The 10th IFToMM International Symposium on Science of Mechanisms and Machines

SYROM'09

LABVIEW BASED CONTROL AND SIMULATION OF A CONSTRUCTION ROBOT

Th. BORANGIU[1] F. D. ANTON[1] S. ANTON[1]

Abstract: *The paper describes a multiprocessor control system for construction robots integrated in a SOA designed as a layered CAD – CAE information structure. The motion control system addresses both the robot's mobile carrier and the partially closed-loop arm of cylindrical type; their motion control is embedded in a multiprocessor Motion Control system for which programming library was developed. Experimental results concerning the motion control of the 5-d.o.f. robot arm are reported.*

Key words: *Robot motion control, bricklaying robot, multiprocessor systems, embedded real time systems, KB programming support.*

1. Introduction

The nowadays demand on the manufacturing process in the building industry is constantly rising to enable competition for quality, standardization and decrease of production costs. The standardization is directly related both to replication of construction procedures (bricklaying, windows placing, application of mortar on walls, finishing operations – painting, polishing) and to increasing working productivity. These two objectives, subject to constraints of efficient use of materials resources and employment of workers, can be reached by using robots in the most common, monotonous, effort demanding building operations – from which the most representative is bricklaying for wall elevation [4].

Rationalisation efforts in the construction industry are more and more associated with the attempt to create information systems used to automate the building processes. Up to now, automated solutions are developed however in each case for each special building process. The repetition of development errors and the increased training expenditure for the users cause high development costs, which make an economic application of the automation systems often impossible. One possible solution for the above problems is an open and modular control for construction robots based on embedded systems operating in a Service Oriented Architecture (SOA), which means:

- Using autonomous guided vehicles as robotic arm carriers moving to pre planned locations in the building site [6], [3].

[1] University *Politehnica* of Bucharest, Department of Automatic Control and Industrial Informatics.

- Using the robot arm in working locations to perform various types of operations: bricklaying, finishing, etc
- Using generic hardware modules (general purpose mathematic processors, motion controllers, universal motion interfaces, remote control and communication terminals) to build up embedded modular robot controllers adaptable to the working tasks and environment, external sensors, and material flow.
- Relying on open standard software (Linux-based real time operating system) and open solutions in software system design: the basic software system is created from a set of task-oriented modules and library functions such as: trajectory generator, motion tracking, end-effector set up, mobile platform navigator, inclination control, range finder and odometry localization, which are selected, attached and combined according to the set of particular services to be provided in a construction application, i.e. creating a Service Oriented Architecture (SOA), [4].
- Using a Knowledge Based Technology System (KBTS) to map technology specifications into production data and area robot work tasks and, by further access to a construction materials data base to automatically generate the parameters of application programs (base wall and brick stack locations, offsets in stacks, motion and grasping data (trajectory type, speed, pick offsets).

Research in control systems for construction robots is still seeking feasibility solutions and possibilities of large scale implementing, through integration with KBTS and databases on construction procedures and materials.

2. Robot functionality and operating modes for construction tasks

Fig. 1 presents the global robot system and its working environment with sensory control. The mechanical robot arm is a 5-d.o.f. partially closed-chain kinematic structure (with parallel linkages) of cylindrical type, carried by a 3-d.o.f. mobile wheeled platform. The resulting 8-d.o.f. mobile construction robot is able to move on quasi-horizontal prepared floors, and generates a 3.5 m-height workspace in 2D locations of the building site.

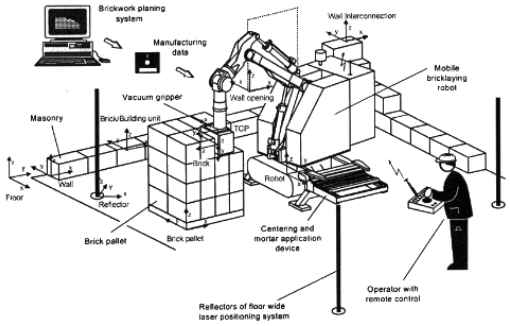

Fig. 1. *The construction robot and its environment*

The inclination of the mobile platform can be measured with inclinometers along two orthogonal symmetry axes and correspondingly corrected by means of a 4-d.o.f. mechanism electrically driven by four ac motors.

