The Rototack: Designing a Computationally-Enhanced Craft Item

Tom Wrensch, Glenn Blauvelt, and Mike Eisenberg

Department of Computer Science and Institute of Cognitive Science

Campus Box 430

University of Colorado, Boulder CO 80309-0430 USA

wrensch;zathras;duck@cs.colorado.edu; +1 303 492 8091

ABSTRACT

This paper describes our progress in creating a device called a *rototack*. In its design, the rototack is an example of a *computationally-enhanced craft item*: a small, robust, inexpensive, and versatile – but also programmable – physical object for use in a variety of educational and home crafting projects. In particular, the tack is a source of rotational motion, suitable for turning light objects or for powering (e.g.) cams, gears, and linkages in complex, user-defined patterns. We describe the engineering decisions and trade-offs involved in creating our current prototype of the tack; discuss the central issues in creating a programming language and environment for the device; and sketch a variety of potential uses to which the tack might be put.

Keywords

Rototack, computationally-enhanced craft items, computation and crafts.

INTRODUCTION

The idea of integrating computational worlds and physical objects is a broad one; and there are many ways to pursue it. One approach, for instance, focuses on the creation of "smart" objects and environments – automobiles that know the path to their destination, houses that adapt their lighting and temperature to the lifestyles of their inhabitants, toys that hold coherent (if rudimentary) conversations with their young owners, and so forth. (See [Mozer 99, Umaschi 97] for examples of some of the more fascinating efforts in this direction.) Another approach (exemplified by [Feiner *et al*. 93]) enhances the behavior of physical objects by permitting them to communicate with portable computational devices. Still another approach exploits the last decade's awe-inspiring advances in miniaturization of components, creating "smart" materials and microelectromechanical systems (MEMS), in which (typically very large) numbers of extremely tiny – and often invisible – computational devices are embedded (cf. [Berlin and Gabriel 97]).

DARE 2000 April, 2000 Elsinore, Denmark

Each of these approaches tackles the integration of computers and the physical world in its own characteristic, and richly productive, way. The approach that we have explored over the past several years ([Blauvelt *et al*. 99; Eisenberg and Wrensch 98; Eisenberg and Eisenberg 99] represents yet another style of virtual-physical integration: one that focuses on the creation of simple, end-user-programmable craft items. That is to say, we begin by examining the landscape of inexpensive, multi-purpose objects found in craft or hardware stores – tacks, hinges, tiles, to name a few – and we then attempt to determine whether there are ways in which these objects might be usefully extended with simple and explainable computational capabilities.

Just to take an example of the idea: imagine that you are a craftsperson who makes dolls. Your latest doll design is a formally attired Victorian woman, complete with fan. You have found a way to make a small fan which functions like its full-sized counterpart; and rather than just have the doll's owner fold and unfold the fan with a large finger, you would like to have the doll open the fan, raise the fan in front of her face, pause, and then refold the fan and lower the arm. This natural and unexpected motion would make your doll different from other dolls in ways other than appearance. Having access to enhanced versions of the materials you currently use – materials and components that offer a means to achieve the specific motion you desire – would allow you to extend the scope of your crafting, from constructing purely static works to constructing works with more complex behavior.

This paper describes our progress in creating one such computationally-enhanced craft item: a programmable tack that we have dubbed a *rototack*. The rototack – still very much a work-in-progress – is a small device with the approximate geometry of a thumbtack. By writing short

programs for the device, the user of the rototack can specify a variety of moderately complex rotational patterns in which the head of the tack rotates around the axis defined by its spike. The tack thus acts as a source of rotational motion that can be used to turn paper displays, power light plastic gears or cams, and move linkages.

The remainder of this paper expands on this highly abbreviated description. In the following (second) section, we describe the idea of the rototack in more detail, and illustrate the device with a scenario of its use. The third section delves into many of the technical challenges involved in creating the rototack hardware, and explains the decisions that we have made to date. The fourth section focuses on the software side of the device, looking at the manner by which the tack can be programmed by its user; in this section, we also discuss some of the broader issues in end-user programming that are brought into high relief by our experiments with craft items like the tack. Finally, the fifth section steps back to speculate on the many craft and educational projects in which rototacks could be used; and we conclude with a discussion both of related work and of the (still formidable) agenda of work to do in bringing the rototack to realization.

