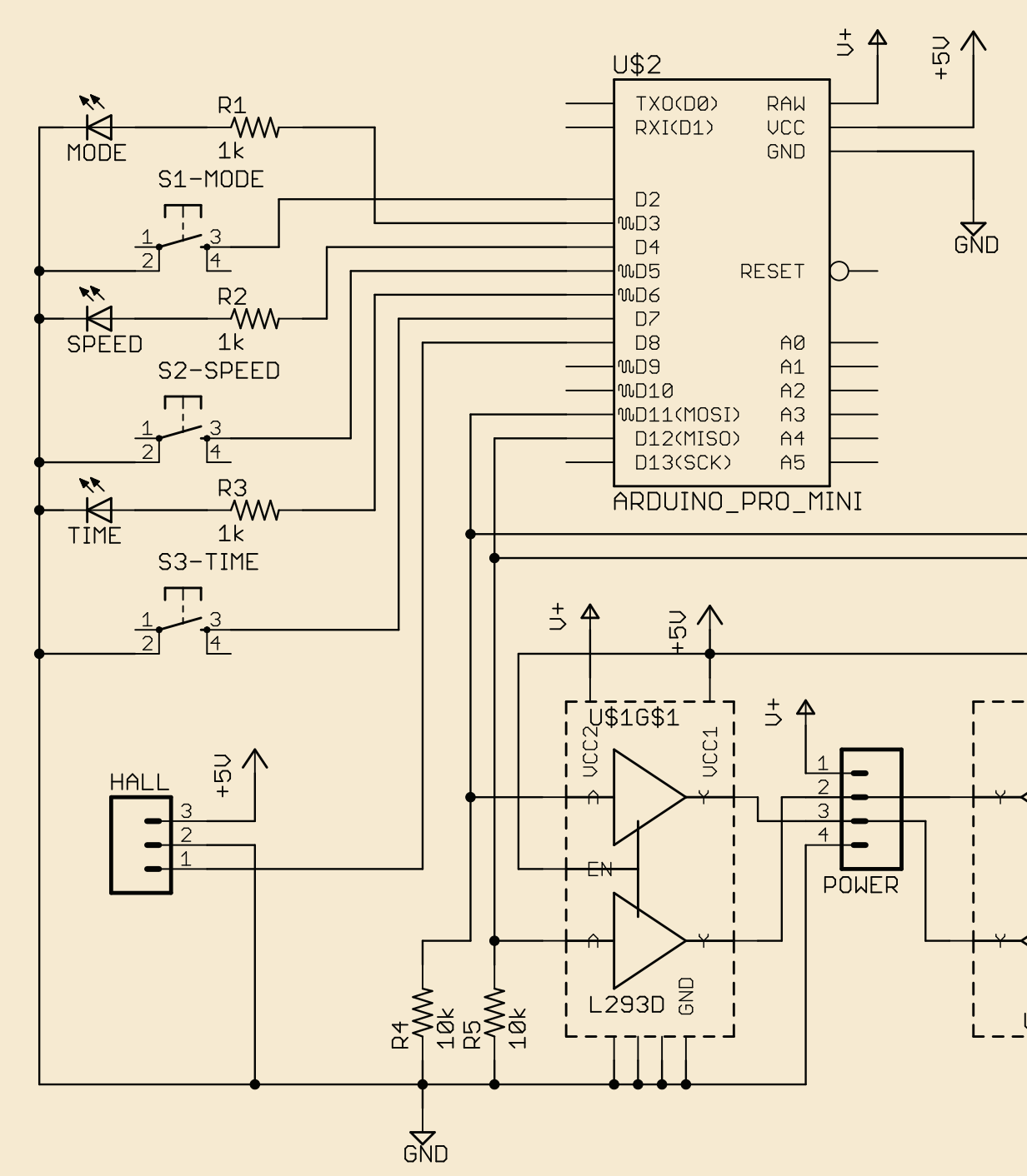
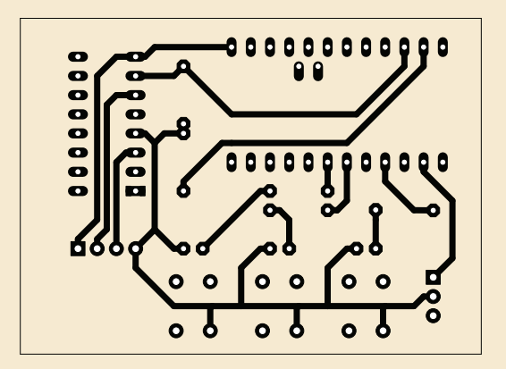
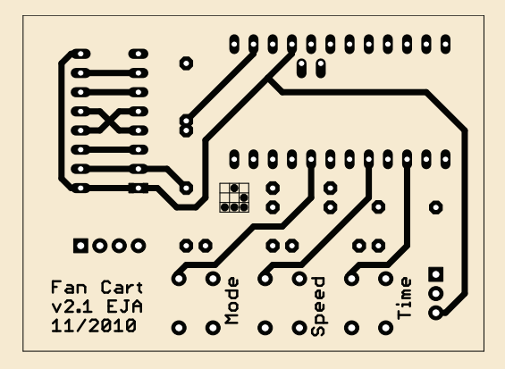
Programmable Fan Cart

