Progress Toward an Inexpensive Real-Time Testbed The Pinball Player Project*

(Presented at the Real-Time Education Workshop, 1997)

Dayton Clark
Department of Computer
& Information Science
Brooklyn College/CUNY
Brooklyn, New York 11210
dayton@brooklyn.cuny.edu

Abstract

The Pinball Player Project is nearing completion of its first phase–construction of a working prototype. The immediate purpose of the project is to develop an inexpensive testbed for real-time programming–a device within reach financially and technically of any Computer Science or Computer Engineering department. This paper discusses the need for such a testbed and the rationale for the particular model (a pinball machine and a PC). We also report on the status of the project and our thoughts for dissemination and use of the testbed within the real-time educational community.

1 Introduction

The major programming or systems areas of Computer Science reached profound levels of maturity in the 1950's, 60's and 70's. That is, fundamental theoretical questions were posed and answered and significant tools were developed to exploit and complement the theory. Consider Table 1 which lists significant developments in several systems areas. The anomaly in Table 1 is computer networks, but by in large networks did not exist at all until the mid-to-late 60's.

The ability to control real-world processes by computer became a reality with the advent of the mini-computer at about the same time as the early networks, the mid-to-late 60's. Yet, even today, for developers of real-time systems there are very few tools available and very little consensus within the real-time community as to which tools. Even if one adjusts for the late start of real-time systems, the area seems overdue for galvanizing events like those in the Table 1.

The theory of scheduling real-time tasks has progressed nicely. There is wide-spread consensus on real-time scheduling algorithms. However, even here, the early date of the ground-breaking work (Liu and Layland in 1973 [liu73]) raises the question, where are the development tools?

The major reason for the slow maturation of the field is that real-time programming is difficult. The imposition of inviolable external time constraints increases the difficulty of "normal" programming in a qualitative manner. This cannot be over-emphasized. However, we feel that a significant contributor to the slow development of real-time programming is more mundane. Specifically, the problem is the lack of access to computer controlled real-time systems brought about by the high cost of such systems. This lack of access greatly reduces the number of programmers who are exposed to real-time systems in their education thereby reducing the general knowledge of and interest in the problems of real-time programming. Typical research into real-time systems and the attendant programming is in military, avionic, space or robotics where the devices are expensive and unavailable to many colleges and universities. They are expensive both in initial cost and in maintenance, requiring skilled personnel to keep them operational. An example of this is provided by the robotic hand developed by the University of Utah and MIT in the mid-1980's for use in research into robotic manipulation [3]. This exquisite device was an order of magnitude more expensive than the rather ordinary computer equipment needed for control experimentation and was consequently available at only a handful of laboratories across the country.

The goal of the Pinball Player project is to develop a complete package (control and development

*This project was supported in part by a grant from the Office of Naval Research (N00014-96-1-1057), by PSC-CUNY Research Award Program (665006) and by Brooklyn College.

Area	Development	Date
Scientific Computing	FORTRAN	late 50's
	Standard numerical libraries	late 60's
Languages and Compiling	Backus-Naur Form	late 50's
	Algol	early 60's
	LL & LALR parsers	late 60's
Operating Systems	Synchronization	early 60's
	Virtual memory	late 60's
	Layering	late 60's
Database Systems	COBOL	early 60's
	Relational databases	early 70's
Artificial Intelligence	LISP (symbolic programming)	early 60's
	Frames	early 70's
Computer Networks	Arpanet	early 70's
	Internet protocols	early 80's
	OSI paradigm	early 80's

Table 1: Significant Developments in Various Systems Areas

computer(s), the controlled device(s), sensors, actuators, interfaces, and basic software) that can be put together from commonly available items and without exceptional mechanical or electronic engineering for under US$10,000. The experiences gained from creating such a system will be shared with the academic and research communities at large so that many institutions can develop and expand on the initial system.