The mobile robot platform is a free-ranging (non-guided) wheeled vehicle, capable to avoid obstacles (e.g. brick pallets, walls) in a structured environment; its arm performs coordinated movements either in the Cartesian space or in the 5-dimension joint space automatically at program execution or under manual control. A PC-based operator console is used both as teach pendant for robot point learning and as robot terminal for execution of monitor commands, program

editing, debugging, execution start-up and monitoring.

The robot vehicle is a chassis with omni directional wheels. Two wheels (in diagonal locations) are driven by two asynchronous motors and generate respectively forward - backward displacements (when control signals are identical) and CW-CCW rotations (for opposite control signals); the remaining two are loose wheels – the angular displacement of one loose wheel being measured by an encoder. The third motor of the mobile robot platform drives turns simultaneously the four omni directional wheels via a common transmission mechanism.

In the most general situation, the displacement of the guided robot vehicle to a location within the building site is performed as a sequence of straight line displacements planned such that the contact with brick stacks of known location and dimension is avoided, whereas the shortest path is followed [6].

For each individual path segment, the four wheels of the robot's mobile platform are first turned along the direction towards the destination) point; then, the two actuated wheels drives the robot vehicle along this direction until the motion program is completed. Brick pallets may be placed everywhere in (known locations) the building site; the robot will turn around them to avoid collision and then resume its planned path towards the destination.

Due to positioning errors of the robot vehicle (wheel slipping, different friction coefficients of the four omni directional wheels), an accurate auto locating procedure is started once the platform stops. This procedure is executed with a laser scanner mounted on the mobile platform, and three reflectors disposed in known locations $x_i, y_i, 1 \le i \le 3$ relative to the reference frame (x_0, y_0) of the building site. A range finder receives the distances $d_i, 1 \le i \le 3$ and computes by triangulation the Cartesian location x_R, y_R of the robot vehicle. For a (prepared) horizontal floor the reflectors are at the same height and are scanned by the laser beam in a 360° planar sweep.

3. Modelling and embedded motion control of robot arm

3.1. Geometric models of the robotic arm

The robot arm mounted on the wheeled platform is a 5-d.o.f. closed-chain (parallelogram linkage CDEF) mechanical structure of cylindrical type having the joint variables vector $\mathbf{q} = [\theta_1 \quad d_2 \quad d_3 \quad \theta_4 \quad \theta_5]^T$. The point connecting the linkages CD and CF can be driven along two orthogonal (horizontal and vertical) directions, described by the joint variables d_2, d_3. This means that the motion of point G is amplified by $k = (L + l)/l \ge 1$ with respect to the motion of point C (Fig. 2). The end-effector wrist G is capable of two degrees of mobility: *pitch* (θ_4) and *roll* (θ_5).

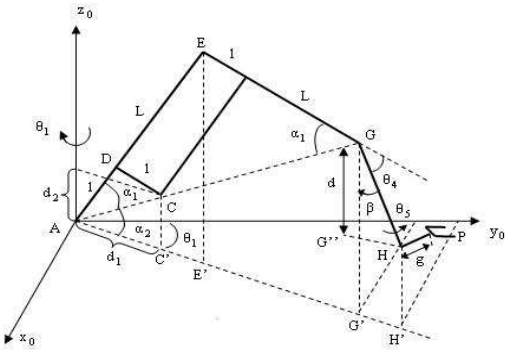

Fig. 2. *The closed-chain kinematic structure of the 5-d.o.f. cylindrical robot manipulator*

Due to its closed structure, the Denavit-Hartenberg formalism and Luh-Paul-Walker Algorithm for Local Coordinate System Allocation cannot be used to get

the implicit Direct Kinematics arm model, DKI. Due to this, a pure geometric approach was used to obtain the direct model: the location of an orthogonal frame is successively moved in important points of the arm structure, until the end-tip location P is reached. The DKI model is computed in two stages:

DK position model:

$$Pos(\theta_1, d_2, d_3, \theta_4, \theta_5) \rightarrow [X_H \; Y_H \quad Z_H]^T$$

$$\begin{cases} X_P = X_H + g\cos\beta\sin(\theta_1 - \gamma) \\ Y_P = Y_H + g\cos\beta\cos(\theta_1 - \gamma) \\ Z_P = Z_H + g\sin\beta\left(1 - 2\sin\left(\frac{\theta_5}{2}\right)\right) \end{cases}, \qquad (1)$$

where

$$X_H = (kd_3 + G'H')\sin\theta_1$$
$$Y_H = (kd_3 + G'H')\cos\theta_1 \;, \quad \frac{l+L}{l} = k$$
$$Z_H = kd_2 - GG''$$

$$D = \sqrt{\frac{4l^2 - (d_2^2 + d_3^2)}{d_2^2 + d_3^3}},$$

$$\alpha_1 = \arccos\frac{\sqrt{(d_2^2 + d_3^2)}}{2l}, \alpha_2 = \arccos\frac{d_3}{\sqrt{d_2^2 + d_3^2}}$$

$$G'H' = \frac{d}{2l}[D(d_2\cos\theta_4 - d_3\sin\theta_4) + d_3\cos\theta_4 + d_2\sin\theta_4]$$

$$GH'' = \frac{d}{2l}[D(-d_2\sin\theta_4 + d_3\cos\theta_4) - d_3\sin\theta_4 - d_2\cos\theta_4]$$

, $\theta_4 \rangle \alpha_1 - \alpha_2$

DK orientation model:

$$Orient(\theta_1, d_2, d_3, \theta_4, \theta_5) \rightarrow \mathbf{R}_5^0$$

The implicit homogenous orientation matrix $\mathbf{R}_5^0$ is derived below; its conversion to *rpy* minimal format is done at run time whenever relative transformations are used to plan arm motion.

$$\mathbf{R}_5^0 = \begin{bmatrix} s_1\sin\beta & -c_1c_5 + s_1\cos\beta s_5 & c_1s_5 + s_1\cos\beta c_5 & 0 \\ -c_1\sin\beta & -s_1c_5 - c_1\cos\beta s_5 & s_1s_5 - c_1\cos\beta c_5 & 0 \\ \cos\beta & -\sin\beta s_5 & -\sin\beta c_5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$s_1 = \sin\theta_1, c_1 = \cos\theta_1, s_5 = \sin\theta_5, c_5 = \cos\theta_5$$

$$\alpha_1 = ar\cos\frac{\sqrt{d_2^2 + d_3^2}}{2l}, \alpha_2 = ar\cos\frac{d_3}{\sqrt{d_2^2 + d_3^2}},$$

$$\beta = 90^\circ - \alpha_1 + \alpha_2 + \theta_4$$

The closed-loop Solution of the Inverse Kinematics problem is obtained using a combined algebraic and geometric computation [2]:

$$\mathbf{T}_5^0(\theta_1, d_2, d_3, \theta_4, \theta_5) = \begin{bmatrix} & \mathbf{R}_5^4 & & X_P \\ & & & Y_P \\ & & & Z_P \\ 0 & 0 & 0 & 1 \end{bmatrix} = \qquad (2)$$

$$= \begin{bmatrix} n_x^* & s_x^* & a_x^* & X_P^* \\ n_y^* & s_y^* & a_y^* & Y_P^* \\ n_z^* & s_z^* & a_z^* & Z_P^* \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} \sin\beta = \sqrt{n_x^{*2} + n_y^{*2}} \\ \cos\beta = n_z^* \end{cases} \Rightarrow$$

$$\Rightarrow \beta = a\tan 2\left(n_z^*, \sqrt{n_x^{*2} + n_y^{*2}}\right)$$

$$(\beta = \beta^*) \Rightarrow \theta_4 - \alpha_1 + \alpha_2 = \beta^* - 90^0$$

$$\cos\beta = n*$$

where

$$\theta_5 = a\tan 2\left(-\frac{a_z^*}{\sqrt{n_x^{*2} + n_y^{*2}}} - \frac{s_z^*}{\sqrt{n_x^{*2} + n_y^{*2}}}\right)$$

