NVE CORPORATION

AG935-07E
“Blinky” Angle Sensor Demo Board

SB-00-072

Kit Overview

Demonstration Kit Includes

- A 5-inch by 5-inch circuit board with:
  - An AAT003 Angle Sensor
  - 60 multicolor smart LEDs (6° spacing) indicate rotation angle
  - An onboard preprogrammed ATtiny microcontroller
  - PWM angle output (8-bit resolution)
  - Factory calibrated; optional field calibration
- Split-pole magnet
- Magnet locating fixture
- 5-volt wall-mount power supply module

AAT003-10E Angle Sensor Features

- Tunneling Magnetoresistance (TMR) technology
- Zero to 5.5 V supply range
- 40 kΩ typical bridge resistance for low power
- 200 mV/V typical output signal
- 1.5% maximum nonsinusoidality error
- Wide sensor-magnet airgap tolerance
- Sine and cosine outputs for direction detection
- Ultraminiature 2.5 mm x 2.5 mm x 0.8 mm TDFN6 package

AAT-Series Sensor Applications

- Rotary encoders
- Motor shaft position sensors
- Internet-of-Things sensor nodes

Available AAT-Series Angle Sensors

Part Number	**Configuration**	**Typ. Output** (ea. output; p-p)	**Required Field**	**Typ. Device Resistance**
AAT001-10E	Half-bridge	200 mV/V	30 Oe	1.25 MΩ
AAT003-10E	Half-bridge	200 mV/V	30 Oe	40 KΩ
AAT006-10E	Half-bridge	200 mV/V	15 Oe	1.5 MΩ
AAT009-10E	Half-bridge	200 mV/V	30 Oe	6 MΩ
AAT101-10E	Full-bridge	400 mV/V	30 Oe	625 kΩ

Visit **www.nve.com** for complete product specifications.

Demonstration Board Operation

⇒ Connect the five-volt power supply.

⇒ Place the magnet in the Plexiglas pocket.

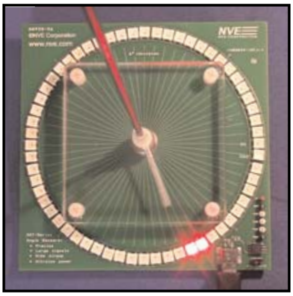
⇒ Rotate the magnet and observe the LEDs. Red LEDs indicate clockwise rotation; green is for counterclockwise; and white is for stopped.

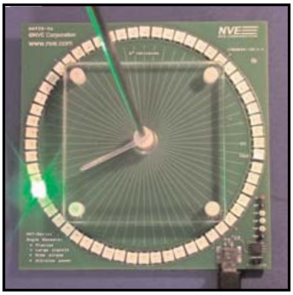
⇒ The SIN and COS raw output test points can be measured with a voltmeter or oscilloscope.

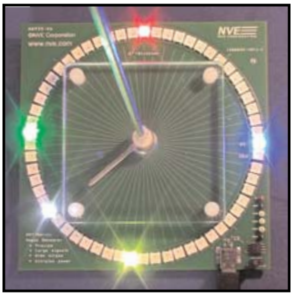
⇒ The calibrated PWM output can be measured with a voltmeter (0 to 360° = 0 to V_{CC}).

⇒ The board is factory calibrated, but it can be recalibrated as follows:

- Install a jumper on the "calibrate" header.
- Cycle the power.
- Reciprocating blue "arrows" indicate the board is waiting for the magnet to turn.
- Rotate the magnet clockwise and counterclockwise for about five seconds (be sure to cover at least the entire 360° of rotation).
- Four colored LEDs (red, green, yellow, and blue) will light as parameters are acquired for each quadrant.
- New calibration constants are stored in the microcontroller's EEPROM, and the board will begin operating normally using the new constants.
- The calibrate jumper can be removed to bypass the calibration routine on subsequent power-ups.


Clockwise rotation.


Counterclockwise rotation.

