INFOTEH-JAHORINA Vol. 9, Ref. A-17, p. 83-86, March 2010.

DESIGN OF AN OPEN SOFTWARE ARCHITECTURE FOR LEG CONTROL OF A WALKING ROBOT

Mladen Milushev,
Faculty of Mechanical Engineering, Technical University of Sofia

Abstract: *In this paper we present a general description and a Software design of a six-legged laboratory prototype robot. The work presented in this paper has been carried out within a project concerning the investigation of a modular architecture for control of mechatronic multi-link structures at the Faculty of Mechanical Engineering, Technical University of Sofia.*
The presentation describes the software that is designed to control the hardware and the actuators and thus the whole machine. The software design is described in detail, the division into modules, each module and the important algorithms. Thus it is easier to grasp is the description in the form of block diagrams. In addition, it should include possible extensions already in the planning.

1. INTRODUCTION

Six legged locomotion is the most popular legged locomotion concept because of the ability of static stable walking. The hexapods are often inspired by nature; two examples of such robots are Lauron [1] and Genghis [2]. Most of the walking robots are laboratory prototypes [3, 4, 5], but there are also few walking machines built for specific applications, such as SILO06 [6], a six-legged robot built for humanitarian demining.

In this paper we consider a walking robot with six identical legs equally distributed along both sides of the robot body in three opposite pairs. The leg joints are driven by pneumatic muscles (FESTO). So far the six-legged walking robot (Fig. 1.) has been developed using Solid Woks.

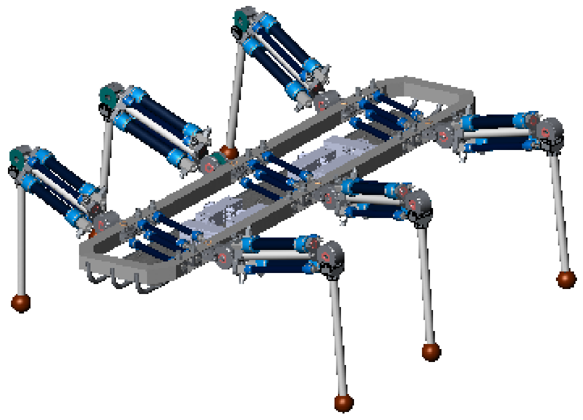

Fig.1. Leg-to-body attachment design.

2. HARDWARE PLATFORM

For the walking robot ***BiMoR*** (**B**iologically **M**otivated **R**obot) a hierarchical and distributed computing architecture has been selected. By distributing the possibility of concurrent control functions is implemented on various micro-controllers. Through the concept of distribution the need for communication is generated. The communication based on the master-slave principle with provides a suitable option for the control system for keeping the protocol economical and within the determined time limits for securing safety to the critical functions. It is important that the used sensors provide information about the absolute co-ordinates. The hardware of the control system must fulfill the following tasks:

- collection and analysis of the measured variables;
- calculation of tax information;
- output control signals to the actuators.

For executing the basic legs functions like the closed-loop joint control (valve control, recording signals from the joint encoders) six R8C/23-microcontrollers are installed. On a basic level each sensor and actuator are connected with the interface board to the micro-controller board. The R8C/23 microcontroller is installed on industrial controller boards and contains one Full CAN module, which can transmit and receive messages in both standard (11-bit) ID and extended (29-bit) ID formats.

3. SOFTWARE SYSTEM DESCRIPTION

The software for the leg's local control is comprised of five modules shown in Fig. 2 along with the related interactions. Both processes run in cycles the first one - *JC_Таск, **J**oint **C**ontrol,* tracks down and regulates the motion of the three joints; the second one *CAN_Таск, **C**ontroller **A**rea **N**etwork* is in charged with the Master generated messages. These processes are regulated by the *Main* Module.

