development tools, which usually do not offer the same level of comfort as standard Linux tools. RTAI and RTLinux allow to work and develop on a standard Linux system, without the necessity of getting acquainted with a new operating system. A non negligible drawback of the free Linux based RTOS is the lack of realtime supporting drivers of most hardware (digital and analog I/O cards). This has to be considered carefully when choosing RTOS and hardware.

Next to the command input elements (joystick, switches, etc.) the operator also needs feedback from the system. Due to the complexity optical indicators only will not suffice to transport enough information to the human user. A display is necessary. In order to free the main controlling computer of the sometimes time consuming task of rendering graphics a second unit (laptop or additional CPU-board) will be assigned to that task.

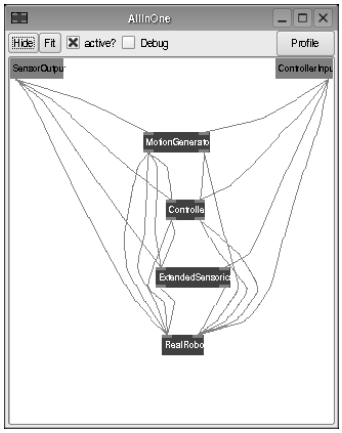

Fig. 12. The four main modules displayed with `MCAbrowser`.

B. MCA2

Summarising all requirements for the controller system we get

- C++ based, to link the multibody systems library
- modularity
- realtime capability
- network transparency (to distribute on several CPU's if necessary)
- provides a graphical user interface
- GPL license (to reuse it on other, also external projects)

The Modular Controller Architecture (MCA, [16]) meets the requirements above. It provides a programming framework in C++ that enforces modularity, offers network transparency, and supports both RTAI and RTLinux (as well as Windows and Linux). Graphical user interfaces (`MCAGUI`) can be assembled from a library of widgets, such as sliders, oscilloscopes, LEDs, 3D-graphic plugins and more. For debugging purposes another tool (`MCAbrowser`) shows the groups and modules, the inter-modular connections and the current values. To isolate and test single modules or groups they can be deactivated or their inputs or parameters changed.

MCA is based on a bidirectional data transportation scheme. In a fist phase the *sense* data is passed from robot up to the user. The second phase passes the *control* data from the user down to the robot (see Fig. 12 and [15]).

When stepping into *Motion Generator* the modules (light grey) and module groups (darker grey) and their connections can be seen in Fig. 13. On the fourth level three of the four operational modes can be seen *Straighten*, *Park* and *Operate*.

For controlling the hardware and the operational states the operator is provided with one or more graphical interfaces. Fig. 14 shows the interfaces developed so far.

The combination of Linux, RTAI, MCA and I/O hardware with RTAI support have turned out to be a satisfying compromise between comfort, support, stability, realtime requirements, and cost for an academic project.

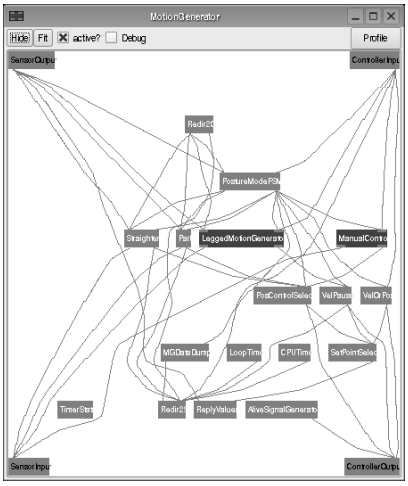

Fig. 13. The *Motion Generator* group displayed in `MCAbrowser`.

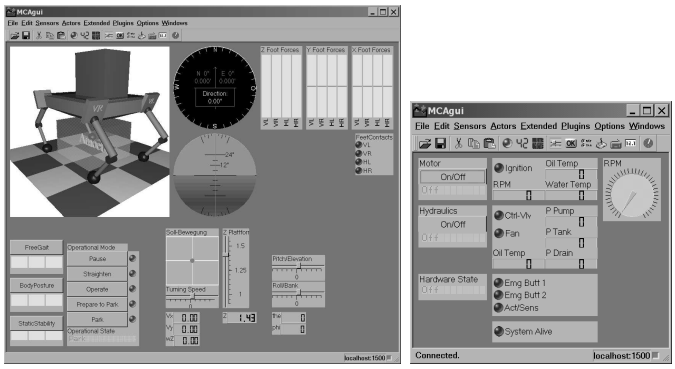

Fig. 14. User interfaces for normal operation (left) and control of hardware (right).

V. Conclusion

This paper has illustrated how the need (desire) to automatise walking excavators has led to the design and realisation of the four-legged walking robot Alduro. With help of a virtual prototype the dimensions and specifications of the robot and its peripheral aggregates have been calculated.