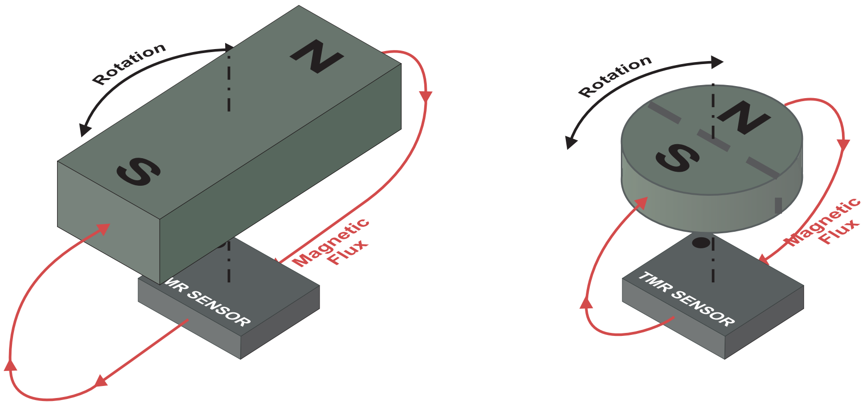

Calibration.

Visit **nve.com** or **YouTube/NveCorporation** for demonstrations.

Sensor Principles of Operation

The heart of AAT sensors is arrays of four unique Tunneling Magnetoresistance (TMR) elements, one in each quadrant. TMR technology enables low power and miniaturization, making the sensors ideal for battery operation.

In a typical configuration, an external magnet provides a saturating magnetic field in the plane of the sensor, as illustrated below for a bar magnet and a diametrically-magnetized disk magnet:



The sensor contains four sensing resistors at 90 degree intervals. The resistors are connected as two half-bridges, providing the sine and cosine voltage outputs. For each half bridge, the resistance of one element increases and the other decreases as the field rotates. Thus the bridge resistance, device resistance, and output impedances remain constant with rotation.

Outputs are ratiometric with the supply, and inherently resistive for easy filtering.

Angle Calculation Algorithms

An obvious way to calculate the angle is the inverse sine or cosine of either output as shown in the pseudocode below, but such calculations depend on the particular sensitivity of each part:

```
Angle = asin(float(AnalogRead(3)-512)/200);
```

[not very accurate]

Since it uses the ratio of the sensor outputs, arctangent provides more accuracy. Arctangent cancels power supply variations, doesn’t need scaling, and takes advantages of the matching of the two sensor outputs:

```
Angle=atan(float((AnalogRead(3)-512)/float(AnalogRead(2)-512);
```

[no quadrature]

The single-variable `atan` only provides angles for half the unit circle, however, since the operand is positive when sine and cosine are either both positive or both negative. Quadrature has to be determined separately if needed.

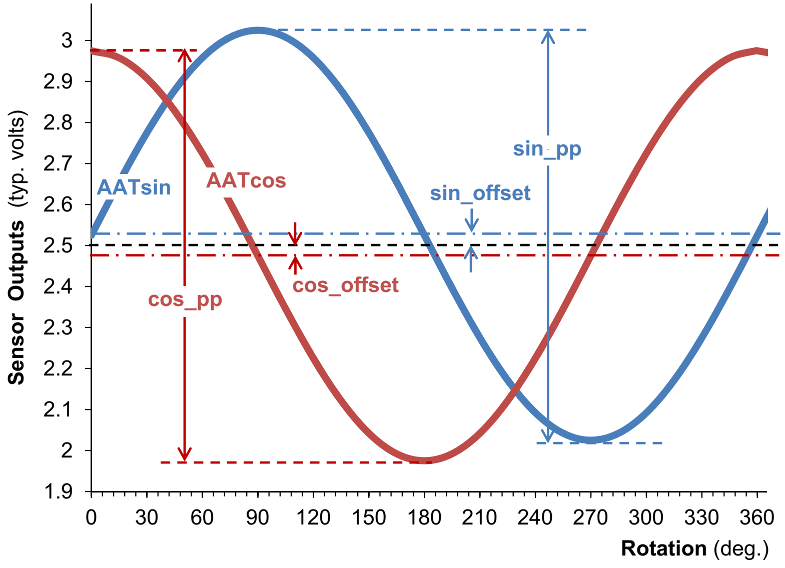
So the calculation used in this evaluation board is based on a two-variable arctangent, which provides a full 360-degree angle range and inherent scaling:

```
//Read uncalibrated angle (direct ADC access to use less
memory than Arduino "analogRead")
int readAngle () {
ADMUX = 3; //Read sensor
AATsin = getADC();
ADMUX = 2;
AATcos = getADC();
return (atan2(float(AATsin-512), float(AATcos-
512))/pi+1)*30; //Uncalibrated angle
}
```

The function above also converts angular radians from the inverse trigonometric function to an integer from zero to 59 for the 60 LEDs in the demonstration.