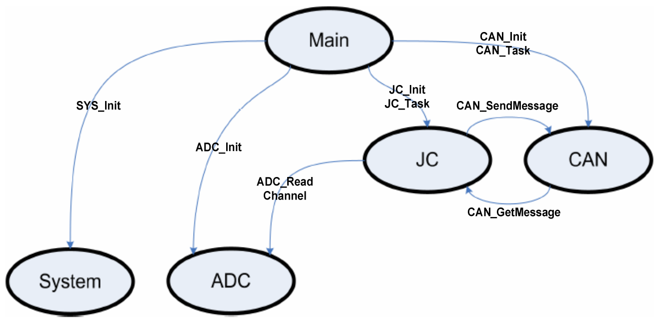

Fig.2. Software modules and their interactions

3. 1. MAIN MODUL

This module is the main module (Fig.3.). It contains only one main function, which has the role of the operating system. It initializes all the modules in the correct order and carries out both processes are cyclical. It obeys to the following initialization rules:

- The system module should be initialized before CAN and ADC, as the CAN and ADC clocks from the Main-clock dependent.
- ADC will be initialized before JC just as with the JC-initialization of the A/D converter - the leg is put in the initial position.
- CAN must be initialized before JC, as in the JC-initialization a CAN module is used. The microcontroller sends this message to the master: “successful initialization” or “initialization error”.
-

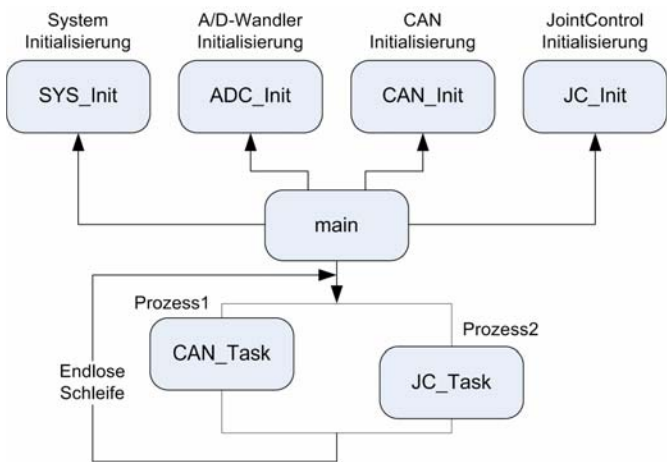

Fig .3. Block diagram of main function.

3.2. SYSTEM MODULE

Simple module to the system initialization – in this case it will be the clock source, clock pre scalar configured and parallel inputs and outputs defined and configured.

3.3. ADC MODUL

This module does the functions of configuration and access to the A/D converter of the microcontroller. The configuration is done through the following steps:

- Choice and setting of the AD channels
- ADC-Clock Set (10 MHz)
- ADC-resolution set (8-bit mode)
- ADC-set mode (one-shot mode)
- A/D conversion method you choose (with Sample and Hold)

The ADC module is currently used to measure the actual value of each of the three joint angles provided but also for other local sensors (for example force sensor and pressure sensor). This will gain through an interface function. She gets as parameters the number of analog input, and then read it very easily to other sensors - one must know only the number of analog input.

3.4. CAN MODUL

Module to configure and control the CAN controlle - implemented were the following functions (Fig.4.):

- *CAN_Init* - function for setting the CAN SFRs: CAN clock (10 MHz), CAN baud rate (500 kbps) and CAN mode (Basic CAN),
- *CAN_ConfigRxSlot* - function for setting a CAN-slot as a CAN-RX slot. There are also interface functions to communicate with the master available.
- *CAN_SendMessage* - Function to send a message to the master via the CAN bus.
- *CAN_GetMessage* - When a new message is completely received it will be copied using this function from the CAN internal RX-buffer in a application RAM-buffer. The execution should happen as quickly as possible because it is being called into the CAN-RX-interrupt.
- *CAN_Task* - one of the two cyclic processes. This function checks whether new messages exists in the receive buffer and if so - they are removed (consumed) from the buffer and processed further.