$$\theta_1 = a\tan 2\left(-\frac{n_z^* s_z^* (a_x^* + a_z^*)}{\sqrt{n_x^{*2} + n_y^{*2}}}, \frac{a_z^* a_x^* - s_z^* s_x^*}{\sqrt{n_x^{*2} + n_y^{*2}}}\right)$$

$$\theta_4 = \beta^* + \alpha_1 - \alpha_2 - 90^0$$

$$d_2 = \frac{2(l+L)\cos\alpha_1 \sin\alpha_2}{k},$$

$$d_3 = \frac{2(l+L)\cos\alpha_1 \cos\alpha_2}{k}$$

$$\alpha_1 = ar\cos\frac{Y_H^* - d\sin\beta^*\cos\theta_1}{2(l+L)\cos\alpha_2\cos\theta_1}$$

where

$$\alpha_2 = a\tan 2\left(\frac{X_H^* - d\sin\beta^*\sin\theta_1}{\sin\theta_1}, Z_H^* + d\cos\beta^*\right)$$

$$X_H = X_P^* - g\cos\beta^*\sin(\theta_1 - \chi^*) = X_H^*$$

$$Y_H = Y_P^* - g\cos\beta^*\cos(\theta_1 - \chi^*) = Y_H^*$$

$$Z_H = Z_P^* - g\sin\beta^*\sin\left[1 - \sin\frac{\theta_5}{2}\right] = Z_H^*$$

where $\chi = ar\sin\left[\frac{\sin\theta_5/2}{n_z^*}\right] = \chi^*$

Bricklaying tasks assume not only pick-and-place sequences, but also linear Cartesian motions whem mortar is applied to bricks held in the gripper.

Other construction tasks requiring the execution of linear Cartesian pats of the robot arm are: wall finishing, painting etc. In addition, all locations of interest in CAD files are specified in Cartesian coordinates and directly mapped to robot points as transformations relative to the world frame (reference frame attached to the building site) of the robot arm.

Consequently, the arm's trajectories are planified in the Cartesian space according to the algorithm:

$t = t_0$

loop:

Wait new control period (sample) ;

Update the end-effector's operational trajectory planner $\mathbf{TP}(t)$: computing the necessary position and velocity data $\{\mathbf{p}(t),\ \phi(t),\ \dot{\mathbf{p}}(t), \boldsymbol{\omega}(t)\}$ in Cartesian space at current time t ;

Compute the closed form inverse kinematics solution in joint space, $\mathrm{IK}[\mathbf{TP}(t)]$, correspondong to $\mathbf{TP}(t)$;

IF $t = t_{\text{final}}$ exit;

ELSE go to *loop*.

The method consists in generating support points by linear interpolation along the imposed operational path, and adding a desired speed profile. Then **TP** converts then the Cartesian support points in joint representations and feds them to the trajectory tracking unit. The **TP** has been implemented using the RMRC technique:

$\delta\mathbf{q}_c(t) = \mathbf{J}^{-1}(\mathbf{q}_c(t))\delta\mathbf{x}_c(t)$, where

$$\delta\mathbf{q}_c(t) = \mathbf{q}(t_{k+1}) - \mathbf{q}(t_k)$$

$$\delta\mathbf{x}_c(t) = I_{\mathrm{OS}}(\mathbf{x}_d(t_{k+1}) - \mathbf{dk}(\mathbf{q}(t_k))$$

One can observe that both the interpolation for support points and their conversion from Cartesian to joint representation are incremental, which reduces the computation time and increases the band width of the **TP** [1].

3.2 Embedded control of the construction robot

Motion tracking is performed with dual DSP-GPGA NI 735x motion controller with PID and feed forward laws; for linear Cartesian trajectories the contouring mode is used, according to which the computed joint-space support points are fed to a reference buffer as a set of relative values with respect to the initial point, followed by cubic spline interpolation between them subject to imposed speed and acceleration constraints.

The robot controller is designed as a ***multiprocessor structure***, including four interconnected processing areas (Fig. 3):

1. A real-time, embedded controller NI PXI 8196 acting as a mathematical processor for execution of application programs, motion planning and

trajectory generation t centralized level):

- Planning motion of the robot arm from trained or edited robot points and specified path types, and of the mobile vehicle by comparing site maps with path specifications (for collision avoidance with interior walls or brick stacks);
- Generating linear Cartesian trajectories of the arm and vehicle, coordinate joint trajectories of the arm, single-axis platform displacements to compensate deviations from horizontality;
- Locating the mobile vehicle by triangulation from range finder data;
- Communication with the operator's console for data and program transfer, diagnosis and production reports.