THE ROTOTACK: AN OUTLINE

This section describes our basic design ideas for the rototack. A prefatory note is in order here: in general, our remarks in this section should be interpreted as a blueprint for the eventual tack, rather than a description of a completed project. To date, we have constructed two separate working prototypes of the tack; the first ("Mark I") is shown in Figure 1, and the second ("Mark II") in Figure 2. In both cases, the object is structured as a large tack whose head contains a small computer, battery, and motor, as shown in the schematic diagram of Figure 3. The software for the device – to be described in more detail later – is likewise at an early stage; we have constructed mockup versions of the tack programming language and environment, and a rudimentary compiler. Thus, as of the time of this writing, we have yet to combine a practical and stable software environment with the rototack object itself.

The geometry of the rototack is suggested by the two prototypes of Figures 1 and 2: that is to say, the tack is a roughly cylindrical head attached to a spike. The purpose of the tack is simply to rotate its head (or more accurately, a shaft that protrudes through the head, as shown in Figure 3) about the axis defined by the spike in a manner determined by the user. Overall, then, the scenario by which the tack is employed can be sketched as follows:

1. The user brings up a tack-programming application on her desktop computer. The application requests that the user connect a rototack object to the machine for programming.

2. The user connects a rototack to the machine (there are several plausible ways of accomplishing this step) so that the tack-programming application may now communicate with the tack directly.

3. The user writes a new program for the tack (or retrieves a pre-existing program) and downloads it to the rototack. The tack's local computer now contains a runnable version of the desired program. Just to take a specific example, the program might indicate that the tack should repeatedly cycle through the following pattern of motion: rotate 90 degrees clockwise; pause five seconds; rotate 180 degrees counterclockwise; and pause for three seconds.

4. The user now inserts the tack, with its embedded program, into a corkboard and presses a small button in the side of the tack; in response, the tack begins running its program. To stop the program, the user presses the button a second time. An alternative design might use an "in-corkboard sensor" within the tack head; the idea here would be that the tack could sense when it has been placed into a surface and could automatically begin running its program at that time (the program stops when the tack is removed from the surface).

Figure 1. A side view of the Mark I rototack. The spike may be seen at right; a gear attached to the rotating shaft is at left. The head is at center is composed of two bottle caps, and houses the power source and internal computer.

Even at this coarse-grained level of description, there are many design alternatives to be considered. In step (2), for instance, we might ask how precisely the desktop computer should communicate with the tack. One possibility would be to send a program to the tack via an infrared signal; this is the approach taken by the MIT Media Lab's "cricket" version of the Lego programmable brick.[Resnick *et al*. 96] Another possibility would be to have a cable attachment from the desktop machine that plugs into a socket in the head of the tack. Still another technique – one that we rather favor – would be to build a peripheral device that

could hold multiple tacks at once, allowing the user to download a program to all of the tacks simultaneously. Such a device might be thought of as a box, connected to the computer, and containing a row of slots into which tacks may be inserted; by downloading a program to the box, the user thus sends that program to all of the tacks in the slots.

Figure 2. A photograph of the Mark II rototack (shown with a variety of other, noncomputational tacks for comparison).

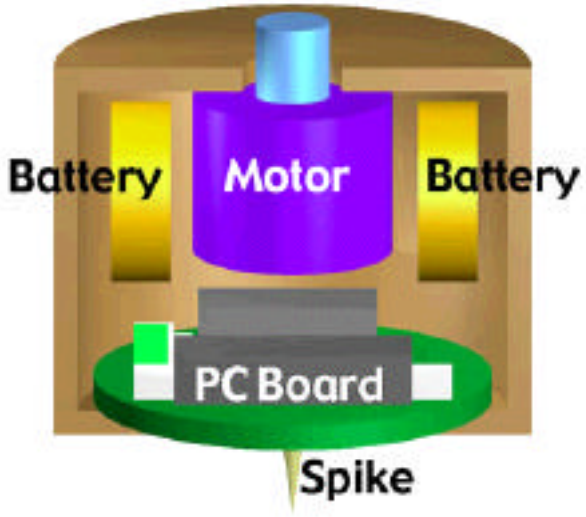

Figure 3. A schematic cross-section diagram of the rototack, showing the electronics, batteries, and stepper motor.

