Proceedings of the 5th IFAC International Symposium on Intelligent Components and Instruments for Control Applications (SICICA 2003), Aveiro, Portugal, July 9-11, pp. 245-250, 2003.

A SIX-LEGGED ROBOT: REAL-TIME ISSUES AND ARCHITECTURE

Mauro Marinoni, Alessio Carlini, Giorgio Buttazzo

University of Pavia, Italy
Email: the.nino@libero.it, lbrgc@tin.it, buttazzo@unipv.it

Abstract: Mobile robot systems often operate under real-time constraints imposed by the interactions with the world in which they act. Such timing constraints are typically assigned to the various software activities (acquisition, control and actuation tasks) that run on the robot controller, and need to be enforced by the operating system that support the application. To reduce weight and power consumption, however, battery-operated robots are usually controlled by small microcontrollers with little computational power and memory capacity. As a consequence, predictability as well as high efficiency is required from the real-time kernel to enforce timing constraints on application tasks.
In this paper we describe a robot control architecture specifically developed for supporting real-time applications in small mobile robots with limited resources. The proposed real-time system has been used to control the behavior of a six-legged walking robot, where the motion of each leg has to be synchronized with the other based on the walking parameters.

Keywords: Real-time kernel, scheduling, Embedded system, Control architecture

1. INTRODUCTION

The advancement of mechanical and electronical components allows the development of robot vehicles and walking machines that are smaller and smaller in size. When such robots must exhibit a certain level of autonomy, the major problem is to design an embedded computer architecture which is small enough to fit on the robot structure, and powerful enough to execute all the robot computational activities needed for achieving the desired level of autonomy. In order to decrease the weight of the batteries, the robot architecture must be designed very carefully to limit energy consumption. As a consequence, such small robots are usually controlled by tiny microprocessors that have limited memory and computational power.

Mobile robot systems, however, often operate under dynamic and unknown environments, and must promptly react to external events in order to perform their goals. Such an interaction with the world causes the robot activities to be characterized by timing constraints, that must be respected to achieve the expected robot behavior. Typical computational activities performed by the robot involve sensory acquisition, data processing and motor control. At a higher level of control, typical tasks include obstacle avoidance, path planning and reactive actions (such as feedback-based operations and tracking behaviors). Most of these tasks are periodic, that is, must be cyclically executed with specific rates, whose value depends on the characteristics of the environment and on the required robot performance (e.g., running speed, reaction times, etc.). Some other activities may be aperiodic, that is can be activated at unknown times on the occurrence of specific events, such as a signal change in a sensor.

To perform an accurate schedulability analysis of the application, a periodic task τ_i is defined as an infinite sequence of jobs $\tau_{i,k}$ ($k = 1, 2, \ldots$), where the first job

⋆ This work has been partially supported by grants from the European Union, under contract IST-2001-34820, and from the Italian National Research Council (CR), under contract CNRC002FE3.

$\tau_{i,1}$ is released at time $r_{i,1} = \Phi_i$ (the task phase) and the generic k^{th} job $\tau_{i,k}$ is released at time $r_{i,k} = \Phi_i + (k-1)\,T_i$, where T_i is the task period. Each job is then characterized by an execution time C_i (typically estimated in worst-case conditions), a relative deadline D_i and an absolute deadline $d_{i,k} = r_{i,k} + D_i$. The ratio $U_i = C_i/T_i$ is called the *utilization factor* of task τ_i and represents the fraction of processor time used by that task. The value

$$U_p = \sum_{i=1}^{n} U_i$$

is called the *total processor utilization factor* and represents the fraction of processor time used by the periodic task set. Clearly, if $U_p > 1$ no feasible schedule exists for the task set.

In order to achieve stability and guarantee a desired performance, timing constraints need to be enforced by the operating system that supports the application. In particular, the operating system should guarantee that all periodic tasks are activated according to their specified periods and executed within their deadlines. In addition, tasks may have different importance, hence a proper priority level need be assigned to them for scheduling purposes.

In 1973, Liu and Layland (Liu, 1973) analyzed the properties of two basic priority assignment rules: the Rate Monotonic (RM) algorithm (according to which priorities are inversely proportional to task periods) and the Earliest Deadline First (EDF) algorithm (according to which priorities are inversely proportional to absolute deadlines). Notice that RM is a static priority algorithm, since periods are usually fixed parameters, whereas EDF is a dynamic algorithm, since absolute deadlines change from a job to the other.