2. Two NI 735x motion controllers addressing three groups of coordinated axes (group 1: 5 axes of the cylindrical robot arm; group 2: 3 axes of the robot vehicle; 4 axes of the platform inclination mechanism):
   - Slave status for motion tracking in vector mode, i.e. coordinated motion in groups of axes configured by the PXI controller;
   - Loop control and S-type speed profile are embedded at motion control level, with PID- (position and speed) and feed forward control;
   - Arm and platform calibration using breakpoint modes (drive axis until occurrence of external event – limit switch);
   - Monitoring encoders, limit switches, external devices, protections and sequencing commands
2. 12 motor servo drive boards (power amplifiers) connected to the motion controllers through universal motion interfaces (UMI).
3. Operator console, acting both as robot terminal and teach pendant – a laptop wireless connected to the PXI real-time embedded central processor.

The system applications were developed differently: motion planning, kinematics model computation and trajectory generation are written as C++ routines, whereas the set of motion tracking routines: vector control (electronic gearing) – for coordinated joints motion, contouring – for traversing Cartesian support points along linear trajectories or

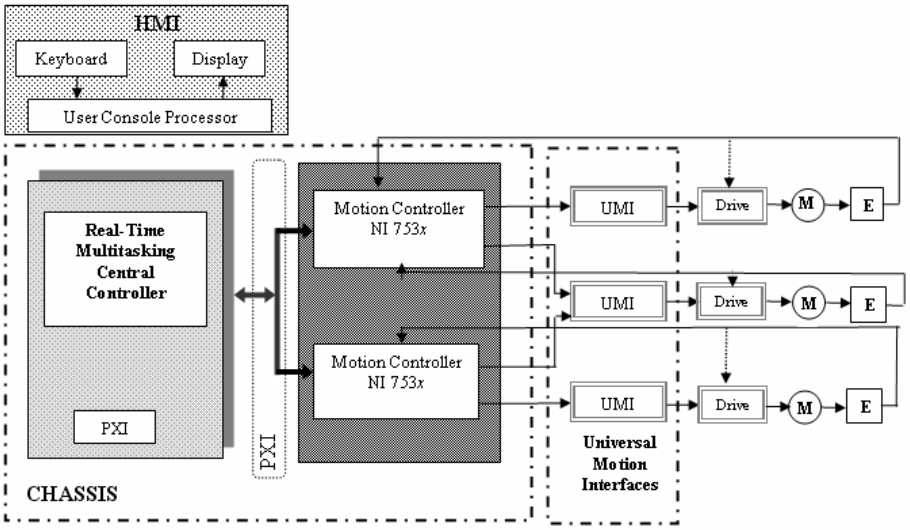

Fig. 3. *Industrial implementing of the multiprocessor robot controller*

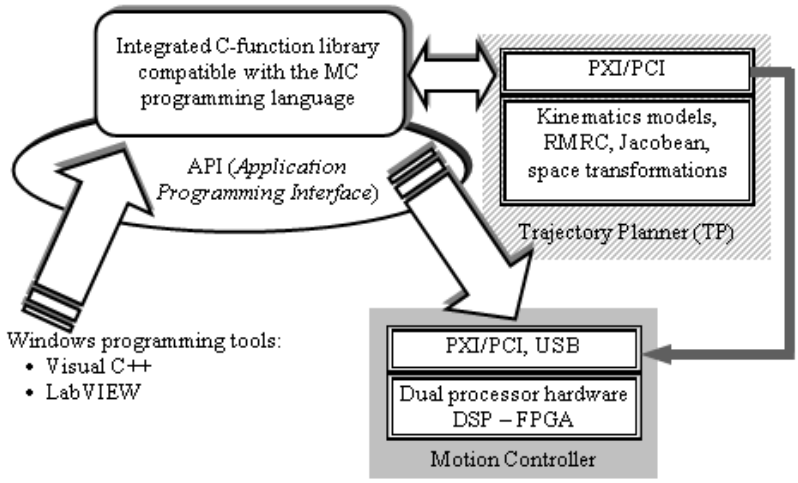

Fig. 4. *Creating, converting and transferring motion control application code for MC hardware units*

breakpoint motion – for calibration were developed as LabVIEW applications with the NI LabVIEW Real-Time module, and downloaded in the embedded real-time PXI central controller via Ethernet.