Calibration and Scaling Algorithms

Calibration is often unnecessary, but to maximize accuracy, this board implements two-parameter (sensitivity and offset) linear calibration on each of the two sensor outputs. As shown in the graph below, calibration parameters are calculated from the minimum and maximum of each output:



Offset correction parameters use the sensor's average outputs as shown in this pseudocode:

```
AATsin  -=  (AATsinmax + AATsinmin) / 2;
AATcos  -=  (AATcosmax + AATcosmin) / 2;
```

Sensitivity is calibrated using the sensor's peak-to-peak amplitudes over its rotation, and the parameter scales the sensor outputs:

```
angle  =  atan2(AATsin / (AATsinmax-AATsinmin),
                AATcos / (AATcosmax-AATcosmin);
```

Digital Filtering and PWM Output

Digital Filtering

This board implements digital filtering to damp mechanical chatter or electrical noise. Filtering is usually unnecessary, but this board represents a challenging noise environment with a combination of noise sources and an inexpensive, lightly-filtered modular power supply. Board noise sources include high-current, clocked LEDs and a clocked microcontroller.

There are many digital filtering algorithms; this board uses a simple first-order, running average algorithm where the value is updated from the old value using a weighting factor *m*:

```
AATfiltered = (1-1/m)*AATfiltered + AATunfiltered/m;
```

This algorithm has an approximately first-order response:

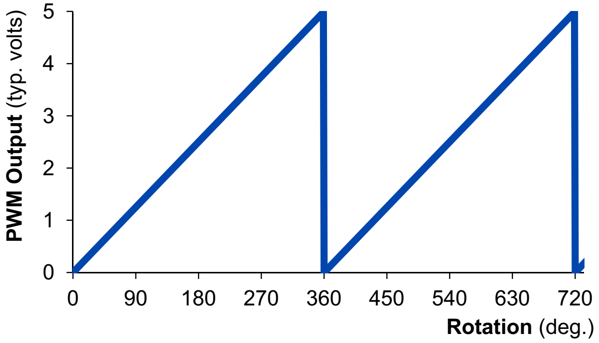
$$f_c = f_s / (2\pi\ m)$$

Where f_c is the cutoff frequency; f_s is the sample rate, and *m* is the filter constant. Digital filtering is generally applied to the sensor outputs rather than the calculated angle because the angle has a discontinuity from 360 to zero degrees.

Since the sensor has resistive outputs, capacitors can also be added to the outputs for filtering. Since we have a microcontroller, however, digital filtering is more flexible, lower cost, and lower parts count.