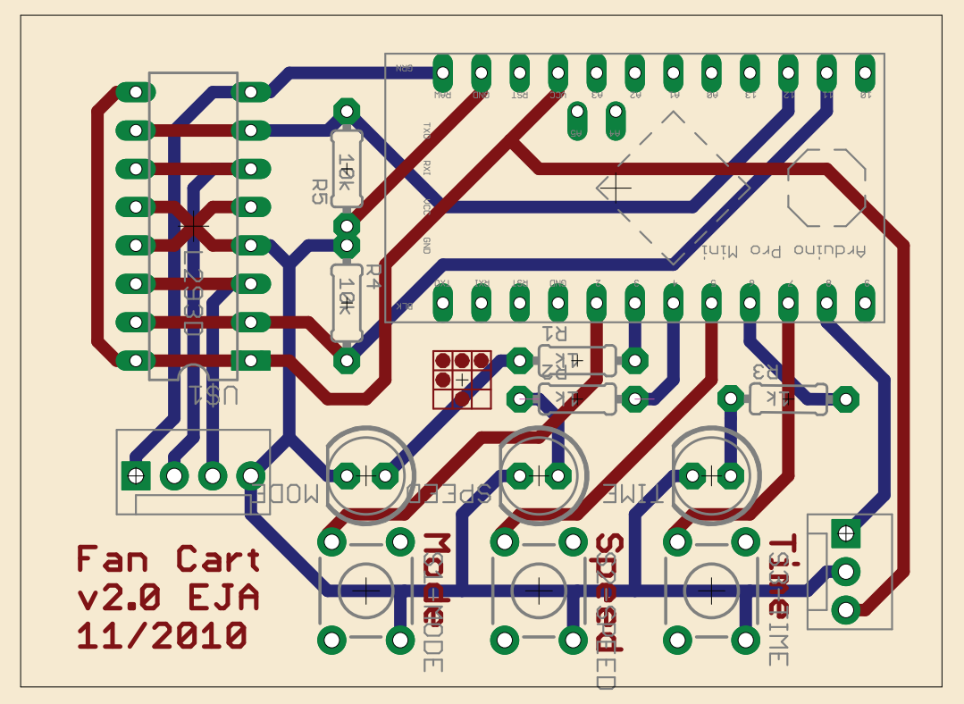
Eric Ayars
California State University, Chico
ayars@mailaps.org