The embedded code is run on a real-time, multitasking operating system.

Fig. 4 shows the generic mechanism used to create, convert and transfer the application code to the hardware processing modules. This mechanism is based on integrated C support for developing multi-platforms of Motion Control type.

The system controlling the Windows application code programming tools (Visual C++ and LabVIEW) shares an *Application Programming Interface* – API with the Motion Control (MC) language.

The same software operations are in the MC programming language and in the function library, which simplifies program migration and reduces adaptation times for applications developed in different languages.

The Real Time Operating System of the robot controller is pre-emptive and re-entrant, having the capability to execute both on mono processor and multiprocessor architectures, and supporting Intel processors. The OS architecture is modular and consists of two main layers (Fig. 5):

- A *user module* (with limitations to system resource access), and
- A *kernel module*, having unrestricted access to the system's memory and external devices.

The **User module** contains subsystems capable to pass I/O requests to suitable software drivers of the kernel module, by using the I/O manager.

Two subsystems are on the user module layer:

- The *Environment* subsystem (E_Ss) executing applications written for various types of OS,
- The *Integral* subsystem (I_Ss) operates specific system functions for the E_Ss.

The Executive interfaces and all subsystems in the User module deal with I/O, object management, security and process management.

The hybrid **Kernel** is positioned between the HAL (Hardware Abstraction Level) and the Executive, in order to provide multiprocessor synchronization, planning and interrupt- and execution threads resolution, as well as trap processing and