Step (3) presents its own challenges as well. What should a programming language for an object like a rototack look like? Should it be a purely visual or iconic language; a dialect of some existing higher-level language like Logo or Basic; or perhaps a brand-new, special-purpose textual language? What sorts of primitive procedures and data objects should be built into such a language? And more generally (and perhaps more importantly), what sort of language "aesthetic" are we trying to capture – should this be conceived as a hobbyist's language that places the user close to the specific hardware of the tack (e.g., by giving the user information about the speed of the tack's motor, its clock rate, and so forth); or should it be conceived as a high-level end-user language that hides as much hardware detail as possible?

Perhaps the greatest challenges of all are reflected, if indirectly, in step (4). What sorts of projects might a rototack be used for; and why would such a device be more appropriate than other, plausible alternatives?

In the remainder of this paper, we will explore the various dimensions of the design problem – hardware, software, and pragmatics – in greater depth. But before delving into these questions, it is worth pausing to examine the larger goals involved in creating new craft items like the rototack, since these goals will both inform and constrain the types of design decisions that we are likely to consider.

What, then, are the overall desiderata for a device such as the rototack? Once again, we look to our governing analogy between computationally-enhanced craft items and their low-tech, noncomputational, homespun cousins. Ideally, then, a rototack should be not too much more expensive than a thumbtack; it should be perceived by the user as an expendable, throw-away sort of item that comes in boxes of fifty or more. Quite probably, there might be several varieties of rototacks, ranging from larger and more powerful devices (for turning, say, basswood or even metal gears) to lighter items (for turning simple cardstock displays). Each variety of tack will reflect a particular trade-off, then, between the potentially conflicting goals of creating an item that is small, light, inexpensive, powerful, robust, and programmable. The means of programming these various tacks should be relatively consistent, however; we would not (e.g.) want to use a distinct programming language for each new size or grade of tack that becomes available.

The same analogy that leads to these potentially thorny design constraints does, at the same time, afford us a certain leeway. We do not, for example, envision a need for particularly powerful computational capabilities in a device as simple as a rototack; it is unlikely (at least as a first approximation) that the device would need to run programs of much greater complexity than the illustration presented in step (3) earlier. By the same token, the language

environment with which the tack's programs are created need not be designed with an eye toward producing long or complex programs, but rather can be tuned to the creation of smaller, "tack-appropriate" programs.

In short, then, we do not wish to view the rototack as a scaled-down general-purpose computer (in the spirit of the programmable Lego brick and its variants). Rather, it is an object with a specific geometry and limited range of behaviors. This makes the tack in some ways less powerful than a general-purpose computer; but it likewise makes it an easier fit to many projects that fall within the scope of the tack's capabilities. Moreover, we envision the tack as, ultimately, just one of a large collection of similarly programmable craft objects. We will return to this point toward the end of this paper.

THE ROTOTACK: COMPUTATIONAL HARDWARE AND PHYSICAL DESIGN

In this section we make a closer examination of the physical and computational design of the rototack, as schematized in Figure 3 earlier. Our discussion focuses on the decisions that we have made to date in the creation of our two prototypes.

• *Motor.* In choosing a motor for the tack, we need one that will turn relatively slowly: given the tack's light weight, if it spins too rapidly with an unbalanced load it could well fly out of the surface in which it has been embedded. Interestingly, finding an appropriately slow motor is not especially easy. Most electric motors rotate at hundreds or thousands of RPM; we in fact needed something closer to a few dozen RPM. In practice, this type of problem is usually solved by employing a gear reduction; but the tack's small size made this solution unworkable.

After a bit of searching we found two possible solutions. The first, and the one we used in the Figure 1 prototype, was the same low-speed/high-torque DC motor employed in the Lego Dacta system. The second, used in the Figure 2 prototype, was a small stepper motor.

Overall, we believe that stepper motors are the better solution for rototack design. They are inexpensive, come in fairly small sizes, have good torque for their size, and their speed and position can be precisely controlled. Even better, there are very small stepper motors that run at three volts, effectively reducing the size of the device and the number of batteries required. But, these advantages come at a price: in comparison to the DC motors, there is a substantial increase in the complexity of electronics and firmware needed to drive the stepper motor.

• *Power.* In creating our two prototype tacks, the power requirements of the motor were the most critical factor in deciding on a power source. The Mark I and Mark II tacks used (respectively) three and two 1.5-volt button alkaline cells. In both cases, the batteries provided sufficient voltage and current to operate the motor, but only at a very low torque. We are currently looking at a prototype using larger batteries for both increased torque and longer battery life.

