A SIMPLE AND EFFICIENT CONTROL ARCHITECTURE FOR WALKING ROBOTS
Application to AMRU5

Q. Bombled, O. Verlinden
Department of Theoretical Mechanics, Dynamics and Vibrations (TMDV), University of Mons
Place du Parc 20, 7000 Mons, Belgium

M. Bagein, P. Manneback
Department of Computer Science, University of Mons, Place du Parc 20, 7000 Mons, Belgium

Keywords: Distributed control, Real-time Linux real-time, PIC microcontrollers, UDP/IP communication, Walking robot.

Abstract: A decentralized control architecture has been developed on the walking machine AMRU5. The vehicle is actuated by 18 DC motors, which have to be highly synchronized to produce a smooth motion of the robot body. Each leg has 3 motors driven by a microcontroller. The six microcontrollers communicate with a PC running real-time Linux which manages the feet motion generation to produce the desired gait. The complete control chain has been developed using standard freely available C tools.

1 INTRODUCTION

Data flows between sensors and controllers is a crucial point in robotics which is solved by using terrain buses such as the ISA (Gonzalez de Santos et al., 2005) or the CAN bus (Berns et al., 1999). But, a general trends nowadays is the development of open source and free frameworks, which use the widely spread ethernet support. Several collaborative projects are still growing, taking advantages of the TCP/IP or UDP/IP protocols: see for example YARP (Metta et al., 2006), Player (Gerkey et al., 2001) or ROS (Quigley et al., 2009).

Following this trend, we propose here a very simple but efficient control architecture for walking machines actuated by DC motors. It is based on a classical master-slaves relationship: the master is a personal computer responsible for gait generation algorithm and motors position control. The slaves are PIC-based boards which simultaneously drives three motors. Moreover, the currents and voltages are measured and sent back to the master which saves them as a data logger. An interesting particularity of this work is that all the exploited devices used are cheap, and usable with free development tools.

Master and slaves are presented in Section 2. The data synchronization and some results are provided in Section 3. Conclusion is given in Section 4.

2 MASTER AND SLAVE BOARD

AMRU5 is a six-legged robot with hexagonal configuration. It weights about 34 kg and its outer diameter varies from 1.2 to 1.6 m. Each leg is actuated by 3 DC motors equipped with incremental encoders. The distributed control architecture is highligted in Figure 1. Each of the six embedded slave boards drives three joints. Moreover it measures the DC motors supply voltages, the rotor shafts positions and the motor currents.

Their synchronization is assumed by the master PC which computes in real-time the target position that each foot has to reach to produce the desired gait (detail about gait generation are out of the scope of this article). Master and slaves communicate with the UDP/IP protocol over 10 Mbit/s local network.

2.1 The Master

The master is a PC (3 GHz, 1.5 Go RAM) running Linux 2.6.31-11 preemptive real-time kernel. The

Bombled Q., Verlinden O., Bagein M. and Manneback P..
A SIMPLE AND EFFICIENT CONTROL ARCHITECTURE FOR WALKING ROBOTS - Application to AMRU5.
DOI: 10.5220/0003404802770280
In *Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2011), pages 277-280
ISBN: 978-989-8425-75-1