The main result found by Liu and Layland is that, if there are not resource constraints, nor precedence relations, a set of n periodic tasks is schedulable by the RM algorithm if

$$\sum_{i=1}^{n} U_i \leq n\,(2^{1/n} - 1). \tag{1}$$

As n increases, the right term of the feasibility condition decreases and it can be shown that:

$$\lim_{n\to\infty} n\,(2^{1/n} - 1) = \ln 2 \simeq 0.69.$$

Under the EDF algorithm, it has been shown that a set of n periodic tasks is schedulable if and only if

$$\sum_{i=1}^{n} U_i \leq 1. \tag{2}$$

Another important result proved by Liu and Layland is that the RM priority assignment is optimal among all fixed priority rules, in the sense that, if a task set cannot be scheduled by RM, then it cannot be scheduled by any other fixed priority assignment algorithm. In (Dertouzos, 1974), Dertouzos showed that EDF is optimal among all priority assignments. Unfortunately, however, EDF is more difficult to implement, since all commercial real-time kernels do not support absolute deadlines and dynamic priorities.

The consequence of these results is that, if periodic tasks are assigned fixed priorities, the processor cannot be fully utilized to guarantee feasibility of the schedule. On the other hand, dynamic priority schemes like EDF, that would fully utilize the processor, cannot be easily implemented on top of commercial kernels that do not support absolute timing constraints. Such a limitation is even more serious when applications need to be realized on small 8-bit microcontrollers.

To overcome the problem illustrated above, we developed a small real-time kernel for the Motorola 68HC11 microcontroller, capable of handling absolute timing constraints and using EDF as a basic scheduling mechanism. The system has been used to develop a control application, in which a six-legged robot is programmed to walk in different modes and directions, and to react to a limited number of events.

This paper describes our experience in developing the robot system and discusses some details of the robot architecture components. In particular, Section 2 presents an overview of the robot system architecture, Section 3 illustrates the characteristics of the real-time kernel, Section 4 describes the control algorithm used to generate the motor actions, and Section 5 states our conclusions and future work.

2. SYSTEM ARCHITECTURE

The robot developed in this work is a walking machine endowed with six independent legs, each having two degrees of freedom. With respect to a four-legged robot, a six-legged machine is more stable during walking, and leg motion generation can be performed without the additional constraint of controlling the center of mass of the robot. This allowed us to simplify the development of the control algorithms and to test several walking modes. The robot system consists of three basic components, as illustrated in Figure 1: the electromechanical structure, the motor controller, and the walking controller.

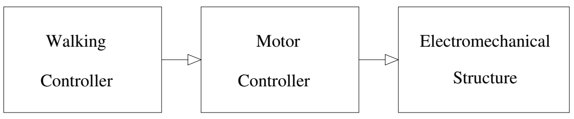

Fig. 1. Block diagram of the robot system.

2.1 The robot

The structure of the robot consists of an electromechanical skeleton, which supports the 12 servomotors that actuate the six legs. Each motor is a Hitec HS-645MG and includes an internal position control loop that allows the user to specify angular positions through a PWM input signal. This feature simplifies the external circuitry and avoids sending feedback signals to the motor control unit. The internal feedback loop imposes an angular velocity of 250 degrees per second and the motor is able to generate a maximum torque of 9.6 kg/cm. The two motors that actuate a leg are connected to a board that provides the motors with power supply and the input signals coming from the control layer.

2.2 Motor controller

The motor control layer is realized using a Microchip 16F876 Programmable Interrupt Controller (PIC) (Microchip, 2001). The main function of the PIC is to receive the joint reference positions from the walking controller and generate the PWM signals for driving the motors. The PWM signal generator is interrupt driven and can drive up to 8 motors. To do so, Interrupt 0 is periodically activated with a rate equal to eight times the rate used by servomotors. Cyclically, each of the eight output signals is set high, and the PIC timer is set to the value equal to the pulse duration corresponding to the desired angular position. Hence, when activated, the routine serving Interrupt 1 only needs to reset all the outputs. A sample output sequence generated by the PIC is shown in Figure 2.