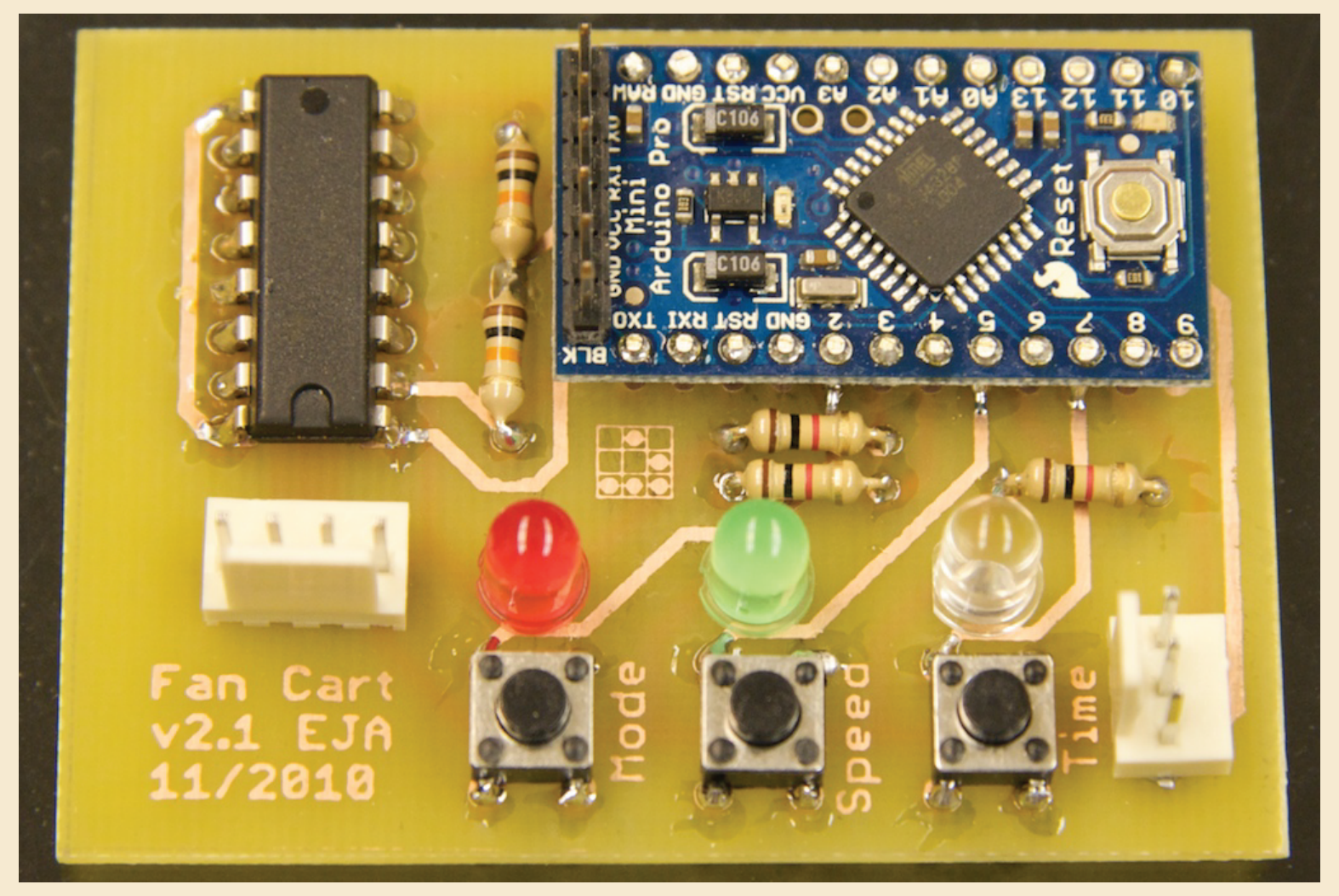


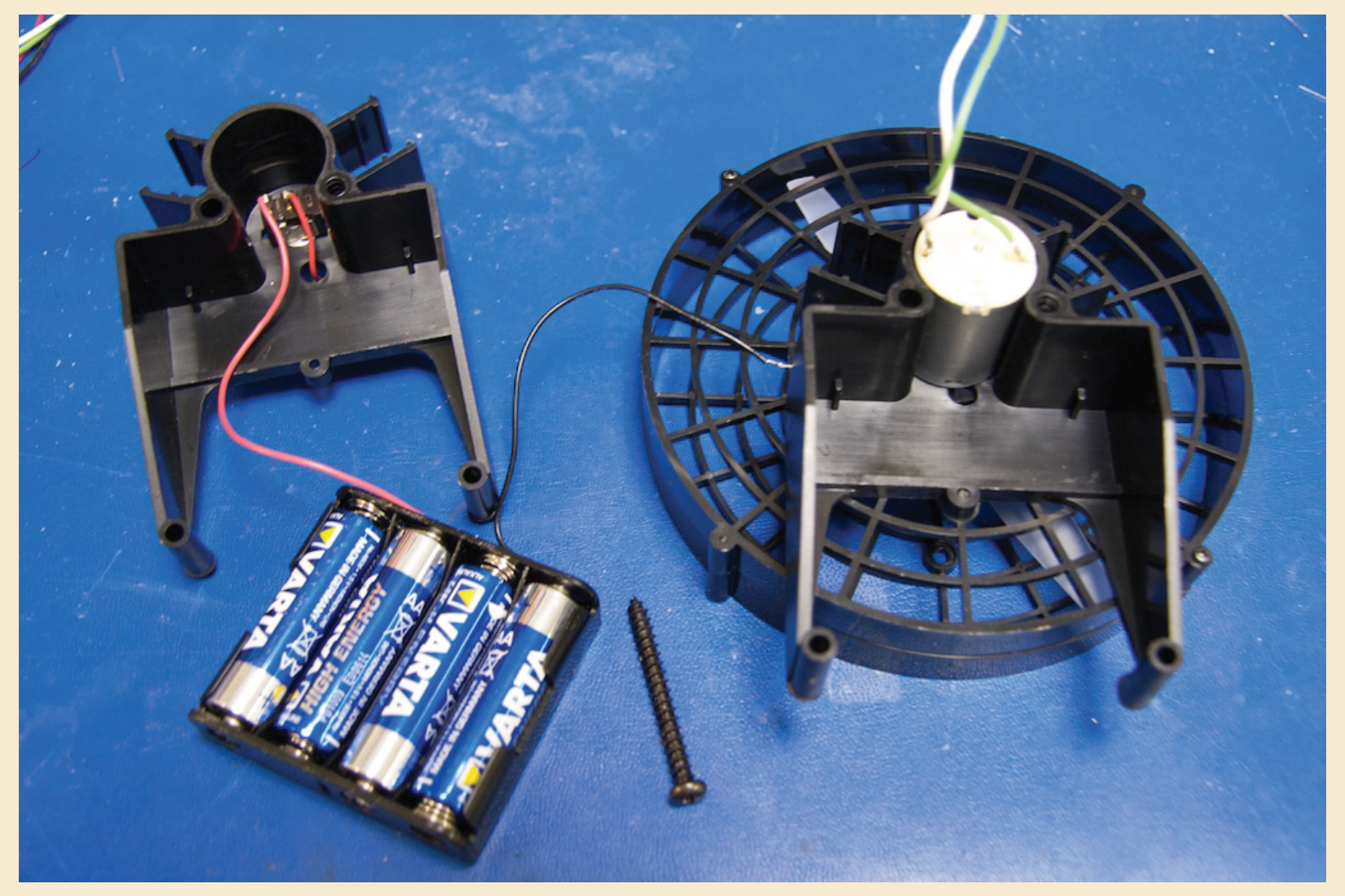
EagleCAD (www.cadsoftusa.com) is a very powerful (and free) tool for making circuit layouts. I highly recommend it! What you see here was designed with EagleCAD, printed on "toner transfer system" paper from circuitspecialists.com, ironed onto copper-clad board, then etched using a ferric chloride solution.