The problem of powering an item such as a rototack is, in general, a difficult one. Conceivably, passive recharging systems (such as those described in [Gershenfeld 99]) might eventually be used for the tack, though such solutions appear to be better suited to items whose power requirements are much lower. Another possibility – one that we have discussed but so far resisted – is to design the tack in such a way that it can be powered externally. (For instance, the tack might be constrained to work only in conjunction with a special corkboard surface behind which we have positioned the tack's power source.) Our preference, at least for now, is to keep the power source within the body of the tack, if only because this poses fewer constraints on the sorts of uses to which the device might be put.

• *Electronics.* The circuit board for our current tack prototype is shown in Figure 4; the board supports a microcontroller, motor driver, red/green LED and clock circuit. The motor driver is a line driver rather than a stepper motor controller. This reduces the overall size, power consumption, and cost of the board, but puts the responsibility for driving the motor on the microcontroller firmware.

Figure 4. A photograph of the current rototack circuit board. The PIC microcontroller is the larger chip toward the bottom.

The microcontroller is the Microchip PIC16LF84, a low voltage version of the popular PIC16X series. There are two major advantages to using this particular processor. The first is that it includes, on chip, 64 bytes of user controlled EEPROM which we use for user program storage. This eliminates the need for a separate memory device, though it does limit the size of the user programs that can be stored (see Figure 5). The second is that it is a very sturdy chip: running it with the voltage out of range,

overclocking, accidentally putting too high a load on an input pin, or even overheating a pin while soldering will rarely cause the chip to fail – a very useful attribute for a hobbyist's item (or, for that matter, an item used in research).

• *Firmware*. The rototack firmware (diagrammed in Figure 5) is the stable computational element of the tack – i.e., the background software that serves as the environment in which the user's program will be executed. The firmware includes a bytecode interpreter, communication code, and device drivers for the stepper motor. Despite this there is ample room in the microcontroller's 1024-word programming space and 68 bytes of RAM for continued development.

The firmware is driven by a main loop which executes once every 128 instruction cycles (approximately every 40 milliseconds). The loop may be summarized as follows:

```
LOOP
    Synchronize to 128-cycle clock
    Check for input from "in corkboard" sensor
    Check if the LED needs attention
    Check if interval timer is on
    Check if stepper motor needs attention
    Do a bytecode instruction
END LOOP
```

Two of actions in the loop – the interval timer and stepper motor checks – may "saturate" this cycle. For example, if the interval timer is on, the cycle is used up and none of the actions below the interval timer check in the loop will happen. In contrast, a sensor event or an LED state change do not stop the cycle's execution.

The bytecode interpreter defines a bytecode language for the device. This includes 26 general-purpose instructions, three instructions for the interval timer, and eight specifically for controlling the rototack hardware. The model used is a simple stack-based system with a fixed set of registers assigned for use as global variables.

• *Materials for the tack case*. The Mark I prototype case (shown in Figure 1) was made from two plastic caps from two liter soda bottles. These are easy to find, inexpensive, are easy to screw on and off, and are an appropriate size. They do lack any internal bracing for the smaller motor and lack something in aesthetics as well. With the Mark II prototype (shown in Figure 2), we used a laser cutter to cut out disks of wood, which were then glued together to build up the case. This technique allowed us to build in internal braces for the batteries, motor, circuit board, and spike, and gave the final tack a more appealing appearance.

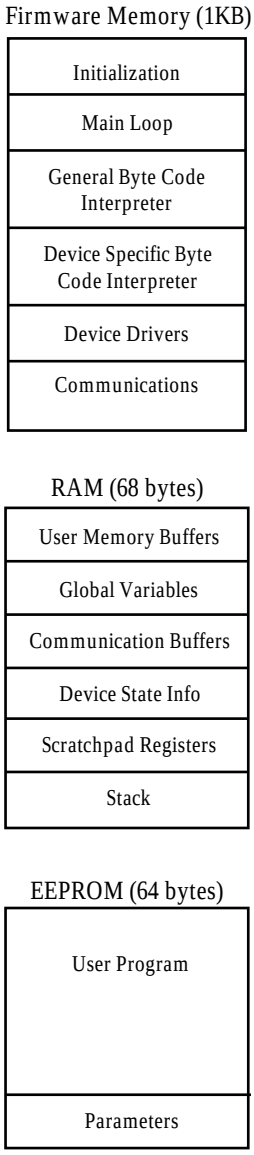

Figure 5. A diagrammatic representation of the rototack firmware, RAM, and user memory.