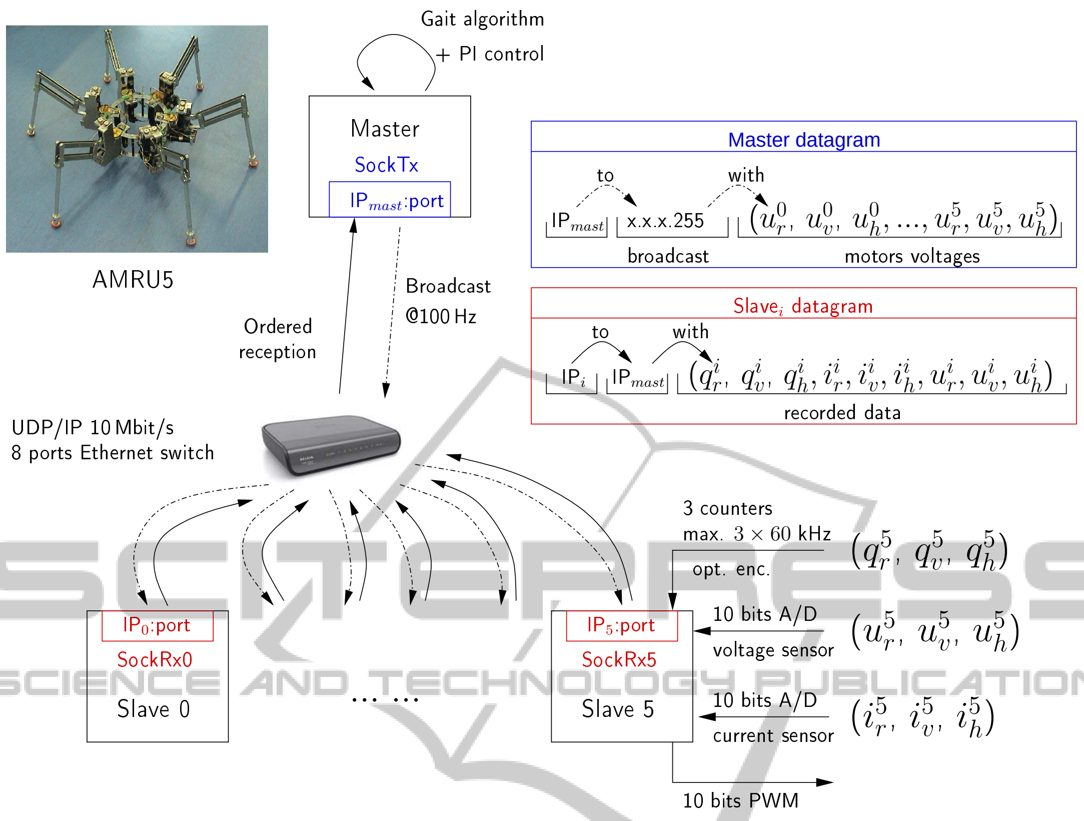

Figure 1: Global overview of the control architecture.

control loop is rated at 10 ms by means of the C-ANSI `select()` function (details are given in Section 3). When all the data from slaves are received, the gait algorithm computes the next 18 joint target positions. Each of them enters in a Proportional Integral (PI) control loop, calculating the new reference voltage to be applied to the corresponding motor.

One specificity of this architecture is that command is computed on the master, letting flexibility during the development phase because position controllers are implemented directly on the master instead of re-programming the six slaves. In return the communication must be very reliable, because the controllers require the updated position of each joint for control calculations.

2.2 The Slave

The three DC motors of each leg are commanded by a SBC65EC from Modtronix[1]. This single board computer with Ethernet capabilities is based on a PIC18F6627 running at up to 40 MHz. A TCP/IP open-source stack written for the Microchip C18 LITE compiler is freely delivered by the manufacturer. This stack naturally includes the UDP/IP protocol, and classical socket definitions similar to the C-ANSI standard.

Basically, the slave board has four tasks to perform within one control time slice: 1) manage local network communication, 2) collect continuously positions of the three motor shafts, 3) drive the three motors by applying the received command, 4) perform A/D conversions of three motor currents and the board voltage supply. The systematic data logging helps us to validate accurately a dynamic model established in (Bombled and Verlinden, 2009). Tasks 1) and 3) and 4) require less than 1.5 ms to be completed.