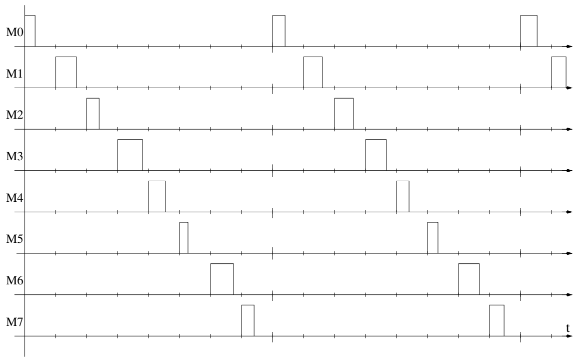

Fig. 2. Example of pulse sequence generated by the PIC.

Input set points are received from the walking layer via a standard RS232 serial line, through the USART integrated in the PIC, which allows a transfer rate up to 56 Kbaud. This chip can also work in a mode that allows to connect several PICs to the same serial line, so extending the number of output PWM signals by a multiple of eight. In this mode, an address must be sent in the first byte of the input sequence, and the chip enables itself only if recognize that address. The reception routine and the corresponding interpreter of data run in background. A check on the correct reception is performed by sending each data back to the sender, which performs the actual comparison with the original data. In addition to the reference values for the motors, from the same serial line, the PIC can also receive a number of commands to disable a motor, change the baud rate, or modify the period of the PWM pulses.

The use of a standard RS232 serial line has the advantage of facilitating the replacement of the walking layer for simplifying testing. In our system, the walking control has been implemented both on a Motorola 68HC11 microcontroller and on a PC, which simplifies the development of the control algorithms and allows to display all system variables in a suitable graphical form. For the moment, the walking controller and the PIC are connected through a serial cable, but this can easily be replaced by a wireless device. In fact, the PIC board provides two jumpers to redirect the reception and transmission lines to an auxiliary communication peripheral, rather than to the Maxim MAX232 chip, that converts TTL signals into 12V signals. A diagram of the electrical layout of the PIC board is illustrated in Figure.

At present, a single PIC is being used to drive 8 motors. This is possible by coupling the movements of two pair of legs (two for each side). In this solution, legs 1 and 5 move synchronously and are driven by the same PWM signal. The same is true for legs 2 and 6. The extension to two PICs can simply be realized by using the address recognition modality described above.

2.3 Walking Controller

In the initial phase of development, in order to simplify testing, the walking controller has been implemented on a Personal Computer, using the Shark real-time kernel (Gai, 2001). Then, the software has been ported on the Motorola 68HC11 microcontroller (Motorola, 2002), running the McuOS kernel (Carlini, 2002), described in the next section. The microcontroller is mounted on a single board computer (Grifo GPC114), incorporating 64 Kbyte of memory (32 Kbye of EPROM and 32 Kbyte of RAM), an RS-232 serial line, a 20 lines standard connector and a 26 lines Abaco I/O bus. The processor is a 8-bit CPU, with a 2 MHz clock, capable of addressing 64 Kbytes of memory. It includes 16 digital I/O lines, an 8-bit A/D converter multiplexed on 8 lines, a 16 bit timer counter and a standard RS232 serial communication line. The application code has been written in C language, using a GCC compatible cross-compiler, with limited optimization capabilities. For this reason, some time critical portions of code have been written in assembly language.

3. THE REAL-TIME KERNEL

To achieve predictability and efficiency on control activities, the walking controller runs a real-time kernel, McuOS, developed at the University of Pavia for supporting small embedded applications on the M68HC11 microcontroller. The kernel has a modular structure and has been designed to be easily portable on different hardware platforms. McuOS supports explicit timing constraints, such periods and deadlines, and it is able to handle periodic and aperiodic tasks with different criticality. In particular, the scheduling mechanism is based on the Earliest Deadline First (EDF) algorithm (Liu, 1973), and soft tasks are handled through the Constant Bandwidth Server (CBS) (Abeni, 1998), an efficient service mechanism that allows achieving resource reservation and temporal isolation among tasks. Using the CBS, each task τ_i is assigned a predefined processor bandwidth $U_i = C_i/T_i$, and the server guarantees that a budget C_i is reserved for task execution in every interval of T_i units of time. The isolation property of the CBS prevents any temporal interference among concurrent tasks, in the sense that a task with reserved bandwidth U_i runs as it was executing alone on a slower processor of speed U_i times the actual speed.