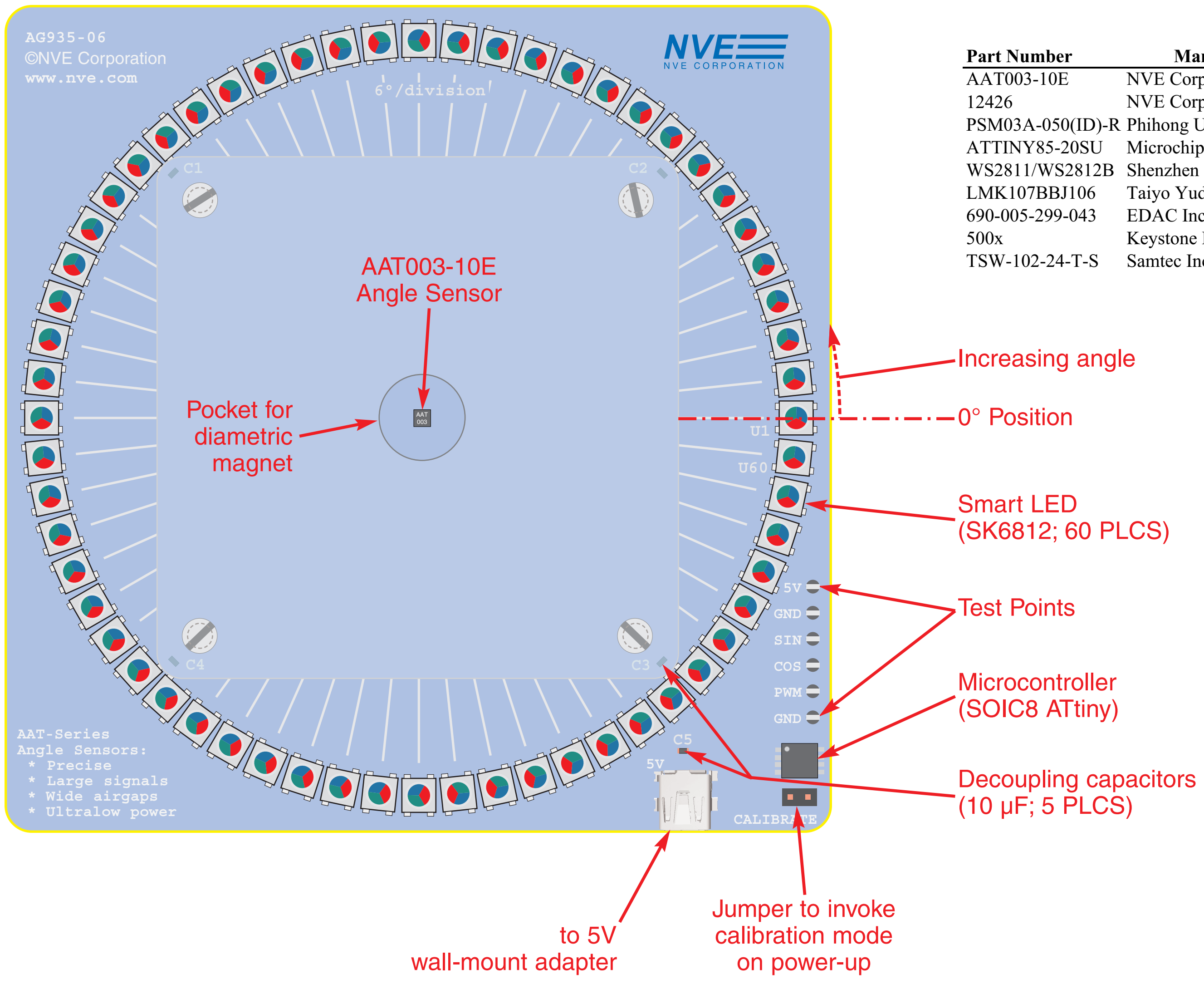
PWM Output

The PWM on this board uses a simple Arduino AnalogWrite function. The output is rationmetric with supply, so the zero to V_{CC} output range corrresponds to zero to 360 degrees, as shown in the figure at right.



The ATtiny processor provides 8 bits of resolution.