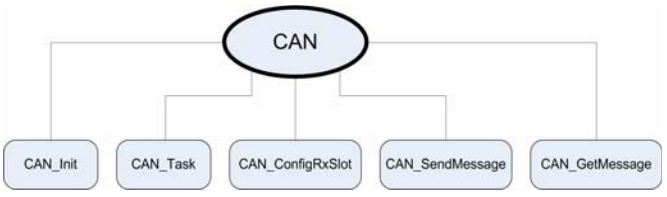

Fig. 4. Functions implemented in the CAN module.

3.5. JC MODUL

In this module two abstract levels are introduced - leg and joint. Each level is assigned to its own state, which is updated regularly and individually. In Figure 5 are shown the functions that are to be implemented in the JC module.

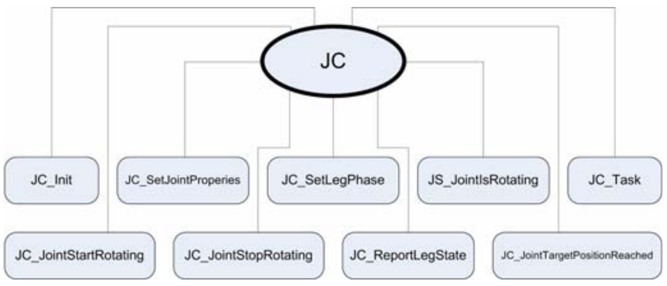

Fig. 5. Functions implemented in the JC module.

Each of these functions is described in detail further in the paper.

JC_Init Function

After switching on and initialization of the system, the ADC- and CAN-modules, each leg enters the state *LEG_STATE_POWERED_UP* and "reports" on to the master. If all six legs are prepared (the master has received six certificates), the master sends to the 1st, 3rd and 5th leg a *INIT-command* - put the leg in a starting position. Each of these legs occurs in the state *LEG_STATE_INIT* and performs an INIT-procedure:

- The leg is lifted (by the β-joint);

- The remaining two joints (α and γ) are set in start position;
- The leg is placed (again, through the β-joint);

  The first joint (angle **ά**) is perpendicular to the robot's body; the first (angle **ά**) and second (angle **β**) joints are perpendicular το one another, while the second (angle **β**) and third (angle **γ**) are parallel.

At this point, all three joints are in their initial position (in the state *JC_STATE_STATIC_INIT*, - the leg is initialized and the state *LEG_STATE_READY* is set. A confirmation message to the master will be sent. All three legs have sent the confirmation and then sent the INIT-command to the master of the 2nd, 4th and 6th legs. The legs are initialized in pairs so that the body always remains stable.

JC_SetJointProperties Funktion

If the leg is in the state *LEG_STATE_READY* it can accept set-points. They will be transferred by the master of the CAN bus and checked through the function *JC_SetJointProperties* for validity. The valid values are in the range [0x06 ... 0xF9] or [26.5 º -26.5 º ...]. For security reasons 1º shift on the left for both side-positions is allowed. The values represent the start and end positions where a phase change takes place. The "longest" step is defined, for example, at a starting value of 0x06 (26.5 º) and at a final value of 0xF9 (-26.5 º).

JC_SetLegPhase Funktion

When the set points are accepted the leg can be set in motion. The command is sent again from the master through the CAN bus. Upon receiving a suitable value (stance or swing phase) the states of the three joints will be set depending on the phase (JC_STATE_ROTATING_START and JC_STATE_ ROTATING _END).

JC_Task Funktion

The JC_task is the second cyclic process implemented in a state machine (Figure 6.). Apart from the stance and swing phases known the Master, additional ones which are used in local control are introduced.