Besides the temporal isolation property provided by the CBS, another peculiar feature of the McuOS kernel is the high resolution time reference mechanism, implemented using a circular timer, according to the Implicit Circular Timer Overflow Handler (ICTOH) algorithm (Carlini, 2003). This novel method provides a simple and extremely efficient mechanism for achieving an infinite lifetime[1] and a high resolution, still using 8-bits variables for time representation. For this reason, the ICTOH algorithm is also characterized by a small runtime overhead and memory requirements when comparing two timing events.

The real-time scheduler implemented in the McuOS kernel also provides a resource reclaiming mechanism, which exploits the unused execution time left by tasks that may complete earlier than expected. This feature significantly improves the average response times of tasks with variable computational requirements and also allows to reduce some problem deriving from dealing with relatively long periods and deadlines.

Whenever the processor becomes idle, the reclaiming mechanism is invoked for resetting all relative deadlines (that may have been postponed by the CBS to enforce isolation) to their nominal value. The great advantages of this method are its simplicity and its low runtime overhead, which make it suitable for small embedded systems. A disadvantage of the proposed reclaiming method is that it is not easy to exploit the unused CBS budget. To overcome this problem, another reclaiming mechanism, denoted as the PASS method, allows to transfer a residual budget to the served task with the earliest deadline. It is worth noticing that the PASS budget sharing mechanism is better suited for tasks with periodic or sporadic arrivals. In fact, if a new aperiodic request arrives just after the previous one is ended, the residual budget (if any) may be transferred to another task, and the new request has to wait for the next recharge, so reducing aperiodic responsiveness.

In the real-time literature, other solutions have been proposed to reclaim the spare budget left by CBS tasks. They are very effective in terms of performance, but too costly to be implemented in a small microcontroller. For example, the GRUB algorithm (Lipari, 2000) makes intensive use of floating point variables, that are not supported in small microprocessors or are heavy to manipulate. The CASH algorithm (Caccamo, 2000) is based on a queue of spare capacities that needs to be managed in addition to the other system queues, thus increasing the runtime overhead and memory requirements.

The CBS mechanism implemememented in the McuOS kernel extends the original CBS algorithm (Abeni, 1998), by making it capable of handling tasks that may share mutually exclusive resources in a predictable fashion. This is done by means of a simplified implementation of the Stack Resource Policy (Baker, 1991), which is able to prevent unbounded priority inversion and limit the maximum blocking time during access to mutually exclusive resources. Although highly predictable, such a method introduces some extra runtime overhead, that must be taken into account when verifying schedulability conditions. In a small embedded microcontroller as the M68HC11, such an approach can only be justified when tasks have a few and large critical sections. For this reason, task communication can also be handled through classical mailboxes.

To simplify the development of real-time applications, a tool is provided for helping the user in compiling the kernel. Such a tool generates the application files from a pair of files provided by the user. This allows developing real-time applications without knowing the implementation details of McuOs. The first file specifies the source code, in C language, of the application tasks, so it basically contains a set of functions with a set of variable definitions. The second file specifies all parameters relative to tasks, CBS servers, and resources. It is written in XML and can also include a pair of user-defined functions for initializing the microcontroller and the interrupt handlers. The contain of a sample configuration file is illustrated in Figure 3.

4. APPLICATION TASKS

The walking controller includes two important tasks: the leg motion generator, which computes the refer-

[1] The lifetime is the maximum time that can be handled by the system.

```
<Task>
    <FunctionName>Task0</FunctionName>
    <StackSize>1000</StackSize>
    <TaskType>SOFT_LIGHT</TaskType>
    <ExecuteTime>1000</ExecuteTime>
</Task>
<CBS>
    <Period>30000</Period>
    <ManageTask>Task0</ManageTask>
</CBS>
```

Fig. 3. Sample XML specification of task parameters.

ence angles for each motor, and the transmission task, which sends the actual data to the PIC.