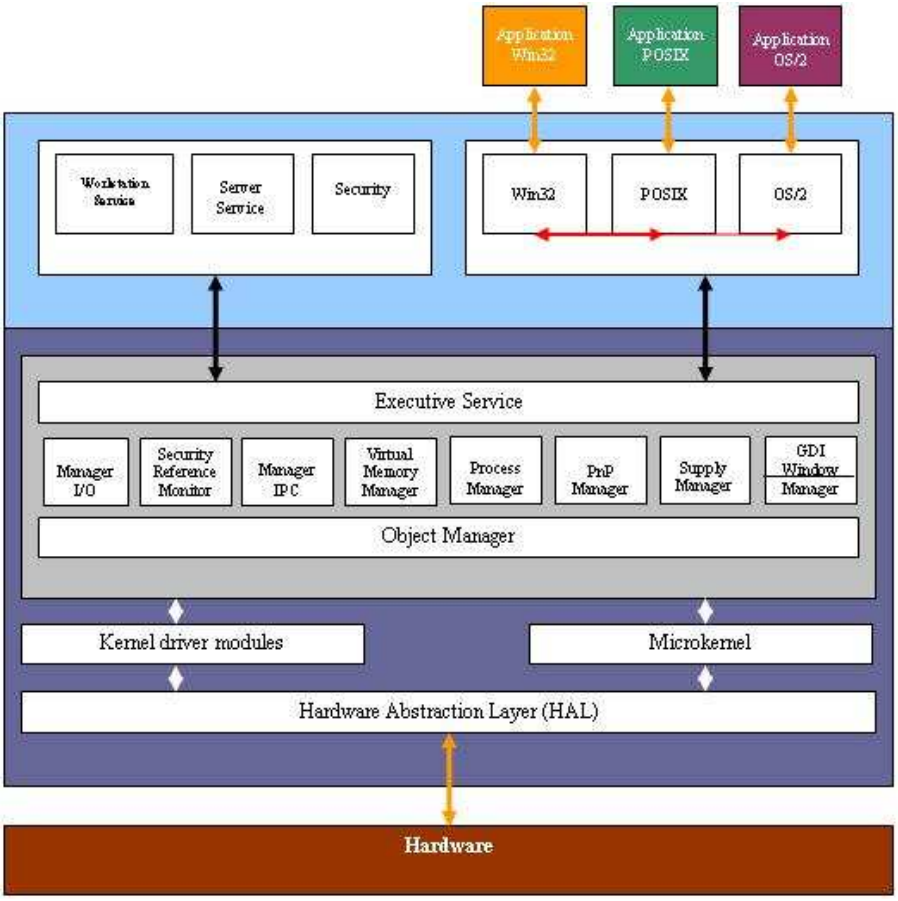

Fig. 5. *The architecture of the operating system (OS)*

exception solving. The Kernel module consists from executive services, in their turn composed from several modules performing specific tasks, kernel drivers, a Microkernel and HAL (*Hardware Abstraction Level*). HAL includes specific code for hardware controlling I/O interfaces, interrupt controllers and multiple processors [5].

The Microkernel is responsible for driver setup at the start moment. The drivers in the Kernel module are on three layers: high-, medium- and low level. Low level drivers are either inheritance drivers controlling directly a device, or busses for the PnP hardware. The PnP Manager is used to detect and initialize at system start up the Plug and Play devices. Low level drivers are either inheritance drivers controlling directly a device, or busses for the PnP hardware. The PnP Manager is used to detect and initialize at system start up the Plug and Play devices.

4. Experimental work. Conclusions

The reported research was carried out in the framework of a National CEEX Grant funded by the Ministry of Education and Research, and aims to develop a robot system and KBTS for automating construction tasks (Fig 6).

On the test has been observed that the positioning precision of the entire structure is 0.2 mm due to mechanical movement transmission, also the robot needs to move using small accelerations and trapezoidal speed profile, on high accelerations

/decelerations the mechanical structure starts to vibrate.

Fig. 6. *The construction robot*

Another problem which we try to solve is that the motors are commanded by two different types of frequency converters which give two different types of speed control (Fig. 7.) (the KEB converters seems to be more stable – green line, the other converter is LS-600).

In Fig. 7. the internal PID loops of the converters are disabled and is used only the PID control loop of the NI controllers.

Fig 8 shows simulation results of RMRC motion control algorithms for linear paths in the joint space of the robot arm. In the left side is represented the Position/Time characteristic, and in the right side is presented the Speed/Time characteristic (Red – KEB, White – LS600). Linear joint space paths are done by the trajectory generator and motion tracking processor according to the "electronic gear" algorithm below:

1. Start from two trained robot joint configurations $\theta_1, d_2, d_3, \theta_4, \theta_5$: initial p_i and final p_f .
2. Compute the rotation/translation path differences between each element of p_f and p_i, as the difference array $\theta_{1\Delta}, d_{2\Delta}, d_{3\Delta}, \theta_{4\Delta}, \theta_{5\Delta}$.
3. Determine the axis maximum path difference
$$\Delta q_i = \max_{1 \le i \le 5}(\theta_{1\Delta}, d_{2\Delta}, d_{3\Delta}, \theta_{4\Delta}, \theta_{5\Delta}), \quad (3)$$
corresponding to the master axis (*MA*) motion for the current trajectory segment.
4. Compute the gear ratio between the maximum path difference and each element of the difference array; the *gear array* $GA = \theta_{1\delta}, d_{2\delta}, d_{3\delta}, \theta_{4\delta}, \theta_{5\delta}$ results. This array will have one value