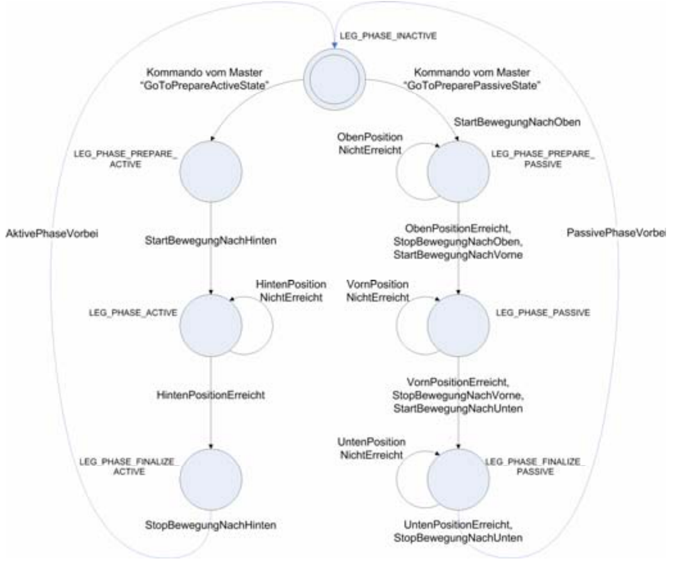
LEG_PHASE_INACTIVE
Kommando vom Master "GoToPrepareActiveState"
Kommando vom Master "GoToPreparePassiveState"
StartBewegungNachOben
ObenPosition NichtErreicht
LEG_PHASE_PREPARE_ACTIVE
LEG_PHASE_PREPARE_PASSIVE
AktivePhaseVorbei
StartBewegungNachHinten
ObenPositionErreicht, StopBewegungNachOben, StartBewegungNachVorne
PassivePhaseVorbei
HintenPosition NichtErreicht
VornPosition NichtErreicht
LEG_PHASE_ACTIVE
LEG_PHASE_PASSIVE
HintenPositionErreicht
VornPositionErreicht, StopBewegungNachVorne, StartBewegungNachUnten
UntenPosition NichtErreicht
LEG_PHASE_FINALIZE_ACTIVE
LEG_PHASE_FINALIZE_PASSIVE
StopBewegungNachHinten
UntenPositionErreicht, StopBewegungNachUnten

Fig. 6. State machine of the JC_Task Process.

In each state the actual value is compared to the value of the corresponding joint; the movement stops when the set point is reached and the new parameters for the next state are load:

- Direction of rotation - when loading a new state in eJointState - JC_STATE_ROTATING_START or JC_STATE_ROTATING_END;
- Turn Speed - when loading a valid (0 <= PWM <= 100) value in uJointRotationSpeed. Currently the speed is fixed to 100.

JC_JointStartRotating Function

This function is set by the input parameters specified joint with the previously loaded rotation and rotational speed in movement. Currently, the control is based on the principle Bang-Bang. This means that there are two discrete states of the control valves - on and off. But it is a controller using PWM (Pulse Width Modulation) provided. This allows the speed control.

Driving the PWM signal to 0% means that the corresponding solenoid valve is closed for the entire period; 100% value means that it is open for the entire period. A value of 50% means that half the time it is open and other half period it is closed.

The direction of rotation determines at which control valve (inflation or deflation) the PWM signal is created. Is the direction from the start to the end position the instruction is TRUE when the current value bigger or equal to the end value is

JC_JointTargetPositionReached Funktion

This Boolean function performs the comparison between the actual value and the value of the input parameters specified by the joint. It reads the actual value of the A/D module and compares it to the appropriate value (start or end position) in contrast to the current direction of movement. If, for example, the direction from the start is toward the final position, then this function turns to be TRUE only if the actual value is > or = of the end value. If the direction of the end is toward the start position this function turns TRUE only if the actual value is < or = of the end value.

JC_JointStopRotating function

This function has to stop the joint and is specified by the input parameters. The stopping is done by way of using of the PWM signal that promotes the appropriate one-or outlet valve to 0%.