The algorithm used to generate leg motion modulates a reference walking step through a number of parameters. An important parameter is the maximum angle that each leg covers in the horizontal plane during its motion (see Figure 4a). A differential change in such angles for the left and right legs causes the robot to turn. Another parameter that can be tuned is the maximum angle that each leg covers in the vertical plane (see Figure 4b). This value depends on the type of surface on which the robot walks: small values are sufficient for walking on smooth surfaces, whereas higher values are needed in the presence of obstacles or protruding regions.

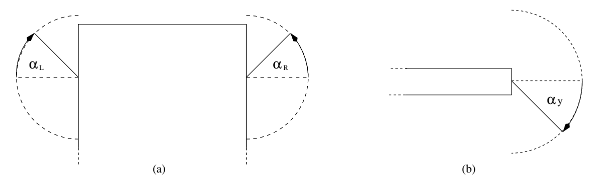

Fig. 4. Leg angular rotations: (a) top view; (b) side view.

To guarantee the equilibrium of the robot during walking, the algorithm always keeps three legs in touch with the ground, in such a way that the center of mass of the robot always falls in the triangle defined by the touching points.

The specific position set points for the motors involved in the horizontal motion are generated by sampling two cosine functions with opposite phases. Similarly, a pair of sine functions with opposite phases was initially used for the vertical leg motion. However, this solution has been modified since it was causing the robot to have a significant roll while walking. The specific shape of the waveform depends on many factors, including equilibrium requirements, speed, and maximum allowed roll.

Figure 5 illustrates the four functions that are presently used for generating the vertical leg motion.

4.1 Timing constraint derivation

In the robot application we described in this paper, task timing constraints have to be properly derived in

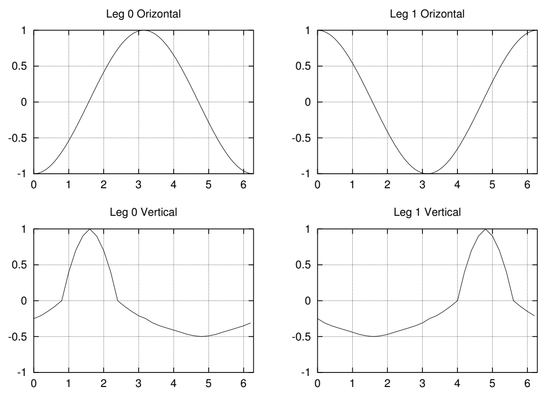

Fig. 5. Functions used for the vertical leg motion.

order to achieve a desired performance. In particular, the deadline to be assigned to the communication task depends on several factors, such as the transmission speed of the serial line, the echoing time, the execution time for error checking and the time for error recovery. This deadline, in turn, determines the minimum period that can be assigned to the leg motion generator task. In fact, for each walking step, this task has to perform eight transmissions to send the new set points to the motors. In the interval between the transmission of two set points, the communication task must be able to send all data related to the previous step, otherwise the message queue saturates in a short time. If T_{walk} is the period of the leg motion generator task and T_{send} is the period of the communication task, then we must have that

$$T_{walk} > 8T_{send}.$$

At present, $T_{walk} = 18$ ms, whereas $T_{send} = 2$ ms.

An another important timing parameter is the period used for sampling the cosine functions used for leg motion generation. In fact, the duration of a complete walking step is equal to the period of the leg motion generation task times the number of samples used for describing the cosine functions. Considering the speed limit of the motors, such a duration cannot be less than the time needed by a servomotor to execute the corresponding rotation. Hence,

$$T_{step} = N_{samples} T_{walk} > \frac{2\alpha_{max}}{V_{servo}}. \qquad (3)$$

where $N_{samples}$ is the number of points used to sample the cosine functions, α_{max} is the cosine amplitude, and V_{servo} is the velocity of the servomotor. Currently, the cosine functions are set to cover and maximum angle $\alpha_{max} = 75$ degrees, and are sampled with 64 points. As a consequence, the time for performing a full walking step is $T_{step} = 1,152$ seconds.