The next two sections present the rationale for the plant and computing platform chosen for the testbed. Section 4 describes the current state of the project. The following section describes our vision of how the Pinball Player could be used educationally. Finally, we draw some conclusions in Section 6.

2 Why Pinball?

Selection of an appropriate system to control is critical to the success of the testbed. The system must have a real-time component. In addition, the system must be inexpensive and safe and the control must be simple enough to be within the reach of students and complex enough to be challenging. There are also advantages in a system that can be controlled by humans as well as by a computer. If students are able to control the system themselves (i.e., as humans) the highest level problems and strategies can be intuitively grasped. Human controllers also provide a ready benchmark for the success of the computer controller. And finally, with a human controllable plant, investigation of control strategies involving human training of the computer controller is feasible.

Arcade type pinball machines fit all of the criteria. They are readily available and inexpensive (US$100 for some as-is machines to US$2,500 for new ones). They are safe and easily understood. The basic acts of tracking and hitting a ball are decomposable into pieces appropriate for instruction. The problems of keeping a ball in play and strategies for high scores are replete with challenges. The ability to have the computer "watch" a human play, for training, is a simple extension of the basic computer/pinball machine interface.

The idea of "toy" plants for real-time laboratories is not new. Zhang etal. [zhang94] presented research based on model train system and the author is aware of at least one use of model trains to teach real-time programming. Surely sentimental favorites, model trains have significant drawbacks for use as general educational tools. Although the basic operation and setup is simple–millions of children have or have had them–the creation of a system that presents interesting scheduling problems is non-trivial. Furthermore, maintenance of a model train system while not requiring advanced skills does require devotion and care. Contrast this with pinball machines which are designed to work with minimal setup yet to operate for long periods in unfriendly environments with maintenance limited to the occasional dusting and emptying of coin boxes. Finally, once one has successfully scheduled a particular model train system, the opportunity for further investigation is limited (e.g., changing train lengths or optimizing train schedules). With a pinball machine, every play is different and one can always strive to lengthen the time balls are kept in play. Furthermore, enhancements such as using multiple balls or obtaining a higher score (as opposed to simply keeping the ball in play) are rich with problems beyond the basic pinball play.

Another source of toy testbeds is the many the small mobile robots that are available commercially or can be constructed (see *Mobile Robots: Inspiration to Implementation* [jones93]). The emphasis in these systems is on the electro-mechanical design more than the computer control and the mechanical design is often meant to eliminate or side-step the need for time-critical control. The control of multiple cooperating or competing mobile robots would present many real-time control problems but for general ease of use and setup we have chosen the pinball machine.

3 What Platform?

The choice of which computer to use for control of the pinball machine is clear. The IBM PC and its clones are ubiquitous. Every school has some and probably many of them on hand. New PC's now cost under US$2,000 and old ones are often discarded long before they are unusable. Students and faculty are familiar with the normal operation of PC's. Adapters for control and sensing devices are plentiful and inexpensive. The architecture of the PC is open, which will facilitate the development of special purpose hardware. Note that our goal of developing a testbed that can be easily setup and maintained proscribes the extensive use of special purpose hardware. But, once a testbed system is installed we expect and encourage students and faculty to experiment with hardware modifications to the testbed. We know of no other computer that would serve as well as the testbed's computer.

Selection of appropriate software platform is not as straight forward as the hardware and is in fact very much an open question. We are currently using FreeBSD[1] a freely distributed version of Unix with its roots in BSD variant of Unix from the University of California at Berkeley. We are attracted to it because it is distributed in source form, it is familiar to us, and is inexpensive. FreeBSD has some support for real-time processes and supports a POSIX compliant threads library[2].