3 DATA FLOW

3.1 Data Synchronization

The UDP/IP Communication protocol has been chosen to meet real-time requirements. Data integrity over local network and low latency are reached, despite of the unsecured aspect of UDP, by dedicating the network to one and only one protocol detailed hereafter. And, even if risk of data losses exists, the

[1]http://www.modtronix.com

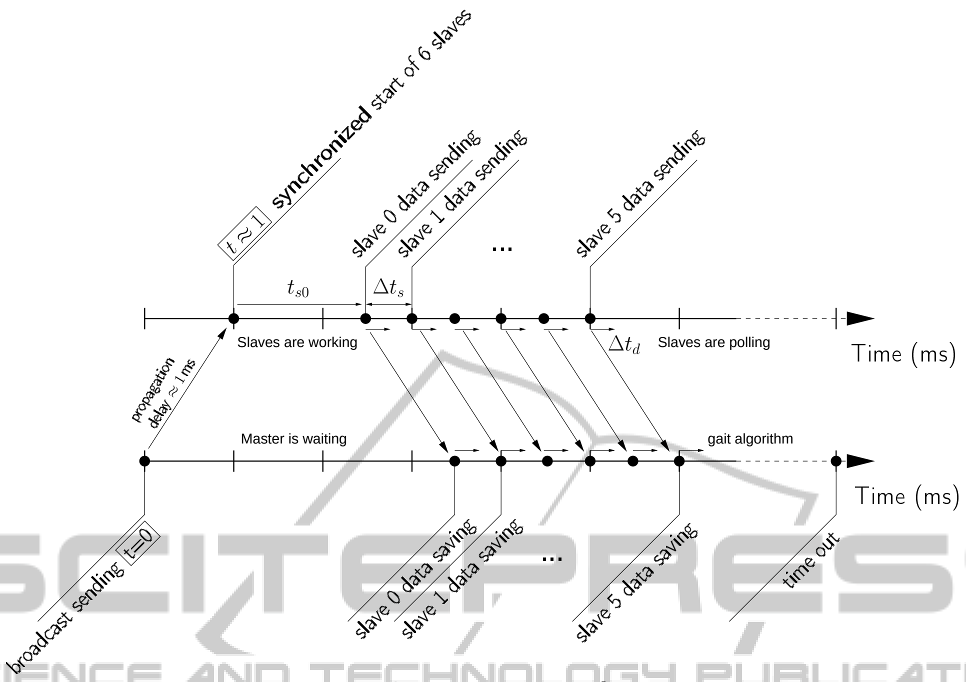

Figure 2: Communication time diagram.

update of the next control cycle should compensate the previous losses.

The master imposes the control rhythm by broadcasting every 10 ms a UDP global packet containing the 18 actuator commands. The broadcast ensures a simultaneous reception of this packet by all the slaves, who are ready to listen data on a specified port. Each slave collects its own three actuator commands from the broadcasted packet.

The master bases its emission-reception cycles on the `select()` C standard function. This blocking function returns in two events: when data are incoming on the socket or when a timeout occurs in the case of no network activity. The timeout is set to 10 ms at the beginning of the control cycle and is updated each time data are received on the master port. Each slave returns data to the master in order, at time $t_{s0} + i \cdot \Delta t_s$, where $i = 1...6$ is the number of the i^{th} slave. The time diagram of the communication process is shown in Figure 2.

The following pseudo-code describes the master cycle:

```
deadline = now() + delay; // delay of 10ms

while(robot_has_to_move())
{
  if(slave_cnt == 6)
  { // all slaves data received
    targets = compute_motor_command();
    slave_cnt = 0;
  }
  switch(select(socket,deadline-now()))
  {
    case -1: // network error
      return(error);
    case 0: // timeout is over
      broadcast(targets);
      deadline = deadline + delay;
      break;
    case 1:
      data_from_slave[slave_cnt] =
                               = receive(socket);
      slave_cnt = slave_cnt + 1;
      break;
  }
}
```

Note that the CPU load of each slave is a function of pulse frequency from the encoders, which increases with the motor velocity. Consequence is that the real slave data emission is slightly delayed. This variation is noted Δt_d in Figure 2.

3.2 Results

A tripod gait with a leg cycle period of 15 s has been implemented on AMRU5. Figure 3 is a plot representing the arrival time on the master, of the data sent by the six slaves (numbered ⓪→⑤), inside the 10 ms of control time slice, for a gait during 30 s. Time 0 on the Y-axis corresponds to the emission of the broadcast packet. In this example, t_{s0} and Δt_s have been fixed to 1.5 and 0.7 ms respectively.