THE ROTOTACK: PROGRAMMING LANGUAGE AND PROGRAMMING ENVIRONMENT

The previous section focused on the hardware and materials used in constructing the rototack. In this section, we turn our attention to the issue of how the tack might be programmed by its user. Several initial points are worth making in this regard:

• It is a safe assumption, at least for the foreseeable future, that rototack programs will be (by software engineering standards) refreshingly simple – perhaps no more than a

page in length. Even within these constraints, of course, a huge variety of programs may be written; it is possible to accomplish quite a lot in a page of code. Nonetheless, many of the standard concerns of programming languages and software design courses (e.g., writing programs in modules or packages, encapsulating data in objects, including numerous error checks, and so forth) are far less likely to apply in the context of programming an object like the tack. This affords us, as language designers, a certain degree of freedom to explore a variety of novel or experimental language models.

• Devising a language for an object such as the rototack provides an interesting exercise in straddling the boundary between a "high-level" and "low-level" language. On the one hand, our language should share some of the traditional features of high-level languages: it will likely include classic control structures such as iteration and conditionals, and will provide primitive procedures for complex operations such as "turning the tack by N degrees". On the other hand, there are some aspects of a rototack language that may be regarded as "low-level": unlike traditional high-level languages, a rototack language is intimately related to the physical hardware for which it is designed. In other words, whereas the traditional assumption has been that (say) a Java or Pascal programmer need not be overly concerned with the particular machine or hardware on which her program will run, a rototack programmer is consciously creating a program for a particular device (or class of similar devices). Thus, a rototack language might wish to present the user with what, by traditional standards, would seem to be low-level hardware details: the language might, e.g., inform the user of the clock rate or motor speed. Likewise, the rototack user might be restricted to (e.g.) only integral numeric values in order to save program length; this type of restriction is reminiscent of machine code programming rather than traditional high-level programming.

• Finally, it is worth noting that when designing a programming language for a craft item such as a rototack, one must keep in mind a crucial distinction between the language in which rototack programs are written and the programming environment in which rototack programs are first created and edited. That is to say, the tack's programs run on the (relatively simple and constrained) hardware of the tack itself, whereas the programming environment exists on a desktop machine. Thus, even though rototack programs may be relatively simple, and even though the programming language may be designed to produce only short and straightforward examples of code, the programming environment may be quite sophisticated. This leaves us free rein to explore ways in which the environment might be made more powerful or accessible (code animation, programming-by-example techniques, and browsable libraries of reusable code are among the possibilities we have discussed; see [Eisenberg et al. 99] for much more along these lines).

What might a typical program for the tack look like? A representative example follows:

```
A := 0
B := 0
Loop
   Repeat 3
      Clockwise
      B := Random(2)
      If B = 0
         Then Turn 90 degrees;
            Wait 10 tenths
      End If
   End Repeat

   Repeat 2
      Counterclockwise
      Turn A degrees
      A := A + 45
      If  A = 360
         Then A := 0
      End If
      Wait 20 tenths
   End Repeat
End Loop
```

In prose, this program loops indefinitely with the following pattern: the tack "flips a coin" three times, each time either turning by 90 degrees and pausing, or remaining in position; then the tack turns counterclockwise twice by a variable amount (the amount increases by 45 degrees at each turn until reaching 360, at which point it returns to 0), pausing a bit longer after each of these two turns.

This program illustrates several of the issues raised earlier in this section. It is, above all, brief; indeed, by rototack standards, this might be a long program, using 43 of the 64 bytes available in user memory. It makes use of classical high-level control constructs such as Repeat, Loop, and If/Then, and it likewise includes primitive procedures such as Random, Turn, and Clockwise that might correspond to nontrivial computational actions; but at the same time, it employs only integer arithmetic, simple variable names, no compound data structures, and no higher-level procedures.

Perhaps most interestingly, the actions that this program takes are, to a great extent, reflected directly in the rotational movement of the tack. This in turn suggests that a language environment providing not only the textual version of the program but also an animated representation of the tack's movements would be useful to the rototack programmer. We have designed a mockup version of such an environment; several screens illustrating the construction

of a moderately complex rototack program are shown in Figure 6.