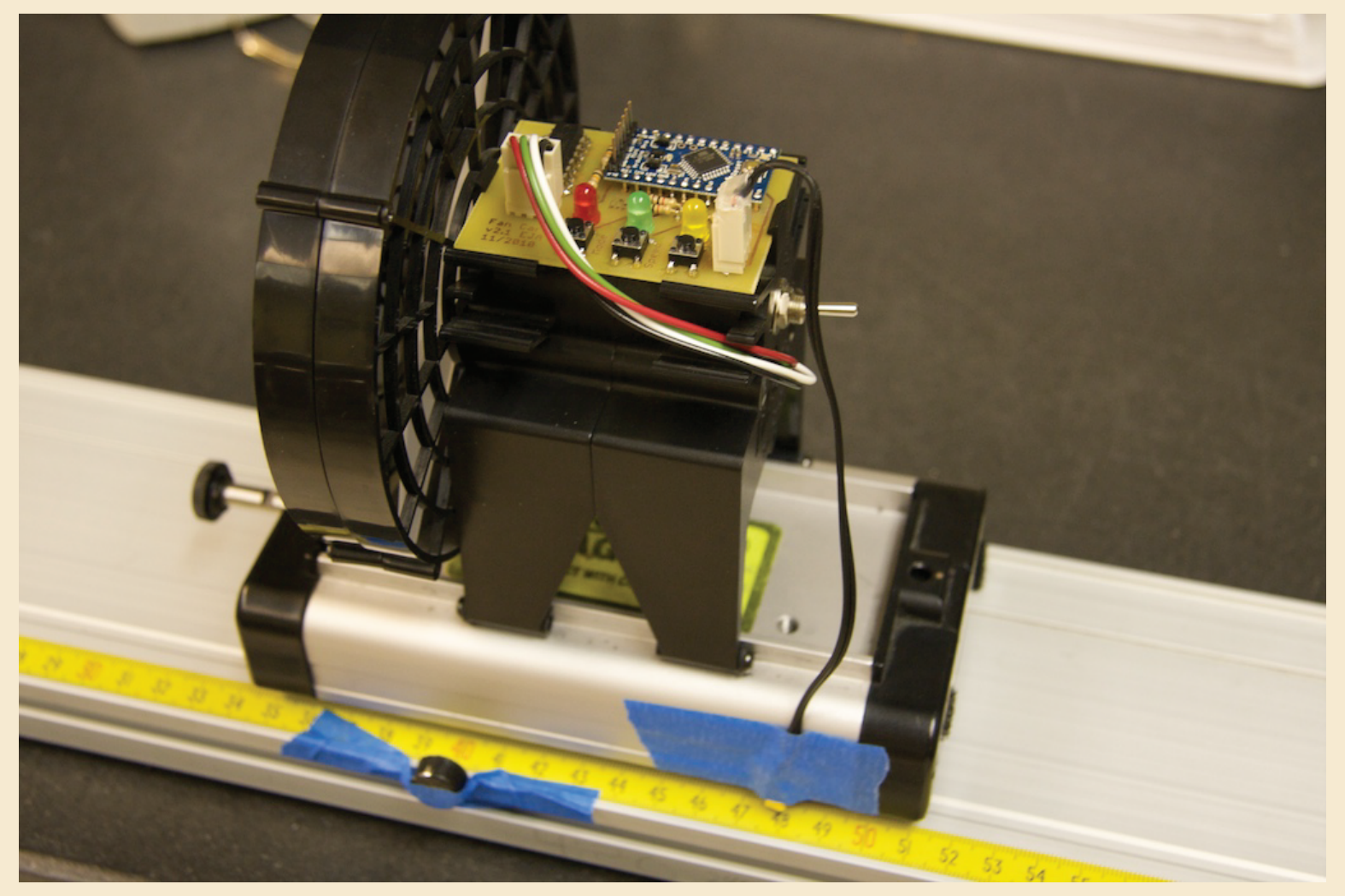












Fancarts: They turn on, they provide an approximately constant force, and they turn off.