There are many operating systems for the PC architecture. We have considered the better known and a brief summary of our thoughts on them follows. The PC's native operating systems from Microsoft (DOS and the various Windows) and IBM (OS/2) are obvious choices. However, we feel it is important that the testbed's operating system be as open as possible. We feel it is important avoid hindering experimentation with hidden and/or immutable characteristics of a non-real-time operating system. So, despite their widespread familiarity we feel these operating systems

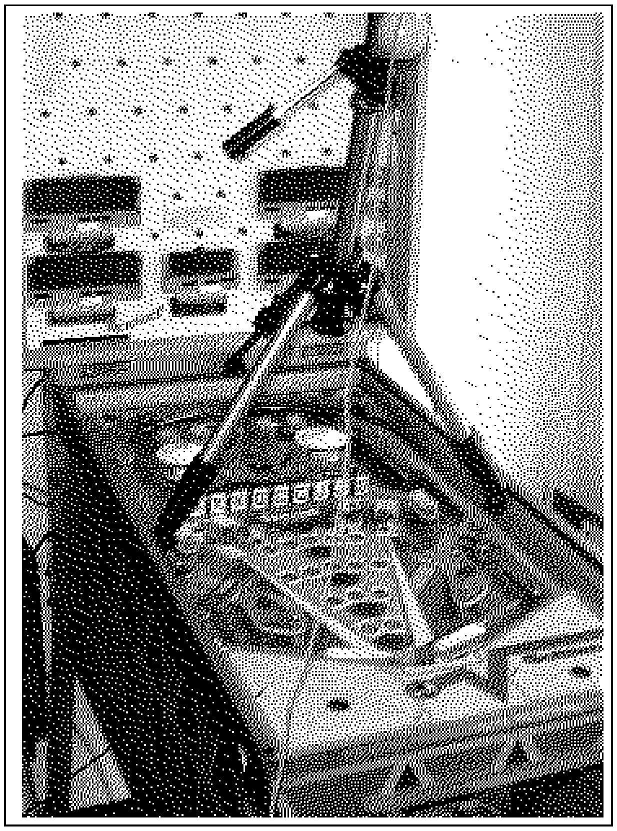

Figure 1: The pinball machine

are inappropriate. For similar reasons we are not using commercial versions of Unix (Solaris and SCO's Unix). Commercial real-time operating systems such as VxWorks by Windriver Systems are also possibilities, but we have opted not to use one of these because of their proprietary nature, lack of familiarity, and cost. There are other free Unix systems, most noticeably Linux, our choice in this cases is mostly a matter of our own familiarity.

It is important to recognize that our use of FreeBSD is not an endorsement of FreeBSD as a real-time operating system. Rather, with the soup of constraints and options available to us, we have found FreeBSD to be the most useful platform for developing the testbed. In fact, our difficulty in selecting an operating system goes to the root of the motivation for this project, namely that with respect to software there is little consensus as what tools should be used.

Our prototype version will be based on this platform but we fully expect that if the project is adopted by other institutions that other operating systems will be used. In particular we would hope that support arose for the standard PC operating systems, DOS