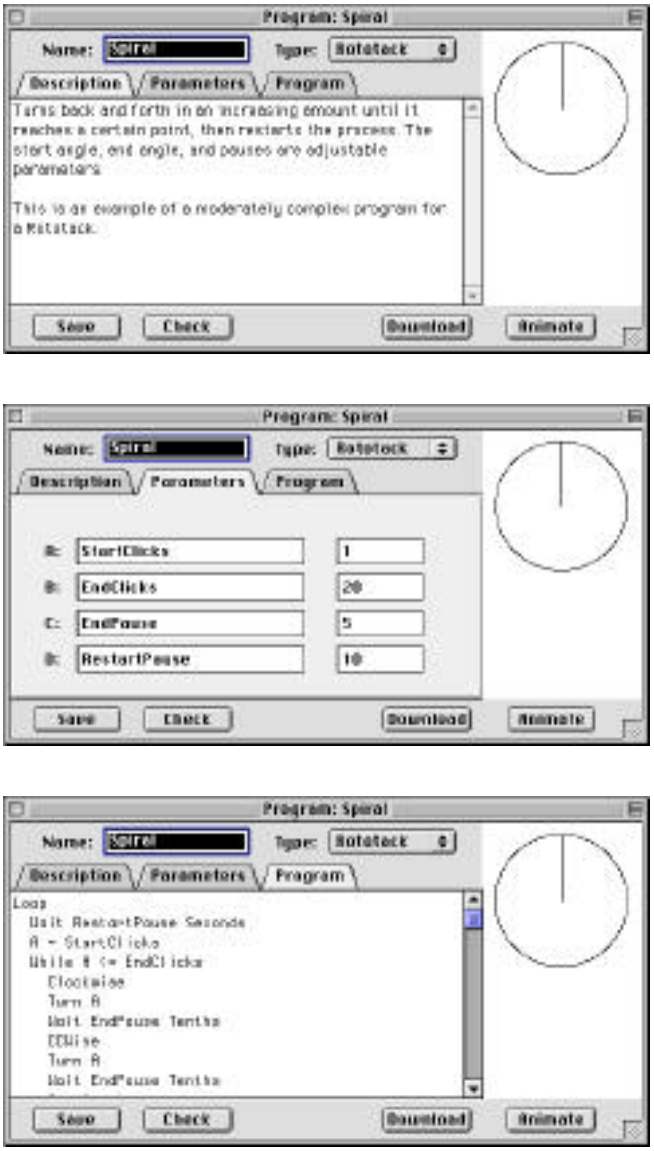

Figure 6. Several mockup screen views of a rototack programming environment. In this system, we assume a simple textual programming language linked with a graphical animated tack. Each brief program is indexed by a textual description (as in the screen at top); an annotated list of parameters (the center screen); and the text of the program itself (the bottom screen). In this last view, the program may be simulated: the Animate button toward the bottom right links the running program to the graphical tack at right.

In this environment, programs may be accessed by a short textual description (as in the top screen); their parameters may be displayed in an annotated list (as in the second screen); and the program text itself may be edited and run in association with an animated drawing of the tack (as in the bottom screen; the line drawing of the tack at right is linked to the running program by the "Animate" button at bottom right). Again, because we are dealing with programs on a relatively simple scale, the "scaling issue" is less salient here than it would be for a traditional programming environment.

The Figure 6 mockup represents our first steps toward designing a working programming environment for the rototack, but even at this very early stage it has proven to be a provocative object for discussion. For example, the natural "animatability" of rototack programs suggests that tack animations would be a feasible means by which to browse through program libraries. That is, a rototack user looking for an existing program with a particular behavior could begin by browsing through a tableau of tack animations showing different patterns of movement; by selecting the appropriate animated pattern, the user could then bring up the program that gave rise to that movement. This is a means of software reuse that is a natural fit to a computationally-enhanced craft item, since tack programs bear a close relationship to identifiable real-world motions; in contrast, it would be far more problematic to construct a similar software-reuse-by-animation system for (say) numerical algorithms.

ROTOTACK PROJECTS; RELATED AND FUTURE WORK

A computational craft item should be sufficiently small and simple so that it does not overwhelm its newest users with complexity; but it should also be versatile enough to find uses in a myriad of projects. The rototack could, we believe, prove to be a powerful example of the genre. Though the tack's programmed behavior will not be, in the vast majority of instances, terribly complicated, rotational motion is still astonishingly protean.