First data incomes at 3.5 ms on the master, because

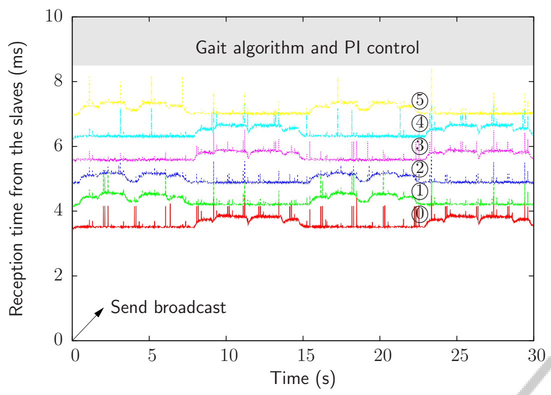

Figure 3: Reception times of data coming from slaves.

a propagation delay of 1 ms has been recorded on the Ethernet network (Figure 2). The network propagation delay between master and slaves mainly comes from the switch.

The varying load on the slave CPU is visible on the receiving times. But in any case, the reception is well ordered, and the time remaining for the gait algorithm and control computations is sufficient (8.5 - 10 ms).

4 CONCLUSIONS

A distributed control architecture has been developed for a hexapod robot. The slave element controls a leg at position level: the PIC-based board allows easy C programming of the control tasks, and is provided with an open-source TCP/IP stack compatible with the C18 Microchip compiler. The master is a PC running real-time Linux kernel on which an application has been developed in C-ANSI for leg motion generation and data synchronization. The resulting control architecture is very flexible and has several advantages:

- The slaves have a generic program: they only drive three motors and collect real-world data. This avoids several re-programmings during the development as it had been the case with a position controller directly implemented on them. They just have a specific identification number, which determines their IP address and data they have to read from the master global packet.
- Any kind of controller and/or gait algorithm could be implemented on the master, provided that the computation is fast enough to stay inside the control time slice. For example, see (Bombled and Verlinden, 2010) for a ground detection algorithm from motors currents sensing.
- Communication hardware is made from widely spread and inexpensive on the shelf materials, namely: a network switch, an Ethernet adapter, and microcontroller boards.

ACKNOWLEDGEMENTS

Many thanks to Prof. Y. Baudoin from the Royal Military Academy of Belgium for the lending of AMRU5. Authors are also grateful to the technical staff of the TMDV Department: R. Berton and K. Nis.

REFERENCES

Berns, K., Ilg, W., Deck, M., Albiez, J., and Dillman, R. (1999). Mechanical construction and computer architecture of the four-legged walking machine bisam. *IEEE/ASME Transactions on Mechatronics*, 4(1):32–38.

Bombled, Q. and Verlinden, O. (2009). Dynamic model, gait generation and control of the amru5 hexapod robot. In *Mobile Robotics : Solution and Challenges*, pages 513–521, Istanbul, Turkey. CLAWAR 12th International Conference.

Bombled, Q. and Verlinden, O. (2010). Current sensing in a six-legged robot. In *IUTAM Symposium on Dynamics Modeling and Interaction Control in Virtual and Real Environments*, Budapest, Hungary.

Gerkey, B., Vaughan, R., Stoy, K., Howard, A., Sukhatme, G., and Mataric, M. (2001). Most valuable player: A robot device server for distributed control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1226–1231, Wailea, Hawaii.

Gonzalez de Santos, P., Garcia, E., Estremera, J., and Armada, M. (2005). DYLEMA : Using walking robots for landmine detection and location. *International Journal of Systems Science*, 36(9).

Metta, G., Fitzpatrick, L., Natale, P., and Natale, L. (2006). Yarp: yet another robot platform. *International Journal on Advanced Robotics Systems*, 3(1):43–48.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *Proc. Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*, Kobe, Japan.