4. PRACTICAL IMPLEMENTATION

For the development the evaluation board EVBR8C20-23 was used; the implementation is done in the "C" language. For the compilation is used the HEW (High-performance Embedded Workshop), 4:02 version by RENESAS. Since it is IDE, it contains all the necessary tools:

- Compiler - to compile the software;
- Left - to the left of the compiled software;
- Flash Tool - for the program to be "registered" in the microcontroller;
- Run-Time Debugger / Emulator - to test the program in real environment;

In Fig. 7 the experimental test rig is shownIt includes he mechanics-the leg prototype; hardware – microcontroller with interfaces. The algorithms had also been tested on the rig.

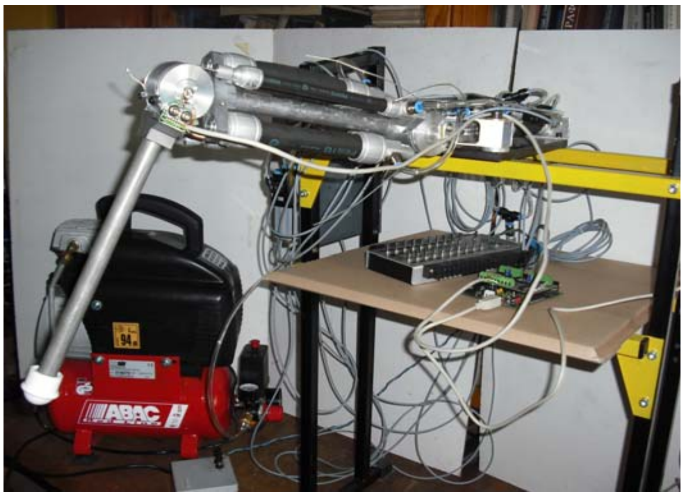

Fig. 7. General view of the leg with the control

5. CONCLUSION

Based on the implementation of control algorithms has been shown how using a dependency analysis a process model can be created that forms the basis for the implementation of a real-time system. Using the example of algorithms based on fixed-point arithmetic has been shown how the control algorithms can be implemented efficiently and predictably over time. In the present case a calendar-based scheduling model could be used. By means of case consideration and maturity provisions the real-time capability of the implementation could be shown.
Based on the achieved results the identified approaches can be applied for further research work. The control architecture is complex and only manageable with appropriate and identical design methods.
Currently, the control is based on the Bang-Bang principle.
Further experiments propose the implementation of the discussed PWM module depending on the pressure within the muscle and the joint's position. This would allow certain synchronization between all three joints in real-time.
Another advantage of the system is the open possibility for including reflexes.

ACKNOWLEDGEMENT

This work was supported by ВУ – ТН – 201/2006 Contract entitled "Research of a Modular Architecture for the Control of Mechatronic Elastic Multi-Link Devices"

REFERENCES

[1] S. Cordes, K. Berns, A Flexible Hardware A Architecture for the Adaptive Control of Mobile Robots, 3rd Symposium on Intelligent Robotic ystems '95, 1995.
[2] http://www.ai.mit.edu/projects/leglab /robots/ robots.html.
[3] Waldron, Kenneth J., Machines That Walk: The Adaptive Suspension Vehicle. The MIT Press, 1989.
[4] K. Berns, V. Kepplin, R. Miller, M. Schmalenbach: Six-Legged Robot Actuated by Fluidic Muscles. In Proc. of the 3th International Conference on Climbing and Walking Robots (CLAWAR), 2000.
[5] V. Kepplin, K. Berns (September 1999) Clawar 99: A concept for walking behavior in rough terrain. In Climbing and Walking Robots and the Support Technologies for Mobile Machines, pp. 509-516
[6] P. Gonzalez de Santos, E. Garcia, J. Estremera and A. Armada, SILO06: Design and configuration of a legged robot for humanitarian demining, Int. Workshop on Robots for Humanitarian Demining,2002