But why stop there? With an easy-to-program Arduino microcontroller, a Hall switch, and a few inexpensive electronic components, it's possible to modify a fancart to give complete control over where it turns on or off, how long it stays on or off, and how fast it spins in between. You can even control the fan direction!

By putting small magnets on the track to trigger the Hall switch, it's possible to have a constant force over a certain distance. Measurements of the cart speed before and after the region where the fan is on show that the change in kinetic energy is equal to the force times the distance, verifying the work-energy theorem.

The same microcontroller can turn the fan on at one magnet, then turn the fan off some time later. Measurements of the cart's speed before and after this impulse show that the change in momentum is equal to the force times the time, verifying the impulse-momentum theorem.

And since the microcontroller can have complete control over everything to do with the fan, it's possible to vary the fan direction as well as speed. This level of control opens up all sorts of new experimental opportunities such as an "anharmonic oscillator fan cart".

The hardware for this apparatus (shown on the left) can be built for less than $30 per unit. Here I used EagleCAD and toner-transfer paper to create a 2-sided circuitboard, but point-to-point soldering on proto-board works also.

The firmware to install on the microcontroller (outlined on the right) can be uploaded with free open-source tools using any Macintosh, Linux, or even Windows computer. The language used is just c/c++ with some added libraries, so it's easy to modify the firmware so that your units do exactly what you want for your teaching purposes.

PLEASE TOUCH!

This cart has the full version of the program on the right loaded into its Arduino, so it can do all three of the "modes" described above. Try them!

• Turn the switch on. The three indicator lights will blink to tell you that the mode is set to 1 (one blink) the speed is set to 2 (two blinks) and the time is 2 seconds (two blinks). Finally all three lights will illuminate, letting you know that it is ready to go.

• Put the cart on the track and roll it past the magnets. In mode 1, the first magnet will turn the fan on and the second will turn it off.

• Try changing the parameters. Pressing the speed button will cause the green light to blink twice, indicating that the speed is currently set at 2. To change the speed, hold the speed button for more than two seconds and then release it. The green light will go out, indicating that the device is in speed adjustment mode. Subsequent presses of the speed button will cycle through the speed options: 3, then 1, then 2... Set it where you want, then press any other button to exit adjustment mode.

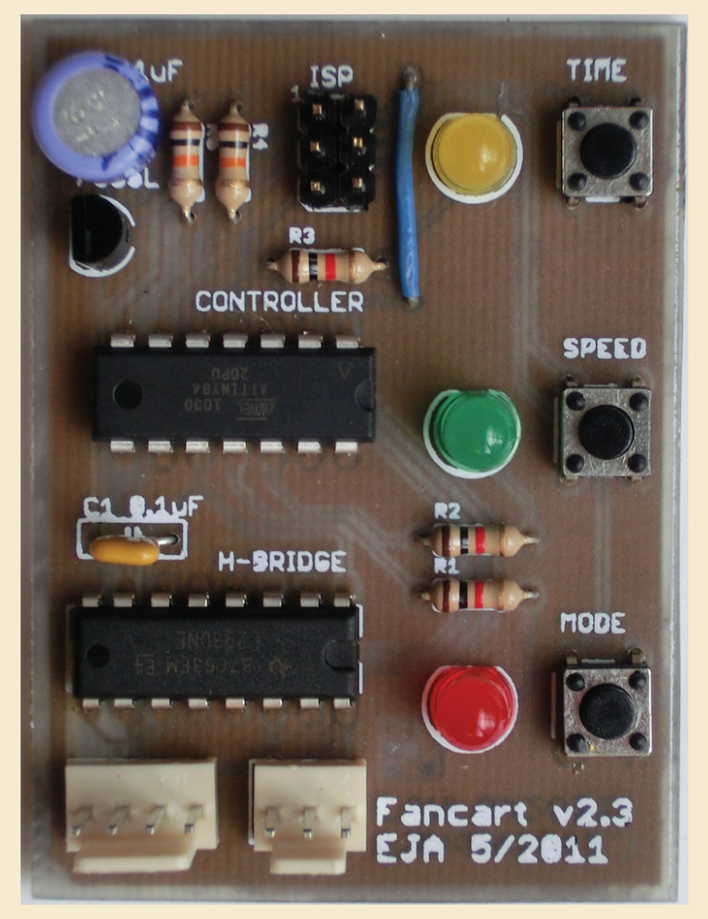
• Change the mode and/or time by the same method. Mode 2 causes the fan to run for a set time after detecting a magnet. Mode 3 causes the fan to change direction each time a magnet is detected.