Just to provide food for thought, then, here is a brief sampler of potential rototack projects (one could think of this list as "a dozen things to do with a rototack"):

• *Optical illusions*. The tack could be used to turn displays that produce dynamic illusions (e.g., a spiral, when turned, produces an illusion of depth). Likewise, the tack might be used to turn striped semitransparent surfaces relative to one another to produce Moiré effects.

• *Kinetic Artwork*. The tack could be used to turn a representation of a moving wheel in a display; it could be used to turn a helical paper form; or it could turn elements in a kaleidoscope to produce changing symmetrical displays.

• *Rotating Information Displays*. Certain types of information displays – musical intervals, color wheels, star charts – have a naturally circular structure. A rototack could be programmed to turn such circular information displays in useful ways. For example, the tack might turn a star chart to follow the apparent movement of stars in the sky over the course of a night or series of nights.

• *Polarized filters*. By attaching a polarized filter to a rototack, and turning the filter behind a second stationary filter, the tack could be used to selectively hide and reveal displayed objects in programmed fashion.

• *Globe and Orrery*. The tack might be used to a turn styrofoam sphere representing the earth to show a simple display of a turning globe. A more ambitious project might employ more than one tack to turn two globes representing an earth/moon system around a stationary light source representing the sun.

• *Automata*. The rototack could power light cams, gears, pulleys, and linkages in various automata and mechanical displays. Such displays could illustrate (e.g.) the operation of motors, the design of gear trains, or the transfer of motion effected by a ratchet-and-pawl arrangement. Nor need these displays be purely abstract; there is a venerable and thriving tradition of gorgeous mechanical artwork that could prove fertile ground for rototack projects. Just to take a specific example: the marvelous automata designed by the British artist Paul Spooner [1986] could be adapted to work with the tack.

• *Moving liquids*. The tack could be used (perhaps in conjunction with a magnet) to create an apparatus for gently stirring liquids; or it could be used as part of an apparatus to cycle water for small homemade or classroom fountains.

• *Rhythmic motion*. The tack could be used to power a large display representing clock hands or a metronome's bar for simple, visible timing.

• *Arrhythmic motion*. Because the tack can be programmed so as to produce random or aperiodic turning, it could be used to create artistic effects that would be hard to produce by other means. For instance, the tack could be used in a paper sculpture of a tree to generate "rustling" of leaves; or (following on the earlier "moving liquid" example) it could be used to produce occasional rippling of water, rather than a steady or cyclic motion.

• *Light display*. Multiple tacks might be attached to inexpensive light sources or lasers to produce a complex rotating light display.

• *Moving joints*. The tack (or multiple tacks) might be employed in a puppet or clockwork figure to produce the apparent motion of joints.

• *Bell-ringing*. The tack could be connected by a string to a bell to produce a timed chime effect.

These examples represent, of course, just a fraction of the space of possible rototack projects. While they reflect the versatility of the object, they also suggest its limitations. For example, most of these projects are designed for a single tack, or for several independently-operating tacks. It would be worthwhile to consider, then, what would be gained in power (and/or lost in simplicity) by permitting tacks to communicate with one another (e.g., via infrared signals or radio waves). Yet another limitation encountered in these projects is that the tacks have no means of sensing input: on this model of the tack we could not, for instance, devise a project in which the tack turns a mirror to follow a light source.

Our inclination, for the present, is to design the initial tack with an eye toward maximal simplicity. We believe that, as an increasing variety of projects are attempted, we will develop better intuitions about what future generations of the tack should look like. In any event, we maintain that adding features to a craft item such as the tack is not something to be done lightly; every new feature comes at a price – a price in cost, in size or weight, in complexity of programming language. Moreover, we would argue that the class of possible tack designs does not represent a totally ordered space: it is not the case that any one design is necessarily better than another. Rather, certain designs will prove useful for certain types of projects. Over time, then, we expect that there will be at least several distinct styles of tacks.

Naturally, we see this discussion as expandable to the larger question of designing all sorts of computationally-enhanced craft items. In our own work, we have created prototypes of a programmable hinge and a programmable ceramic tile [Wrensch and Eisenberg 98]. Still other craft items – string, yarn, screws, paper, fabric, and so forth – might likewise be enriched by a modest degree of computational power and programmability. There is, we believe, a tremendous landscape of home and educational craft items to be populated in this way.