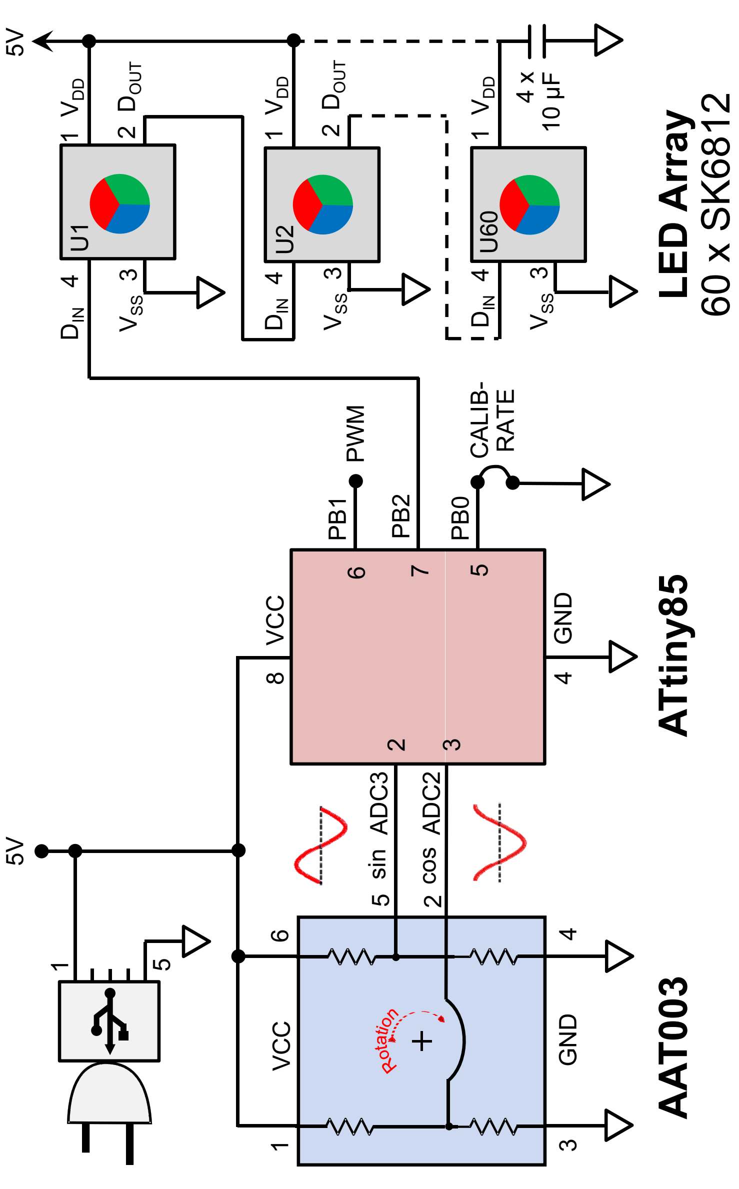
Evaluation Board Layout



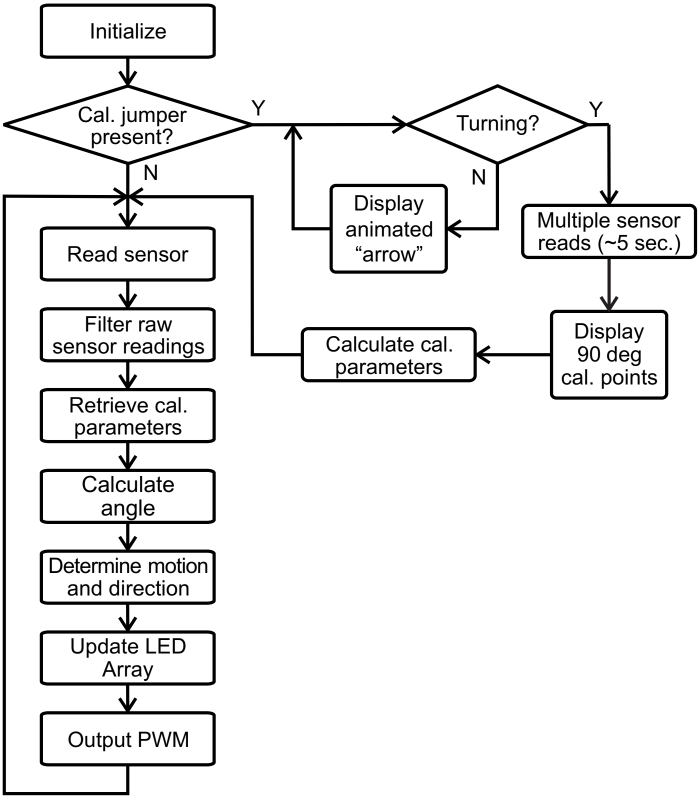
Bill of Materials

Part Number	Manufacturer	Qty	Description
AAT003-10E	NVE Corporation	1	ANGLE SENSOR, 40 KOHM, TDFN6
12426	NVE Corporation	1	SPLIT-POLE ROUND HORSESHOE MAGNET
PSM03A-050(ID)-R	Phihong USA	1	AC/DC WALL MOUNT ADAPTER 5V 3W
ATTINY85-20SU	Microchip Technology	1	IC MCU 8BIT 8KB FLASH 8SOIC
WS2811/WS2812B	Shenzhen LED Color Opto	60	DIGITAL RGB LED 5050 SMD
LMK107BBJ106	Taiyo Yuden	5	CAP CER 10uF 10V X5R 0603
690-005-299-043	EDAC Inc.	1	CONN MINI USB RCPT RA TYPE B SMD
500x	Keystone Electronics	6	TEST POINT PC MINI .040"D
TSW-102-24-T-S	Samtec Inc.	1	.025" SQ. TERMINAL STRIPS