• Pressing any button while the fan is running will cause the fan to turn off.

One more thing: this board is designed to fit the PASCO ME-9491 fan cart accessory, but it can work with ANY small fan motor... or any other DC device that can take a Pulse-Width Modulated (PWM) input! It's safe with supply voltages up to 15 Volts, and can source up to 1.2 Amps. There's nothing special about fan cart motors – it could just as easily be used to drive a Peltier heater, for example.

There's nothing special about the Hall effect switch, either: Any component that takes a 5V supply and returns a logic-level output would work fine.

Want to build this for less? If you're comfortable with (or willing to learn) in-system programming, it's possible to build a functionally equivalent circuit using an ATtiny84 and an LM78L05 voltage regulator instead of the Arduino board. This cuts the component cost roughly in half, while still allowing complete control over all aspects of the fan behavior. Contact me for an EagleCAD layout of this circuit if you're interested.



```
/*  Fan cart driver version 2.1
    Eric Ayars 5/24/11
    What follows is an abbreviated listing of the code. Full code is available at
    http://physics.csuchico.edu/~eayars/code/fancart.zip
*/

#define MOTORPWM 11
#define MOTORDIR 12
    ... etc ...

byte UserInput = 0;
boolean ProgramState = false;
long int CurrentTime;
byte Speed = 2; // Ranges from 1 to 3 (0 not used).
byte PowerLevel[] = {0, 130, 200, 255};
    ... etc ...

byte ButtonDown() {
    // Returns 3-bit indicator of which buttons are currently down.
    // bit 0 - Mode, bit 1 - Speed, bit 2 - Time
        ...
}

boolean MagnetSensed() {
    // This returns true or false depending on whether the cart just passed a magnet.
        ...
}

void AnnounceReady() {
    // Turns on all three lights to indicate that the fan is ready to go.
        ...
}

void ShowStatus(byte Light, byte Blinks) {
    // Blinks a given light some number of times. Used to show how things are set.
        ...
}

void MotorOn(byte howFast) {
    // turns motor on in direction set by global MotorDirection.
        ...
}

void MotorOff() {
    // turns motor off.
        ...
}

void ProgramMode(byte ProgramThis) {
    // Sets mode, speed, time...
    boolean State = false;
    boolean Exit = false;

    if (ProgramThis & B0001) {  // Program the Mode.
        // Turn mode light off and the other two lights on.
        digitalWrite(MODELIGHT, LOW);
        digitalWrite(SPEEDLIGHT, HIGH);
        digitalWrite(TIMELIGHT, HIGH);

        while (!Exit) {
            delay(DEBOUNCE);
            if (ButtonDown() == 1) {
                // Mode button has been pressed. Increment mode.
                Mode += 1;
                if (Mode > 3) {
                    Mode = 1;
                }
                ShowStatus(MODELIGHT, Mode);
            }
            UserInput = ButtonDown();
            if (UserInput > 1) {
                Exit = true;
            }
        }
    }
        ... Similar code for changing speed and time omitted ...
}

void DoMode(byte Mode) {
      // This routine runs the desired mode until a button is pressed.

      AnnounceReady();
      MotorDirection = false; // false is forward, true is backwards.

      while (!ButtonDown()) {
            // No button has been pressed, so keep going.

            // Mode 1: the fan turns on and off on alternating magnets.
            if (Mode==1) {
                  MotorDirection = false;
                  if (MagnetSensed()) {
                        if (MotorState) {
                              // Motor is currently on. Turn it off.
                              MotorOff();
                        } else {