To isolate the physical robot from the motion generation unit the concept of a hardware abstraction layer has been adopted. This allows for the *Motion Generator* of statically stable walking quadrupeds to be developed independently of the robots geometry, sensors and actuation type. The *Motion Generator* itself is subdivided in different separate tasks that run in parallel modules, communicating with each other through the real world only. To facilitate this a modular software design has been decided upon.

Probably one of the main issues of the project of building a quadruped robot is the gait generation. Especially when operating in rugged terrain and wanting to move in all directions a flexible *free gait* algorithm is essential. To choose the leg approaching fastest its limit of workspace as next stepping leg was selected from several different methods.

The controlling of a robot requires realtime capabilities of the operating system. Due to the broad spectrum of tools and low costs the choice was a free realtime extension of standard Linux, RTLinux or RTAI. The software framework MCA fulfills the requirements of modularity, flexibility, graphical tools, realtime support, and network transparency. Thus the controlling software of Alduro is an assembly of a plethora of MCA modules and module groups.

References

[1] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, March 1986.

[2] J. Connell, "A colony architecture for an artificial creature," Ph.D. dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1989.

[3] M. Frik, A. Buschmann, M. Guddat, M. Karatas, and D. C. Losch, "Generation, evaluation and visualisation of gait patterns for walking machines," in *Proceedings of the 1st IFAC-Conference on Mechatronic Systems, International Federation of Automatic Control*, Darmstadt, Germany, 2000, pp. 635–640.

[4] F. Hardarson, "Stability analysis and synthesis of statically balanced walking for quadruped robots," Ph.D. dissertation, The Royal Institute of Technology of Sweden, Stockholm, 2002.

[5] C. Heckhoff, "Free-Gait: Gangmustergenerierung für den ALDURO," Projektarbeit, Universität Duisburg-Essen, 2005, lehrstuhl für Mechatronik.

[6] M. Hiller and T. Schmitz, "Kinematics and dynamics of the combined legged and wheeled vehicle 'RoboTRAC'," in *Proceedings of the CSME Mech. Eng. Forum*, Toronto, June 1990, pp. 387–392.

[7] IEEE, "IEEE Std 1003.1 (POSIX-1)," http://standards.ieee.org, 1999.

[8] J. Ingvast, C. Ridderström, F. Hardarson, and J. Wikander, "Towards walking in rough terrain – control of walking," in *Proceedings of the 6th International Conference on Climbing and Walking Robots*, Catania, Italy, September 2003.

[9] A. Kecskeméthy, "Object-oriented modelling of mechanical systems," in *Kinematics and Dynamics of Multi-Body Systems*, ser. CISM Courses and Lectures, no. 360. Wien, New York: Springer-Verlag, 1995, pp. 217–276.

[10] T. Miyashita, K. Hosoda, and M. Asada, "Reflective walk based on lifted leg control and vision-cued swaying control," in *Proceedings of the International Symposium on Climbing and Walking Robots*, 1998, pp. 349–354.

[11] E. F. Moore, "Gedanken experiments on sequential machines," *Automata Studies, Princeton University Press*, vol. 34, 1956.

[12] J. A. Morgado de Gois and M. Hiller, "Implementation of a sensor system for obstacle avoidance at a four-legged robot," in *Proceedings of the Eleventh International Symposium on Dynamic Problems of Mechanics*, Ouro Preto, Brazil, February 28–March 4 2005.

[13] J. Müller, *Entwicklung virtueller Prototypen des hydraulisch angetriebenen Schreitfahrwerks ALDURO*, ser. Fortschritt-Berichte VDI, Reihe 1, Nr. 356. Düsseldorf: VDI-Verlag, 2002.

[14] D. J. Pack and A. C. Kak, "A simplified forward gait control for a guadruped walking robot," in *Proceedings of the 1994 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, Munich, Germany, 1994, pp. 1011–1018.

[15] K.-U. Scholl, J. C. Albiez, and B. Gassmann, "MCA - An Expandable Modular Controller Architecture," in *3rd Real-Time Linux Workshop*, Milano, Italy, 2001.

[16] K.-U. Scholl, "Modular Controller Architecture MCA2," http://mca2.sf.net, 2001, Forschungszentrum für Informatik, Interaktive Diagnose- und Servicesysteme.

[17] S.-M. Song and Y.-D. Chen, "A free gait algorithm for quadrupedal walking machines," *Journal of Terramechanics*, vol. 28, no. 1, 1991.

[18] M. Zimmermann, "Concurrent behaviour control – a systems's thinking approach to intelligent behaviour," Ph.D. dissertation, Swiss Federal Institut of Technology, Zürich, 1993.