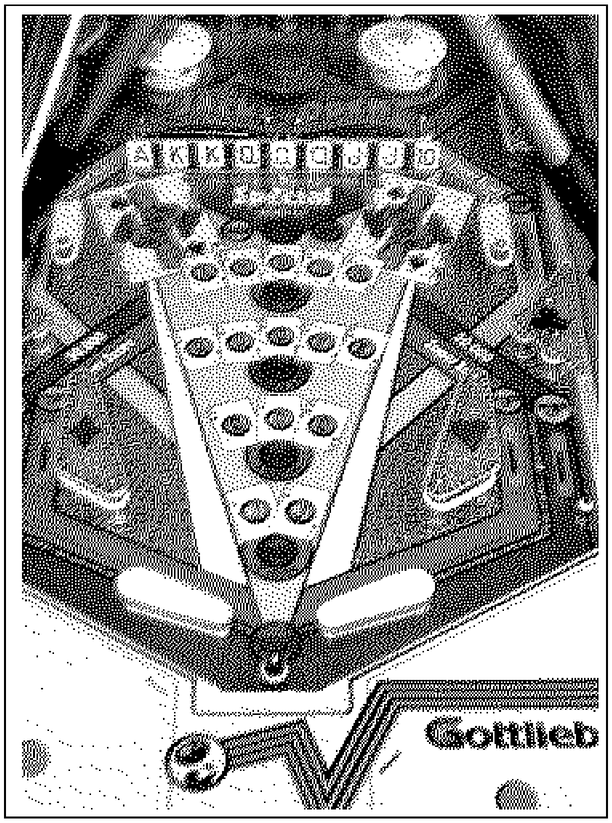

Figure 2: The pinball machine playing area

and Windows, other free operating systems such as Linux, and hopefully for real-time operating systems.

4 Current Status

This section describes the current status of the development of the prototype system. We present first the hardware, our pinball machine, control computer, the sensors and the actuators.

4.1 Hardware

The hardware in the prototype system consists of a pinball machine, a PC, a video input device, a digital and analog I/O board, and interface circuitry consisting of two relays connected to the pinball machine's flipper circuits (see Figure 3). The software consists of the operating system (modified slightly to support the sensors and actuators), our control software, and some initial control strategies.

4.1.1 The Pinball Machine

The plant for our prototype is a pinball machine called *Jacks to Open*. It was manufactured in the early 1980's by a company called Gottlieb. Figure 1 shows the pinball machine and Figure 2 shows the playing area.

The upper third of the playing surface consists of an obstacle course of bumpers and gates through which the ball must travel. The purpose of the area is to randomize the ball's behavior. The lower two-thirds of the playing area (roughly 45cm by 45cm) is mostly unobstructed with only a few bumpers and gates. At the bottom of this area are two flippers which guard the ball's main avenue of escape.

The only control available to the player is the flippers, each of which is controlled by a normally open switch located on the side of the machine. When a switch is closed the corresponding flipper is driven by a solenoid through an arc of about 90 deg to its "up" position. The flipper stays in this position until the switch is released. Control of the flipper is binary; there are no intermediate positions; it is either full on or full off.

4.1.2 Sensors

Pinball machines are designed to stimulate humans. There are a great many audible noises and flashing lights accompanying the play. Some of this activity provides information to the player such as the number of balls and games remaining, the current score, and "bonus" states in which certain actions further increase the player's score. It will be interesting in later stages of the project to track some of the auxiliary activities. At present we are exclusively interested in the ball's location so we can actuate the flippers at the appropriate times.

The pinball machine itself has several dozen sensors for keeping score and activating lights and sounds. There are sense switches in most of the gates. Many of the bumpers sense when the ball has struck them, Some of the obstacles sense when the ball strikes and then actively push the ball away. Our particular machine is monitored by a microprocessor (one would assume that this is true of all pinball machines manufactured in the last two decades). Tapping the sensor circuits is conceptually easy but it turns out that the information they provide is of little use when trying to hit the ball. Examination of Figure 2 shows that the top of the play area is densely populated with bumpers and gates and the lower portion (just above the flippers) is sparsely populated. Thus the machine's sensors provide little information about the ball's trajectory as it nears the flippers, precisely when we need it most.

So we are led to develop our own sensors. Our original plan was to avoid video input for two reasons. One reason was that video equipment tends to be more expensive, which was contrary to the goal of