Our interest in computational craft items resonates with a growing body of work in the creation of small, informal objects that blend physical and computational attributes. Undoubtedly we have been most influenced by the work of Mitchel Resnick and his colleagues at the MIT Media Lab in their development of the programmable Lego brick and its descendants.[Resnick *et al*. 96, Resnick *et al*. 98] It is particularly worth emphasizing that this is a style of work

in which computational objects are made end-user-programmable; that is, the emphasis here is not on making physical objects more intelligent, responsive, or adaptive *per se*, but in making them more expressive. In this respect, Resnick's work is distinct from much of the tradition of "ubiquitous computing" in which powerful (but arguably mysterious) computational devices are embedded throughout the everyday environment. At the same time, we have also been influenced by the remarkably beautiful work of Hiroshi Ishii and his colleagues, also at the Media Lab [Ishii and Ullmer 97]. The objects and systems designed in Ishii's laboratory are especially striking in the variety of materials that they employ and in the sheer profusion of techniques for embedding computation into informal, playful objects and settings that they exhibit. Our hope, then, is that by creating computationally-enhanced craft items such as the rototack, we can find ways of blending the traditions of (on the one hand) end-user-programmability and educational richness, and (on the other) material variety and aesthetic elegance exemplified by these two researchers and their colleagues.

ACKNOWLEDGMENTS

We are indebted to the ideas and conversation of Robbie Berg, Ann Eisenberg, Gerhard Fischer, Mark Gross, Mitchel Resnick, and Carol Strohecker, among many others. The work described in this paper has been supported in part by National Science Foundation grants CDA-9616444 and REC-961396, and by a Young Investigator award IRI-9258684 to the third author. The second author is supported by a gift from the Mitsubishi Electric Research Laboratories (MERL) in Cambridge, Massachusetts. Finally, we would like to thank Apple Computer, Inc. for donating the machines with which this research was conducted.

REFERENCES

1. Berlin, A. and Gabriel, K. [1997] Distributed MEMS: New Challenges for Computation. *IEEE Computational Science and Engineering*, Jan-March 1997, pp. 12-16.

2. Blauvelt, G.; Wrensch, T.; and Eisenberg, M. [1999]. Integrating Craft Materials and Computation. *Proceedings of Creativity and Cognition 3*, pp. 50-56.

3. Eisenberg, M.; Wrensch, T.; and Blauvelt, G. [1999] Geometry-Specific Languages and Their Interfaces. University of Colorado Department of Computer Science Technical Report CU-CS-886-99.

4. Eisenberg, M. and Eisenberg, Ann N. [1999] Middle Tech: Blurring the Division Between High and Low Tech in Education. In A. Druin, ed. *The Design of Children's Technology*, San Francisco: Morgan Kaufmann, pp. 244-273.

5. Feiner, S.; MacIntyre, B.; and Seligmann, D. [1993] Knowledge-based Augmented Reality. *Communications of the ACM*, 36:7, pp. 52-62.

6. Gershenfeld, N. [1999] *When Things Begin to Think*. New York: Henry Holt.

7. Ishii, H. and Ullmer, B. [1997] Tangible Bits: Towards Seamless Interfaces between People, Bits, and Atoms. *Proceedings of CHI '97*, pp. 234-241.

8. Mozer, M. C. [1999]. An intelligent environment must be adaptive. *IEEE Intelligent Systems and their Applications*, 14(2), 11-13.

9. Resnick, M.; Martin, F.; Sargent, R.; and Silverman, B. [1996] Programmable Bricks: Toys to Think With. *IBM Systems Journal*, 35:3, pp. 443-452.

10. Resnick, M. *et al*. [1998] Digital Manipulatives: New Toys to Think With. *Proceedings of CHI '98*, pp. 281-287.

11. Spooner, P. [1986] Spooner's Moving Animals. New York: Harry N. Abrams, Inc.

12. Umaschi, M. [1997] Soft Toys with Computer Hearts. *Proceedings of CHI '97*, Atlanta, pp. 20-21.

13. Wrensch, T. and Eisenberg, M. [1998] "The Programmable Hinge: Toward Computationally Enhanced Crafts" *Proceedings of UIST 98*, San Francisco, November, pp. 89-96.