                              // Motor is currently off. Turn it on.
                              MotorOn(PowerLevel[Speed]);
                        }
                  }
            }

            // Mode 2: the fan turns on at the first magnet, then turns off a set time later.
            if (Mode==2) {
                  MotorDirection = false;
                  if (MagnetSensed()) {
                        // Turn the motor on...
                        MotorOn(PowerLevel[Speed]);
                        // Wait a given time...
                        delay(TimeLevel[Time]);
                        // And turn the motor off.
                        MotorOff();
                  }
            }

            // Mode 3: the motor turns on when it first sees a magnet, then reverses each time it passes a magnet.
            if (Mode==3) {
                  if (MagnetSensed()) {
                        // Reverse the motor...
                        MotorDirection = !MotorDirection;
                        // And turn it on in the new direction.
                        MotorOn(PowerLevel[Speed]);
                  }
            }

            // Add more modes here if desired!

      }

      // Button has been pressed, so...
      MotorOff();
      MotorDirection = false;
}


void setup() {
    /*
     * The setup() function runs ONCE when power is first applied.
     */

    // Set appropriate pins to output.
    pinMode(POWERINDICATOR, OUTPUT);
        ... etc ...

    // Set appropriate pins to input, with pull-up resistors as needed.
    pinMode(MODESWITCH, INPUT);
    digitalWrite(MODESWITCH, HIGH);     // setting an input to HIGH turns on an internal 20k pull-up resistor.
        ... etc ...

    // Turn off motor just in case.
    MotorOff();

    // Indicate Status.
    ShowStatus(MODELIGHT, Mode);
    ShowStatus(SPEEDLIGHT, Speed);
    ShowStatus(TIMELIGHT, Time);

    // Indicate ready-to-go
    AnnounceReady();
}

void loop() {
    /*
     *  The loop() function runs repeatedly after the setup() function runs once. This loop checks for user input,
     *  passes input (if any) to the "ProgramMode" routine, then runs "DoMode". That's it!
     */

    // Check to see whether the user wants anything
    UserInput = ButtonDown();
    CurrentTime = millis(); // Check what time it is

    while (ButtonDown()) {
        // Wait until button is up to decide what to do.
        delay(10);
    }
    if (millis()-CurrentTime > 2000) {
        // Button was held for more than 2 seconds, so program the device.
        ProgramMode(UserInput);
    } else {
        if (UserInput & B0001) {
            // The mode button was pressed.
            ShowStatus(MODELIGHT, Mode);
        }
        if (UserInput & B0010) {
            // The speed button was pressed.
            ShowStatus(SPEEDLIGHT, Speed);
        }
        if (UserInput & B0100) {
            // The Time button was pressed.
            ShowStatus(TIMELIGHT, Time);
        }
    }

    // Here's where it actually does something. Whatever the mode is...
    DoMode(Mode);
}
```

All this code is just c/c++, so it's easy to modify to fit your particular needs. Once a program is uploaded to the Arduino, that program runs automatically any time the power is turned on... and you can upload new code to the Arduino as many times as you need!

You can make these modes do anything you like! Linearly-ramped fan speed? No problem:

```
    for (byte j=50;j<250;j+=5) {
        MotorOn(j);
        delay(50);
    }
```

Gaussian fan-speed pulse? Trickier, but do-able...
Have the fan pitch play the theme from the final movement of Beethoven's 3rd symphony?
Uh... what's the teaching load at your school, anyway?