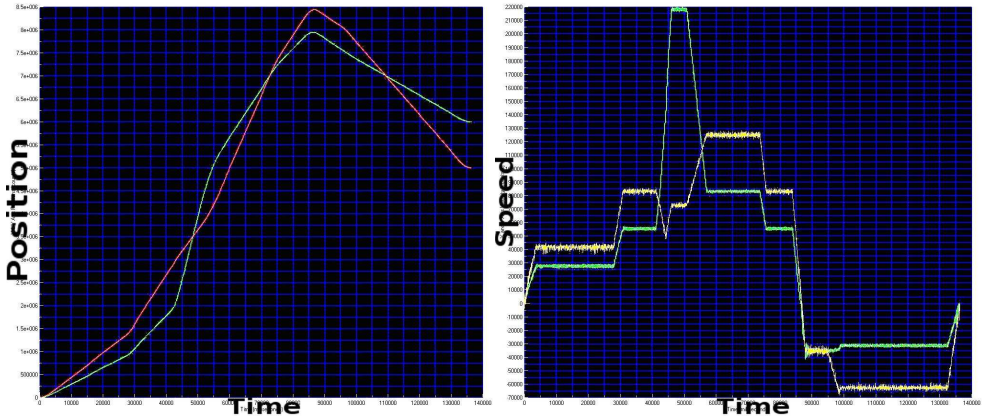

Fig. 7. *Blending moves for two axes (position – left, and speed-right)*

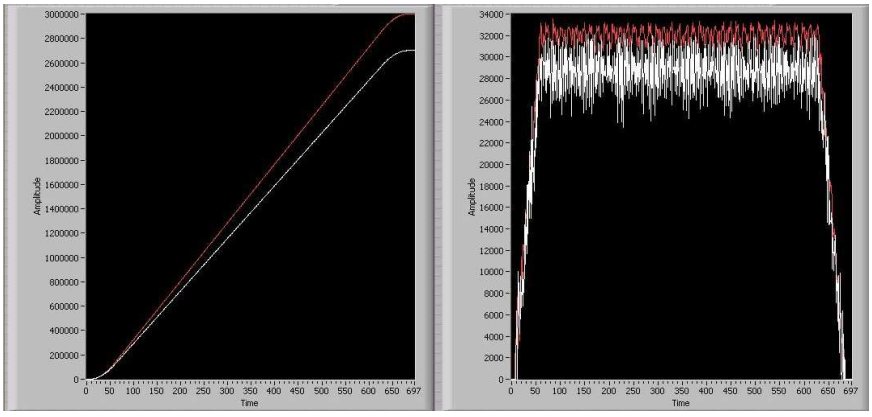

Fig. 8. *RMRC and electronic gear path control results*

equal to 1 and the other values in the range (-1, 1).

5. The positions of the five axes are modified incrementally by adding the values from the gear array multiplied by a value obtained by dividing the maximum difference to the resolution of the motor mounted on this axis (the other slave axes (*SA*) will have at least the same resolution). The value of *n* in the equation below incrementally changes axes position along the linear joint path:

$$MA\_pos = (MA\_pos - offset) + \Delta q_i / n$$
$$SA\_pos = (MA\_pos - offset) \cdot GA \quad (4)$$

6. Actions in step 5 are executed until the difference between p_i and p_f is small enough.

The parameters of the motion control algorithms were tuned to provide 1 mm worst error for Cartesian displacements and 0.2 mm worst positioning error.

In Fig. 8 we can see the big difference between the two converters; the LS-600 converter has problems in keeping the constant speed. The position is reached, but the speed has big variations which can lead to problems when the robot is loaded with high payload.

References

1. Borangiu, Th.: *Task-Driven Control of Robots Integrated in Intelligent Manufacturing Systems*, Proc. of 3rd IFAC Workshop Intell. Manufact. Syst. IMS'95, Oxford, Pergamon Press, 1995, p.79-89.
2. Borangiu, Th., Oltean E.: *Multi-Processor Design of Nonlinear Robust Motion Control for Rigid Robots*. Lecture Notes In Computer Science, 1798, Springer Verlag, 1999, p. 224-238.
3. Nehmzow, U.: *Mobile Robotics: A Practical Introduction*, Springer, London, 2003.
4. Pritschow, D., Kurz, J., Mc Cornack, S.E., Dalacker M.: *Practical Sensor Strategies for On-Site Positioning of a Mobile Bricklaying Robot*, Proc. of the 13th ISARC Symposium, Tokyo, Japan, 2006, p. 281-290.
5. Tourassis, V., Tourassis M., Ang M.: *Task Decoupling in Robot Manipulators*, Journal of Intell. and Robotics Syst., **14**, Kluwer, 1995, p. 283-302.
6. Zhao, Y.: *Kinematics, Dynamics & Control of Wheeled Mobile Robot*, Proc. IEEE Int. Conf. on Robotics and Automation, Nice, 1992, p. 91-96.