Schematic


5V
1 V_{DD}
2 D_{OUT}
U1
4 D_{IN}
3 V_{SS}
1 V_{DD}
2 D_{OUT}
U2
4 D_{IN}
3 V_{SS}
1 V_{DD}
4 x 10 µF
U60
4 D_{IN}
3 V_{SS}
LED Array
60 x SK6812
PWM
PB1
PB2
PB0
CALIB-RATE
6
7
5
VCC
8
GND
4
2
3
ATtiny85
5 sin ADC3
2 cos ADC2
5V
1
5
6
VCC
Rotation
GND
4
1
3
AAT003

Flowchart



Software available on: ***https://github.com/NveCorporation***

Software

```
/*********************************************************************************************
AG935 demo board with an AAT003 sensor connected to a 60 smart-LED circular array via an
ATtiny85. Code written in Arduino IDE targeted at "Adafruit Trinket (ATtiny85 @ 8 MHz),"
and is portable to other Arduino boards (add delays per comments for faster processors).
Sensor "SIN" output to PB3; "COS" to PB4; array input on PB2; PWM output on PB1
(ratiometric; 0-Vcc = 0-360 deg.). Active low "CALIBRATE" jumper on PB0.
3 bytes RAM/LED ==> 120 of 512 bytes used in an ATtiny85.
Program uses ~5K flash out of 8K (5.4K available with Arduino bootloader).
Source code available on github.com.                                          Rev. 6/12/18
*********************************************************************************************/
#include <Adafruit_NeoPixel.h> //Use NeoPixel Arduino routines for LEDs for convenience
unsigned char EE_read(unsigned char addr);
void EE_write(unsigned char addr, unsigned char ucData);
int getADC();
unsigned char readAngle();

//60 LEDs on PB2; 800 Khz LEDs:
Adafruit_NeoPixel strip = Adafruit_NeoPixel(60, 2, NEO_GRB + NEO_KHZ800);

int AATsin; //AAT signals
int AATcos;
float sinFiltered = 0.0; //Digitally filtered AAT signals
float cosFiltered = 0.0;
const float pi = 3.14159;
int angle; //Uncalibrated angle (0-59)
int angleOld; //Previous angle
bool dir = 0; //Rotation direction (cw = 1; ccw = 0)
unsigned int cycleCounter = 0; //Number of program iterations since angle changed
const int stopCycles = 25; //Sensor cycles stopped before turning off direction colors
const int arrowCycles = 4; //Loop cycles to update arrow (inverse animation speed)
unsigned int arrowPos = 0; //Arrow position away from start
char arrowPixel; //Arrow pixel animation position
unsigned char arrowPixelBrightness;
unsigned char i; //Arrow pixel index
char j; //Indicates cw or ccw arrow
const unsigned char brightness = 2; //Brightness (1-8)
const unsigned char m = 2; //Filter constant; Fc=Fsample/(m*2*pi); Fsample=~250/s

//Sensor min and max outputs (actual values determined in calibration routine)
unsigned char AATsinmin = 63; //Defaults to +-65 mV/V min amplitude offset by 128
unsigned char AATsinmax = 193;
unsigned char AATcosmin = 63;
unsigned char AATcosmax = 193;

//Uncalibrated angles where mix and max occur
unsigned char angleSinMin;
unsigned char angleSinMax;
unsigned char angleCosMin;
unsigned char angleCosMax;

//EEPROM address pointers
const unsigned char sin_offset_addr = 0; //Sin offset + 128
const unsigned char cos_offset_addr = 1; //Cos offset + 128
const unsigned char sin_pp_addr = 2; //Sin pk-pk amplitude (255 = 250 mV/V max)
const unsigned char cos_pp_addr = 3; //Cos pk-pk amplitude
```

```
void setup() {
//Full-speed clock needed for ATtiny to interface to smart LEDs
//Not needed with faster processors