Turns during walking can be performed by changing the amplitude, α_R and α_L, of the left and right angle. Assuming that $\alpha_R = \alpha_L$ (i.e., straight line motion) we can compute the average walking speed of the robot as the ratio of the space Δ_x covered in a full walking

L_0	105 mm
α_{y0}	37,5
$\Delta_x(\alpha_{max})$	322 mm
$v_(\alpha_{max})$	0,28 mm

Table 1. Values of the robot parameters.

step and the time T_{Step} required for it. The space Δ_x is given by

$$\Delta_x = 4L_0 cos(\alpha_{y0}) sin(\alpha_{max})$$

where L_0 is the leg length and α_{y0} is the amplitude of the vertical cosine function (see Figure 2). Hence,

$$v = \frac{\Delta_x}{T_{step}}.$$

Using the values reported in Table 1, the average speed of the hexapod is about 280 mm/s.

5. CONCLUSIONS

In this paper we presented a real-time architecture for supporting the development of robotic control applications based on PWM-driven servomotors. As a sample application, we described a six-legged walking machine endowed with 12 position controlled servomotors. The PWM signals for the servomotors are generated through a PIC subsystem, connected to the host computer through a standard RS232 serial line. The host computer, running the higher level control algorithms, has been implemented both using a PC and a Motorola 68HC11 microcontroller. In both cases, the application tasks have been handled by a real-time operating system to achieve predictable behavior and guarantee of task timing constraints. A novel real-time kernel (McuOS) has been used for the Motorola 68HC11 microcontroller, providing efficient internal mechanisms for time management, task scheduling, temporal isolation, resource sharing, and resource reclaiming.

Experimental results performed on the robot system showed the effectiveness of the proposed approach and its potential use for other similar systems.

As a future work, we plan to endow the robot with a set of sensors for achieving some autonomous behavior, like proximity ultrasound transducers, a videocamera, whisker-like contact switches, and light sensors. In addition, we plan to monitor leg torques by reading the current drained by the motor drivers. This is useful for identifying possible situations that may prevent the motion of some leg, due for example to big obstacles along the path. Finally, we would like to redesign the locomotion system to give each leg three degrees of freedom. This would allow to implement more sophisticated walking patterns, including lateral translation.

REFERENCES

Abeni, L. and Buttazzo, G.C. (1998). Integrating Multimedia Applications in Hard Real-Time Systems. *Proc. of the IEEE Real-Time Systems Symposium*, Madrid, Spain.

Baker, T.P. (1991). Stack-Based Scheduling of Real-Time Processes, The Journal of Real-Time Systems 3(1), pp. 76-100.

Caccamo, M. and Buttazzo, G.C. and Sha, L. (2000). Capacity Sharing for Overrun Control. *Proceedings of the 21st IEEE Real-Time Systems Symposium, Orlando, Florida.*

Carlini, Alessio. (2002). A real-time kernel for the M68HC11 microcontroller. Master Thesis, Department of Computer Science, University of Pavia.

Carlini, A. and Buttazzo, G.C. (2003). An Efficient Time Representation for Real-Time Embedded Systems. *Proceedings of the 18th ACM Symposium on Applied Computing*, Track on Embedded Systems: Applications, Solutions, and Techniques, Melbourne, Florida, USA.

Dertouzos, M.L. (1974). Control Robotics: the Procedural Control of Physical Processes. *Information Processing*, 74, North-Holland, Publishing Company.

Gai, P. and Abeni, L. and Giorgi, M. and Buttazzo, G.C. (2001). A new kernel approach for modular real-time system development. Proc. 13th IEEE Euromicro Conf. on Real-Time System, Delft, Netherland.

Lehoczky, J. P. and Sha, L. and Strosnider, J. K. (1987). Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 261-270.

Lipari, G. and Baruah, S.K. (2000). Greedy Reclaimation of Unused Bandwidth in Constant Bandwidth Server. *Proceedings of the 12th IEEE Euromicro Conference on Real-Time Systems.*

Liu, C.L. and Layland, J.W. (1973). Scheduling Algorithms for Multiprogramming in a Hard real-Time Environment. *Journal of the ACM* 20(1), pp. 40–61.

Microchip Technology Inc. (2001). PICmicro mid-range MCU family reference manual. www.microchip.com, 2001.

Motorola Inc. (2002). M68HC11 Microcontrollers reference manual. URL: www.motorola.com/semiconductors.

Sprunt, B. and Sha, L. and Lehoczky, J. (1989). Aperiodic Task Scheduling for Hard Real-Time System. *Journal of Real-Time Systems*, 1, pp. 27-60, June.

Spuri, M. and Buttazzo, G.C. (1996). Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real-Time Systems*, 10(2).