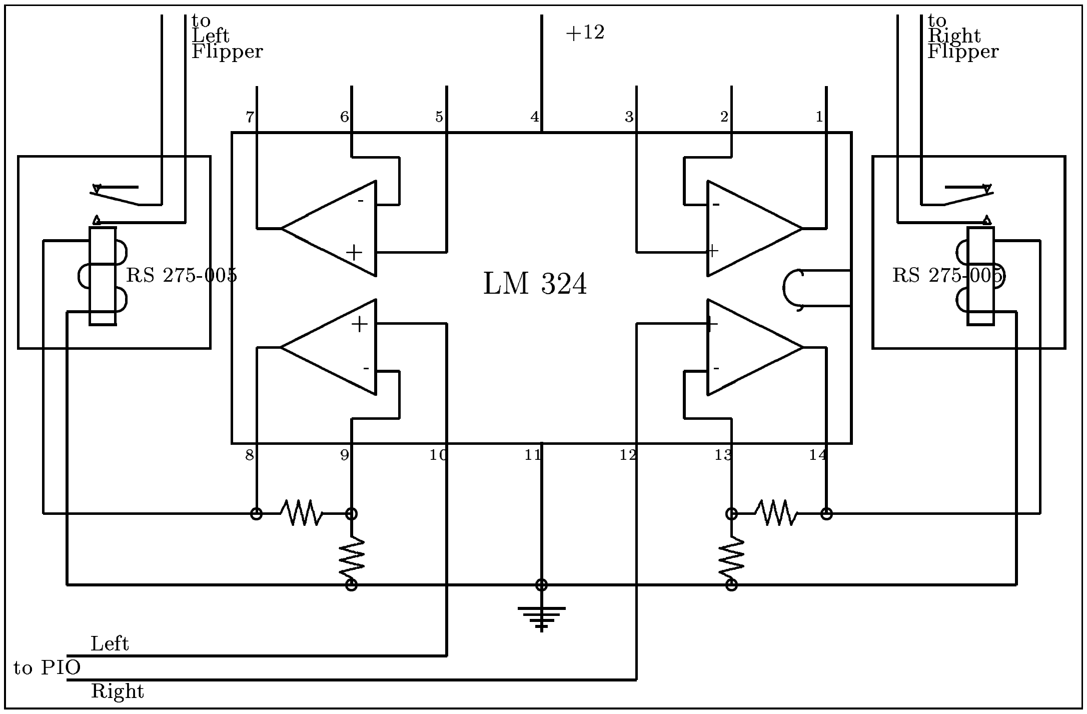

Figure 3: Circuitry connecting PIO to flippers.

an inexpensive testbed. The second reason was that the immense problems of computer vision would likely overwhelm the control problems which are the purpose of the testbed. Two experiences led us to reevaluate this choice. First, we experimented with various sensing schemes based on broken or reflected light beams with occasional but very sporadic good results. Second, as our frustration rose, small cameras for use in video conferencing over the Internet arrived on the market. These cameras, often shaped like billiard balls are connected directly to a computer's parallel port (no frame grabbing card), are able to deliver around $15 frames/second$, and are very inexpensive (less than US$100 for a black and white camera).

We installed a *QuickCam* camera by Connectix and have had tremendous results. We can now track the ball in real-time using a very simple algorithm (see Figure 4). We detect motion by subtracting each image from its predecessor and thresholding. This yields a binary *difference* image which contains only the ball's locations in the two original images. A simple calculation using two of these difference images yields an image containing only the ball's current location. There are three drawbacks to this simple scheme. One is that the flippers also move so we need to distinguish them from the ball. At present, we simply ignore any activity in the immediate region of the flippers. Second, the pinball machine's own lights which flash at indeterminate times can fool our sensors. Our current solution is to turn off the machine's lights. Third, shadows cast by someone walking near the machine may throw the sensor off. Our response is to stand clear of the machine while it is playing. We plan to implement a more sophisticated detection algorithm soon which we hope will overcome these disturbances.

4.1.3 Actuators

Each of the machine's flippers is driven by a 24 volt solenoid. When the input circuit is closed the solenoid pushes the flipper through an arc of approximately 90 deg. The flipper remains in this position until the circuit is opened. We have put a relay in the circuit, in parallel with the button used to activate the flipper. An output signal from a digital I/O board is amplified and becomes the input to the relay. The electronics are very simple and a program controls each flipper by toggling a bit on the I/O board.

A common question about the project is "How do you control the plunger which launches the ball into play"? At this time we don't. A device which would pull back the plunger and then let go to launch the ball would be interesting, but the problem is mostly a mechanical engineering problem and not of direct concern