  CLKPR = 0x80; //Enable changing the internal clock
  CLKPR = 0; //Set full speed internal clock

  pinMode(0, INPUT); //Active low "CALIBRATE" jumper on PB0
  digitalWrite(0, HIGH); //Turn on pull-up

  //ADC setup (avoided Aruino routines to save memory)
  ADCSRA &= ~(_BV(ADATE) | _BV(ADIE)); //Clear ADC auto trigger and interrupt enable
  ADCSRA |= _BV(ADEN); //Enable ADC
  strip.begin();
  strip.show();

  //Read uncalibrated angle
  angle = readAngle();
  angleOld = angle;
  if (!digitalRead(0)) { //Jumper in place; invoke calibration routine
    while (angle == angleOld) { //Reciprocating cw/ccw arrows until sensor turns
      angleOld = angle;
      strip.clear(); //Reset the LED array
      for (i = 1; i<30; i++) {
        arrowPixel = 45+(15-i)*(1-dir*2); //Arrows centered at LED 45 (12:00)
        //Brightness profile simulates an arrow
        arrowPixelBrightness = 30-arrowPos/arrowCycles - i;
        //Arrow length 3
        arrowPixelBrightness *= (arrowPixelBrightness > 0)&&(arrowPixelBrightness < 4);
        //Cube for nonlinear brightness vs. position; scale
        arrowPixelBrightness *= arrowPixelBrightness*arrowPixelBrightness*brightness;
        strip.setPixelColor(arrowPixel,0,0,arrowPixelBrightness); //Animated blue arrow
      } //Finish setting arrow pixels (add delay here for faster processors)
      strip.show();
      arrowPos++;
      if (arrowPos == arrowCycles*27) {
        arrowPos = 0; //Wrap arrow position at 11
        dir = !dir; } //Reverse arrow
      angle = readAngle(); }
    //Find sensor output minimums and maximums
    for (cycleCounter = 1; cycleCounter < 1500; cycleCounter++) { //1500 loops (~5 sec)
    strip.clear(); //Reset the LED array
      angle = readAngle();
      AATsin -=384; //Subtract minimim outputs to ensure 8-bit (0-256), positive range
      AATcos -=384;
      if (AATsin < AATsinmin) {
        AATsinmin = AATsin;
        angleSinMin = angle;  }
      if (AATsin > AATsinmax) {
        AATsinmax = AATsin;
        angleSinMax = angle;  }
      if (AATcos < AATcosmin) {
        AATcosmin = AATcos;
        angleCosMin = angle;  }
      if (AATcos > AATcosmax) {
        AATcosmax = AATcos;
        angleCosMax = angle;  }

      //Sensor position=White; calibration points in color
      strip.setPixelColor(angle, 16*brightness, 16*brightness, 13*brightness);
      strip.setPixelColor(angleSinMin, 32*brightness, 32*brightness, 0); //SINmin=Yellow
      strip.setPixelColor(angleSinMax, 32*brightness, 0, 0); //SINmax=Red
      strip.setPixelColor(angleCosMin, 0, 0, 32*brightness); //COSmin=Blue
      strip.setPixelColor(angleCosMax, 0, 32*brightness, 0); //COSmax=Green
      strip.show();
    }
//Store calibration parameters
    EE_write(sin_offset_addr,(AATsinmax+AATsinmin)/2); //Offsets = average outputs
    EE_write(cos_offset_addr,(AATcosmax+AATcosmin)/2);
    EE_write(sin_pp_addr, AATsinmax - AATsinmin); //pk-pk amplitudes for calibration
    EE_write(cos_pp_addr, AATcosmax - AATcosmin);
  } //End CALIBRATE routine
  strip.clear();
  cycleCounter = stopCycles;
} //End setup
```

```
void loop() {
  readAngle();

//Offset correction using EEPROM parameters; add back 384 previously subtracted
AATsin -= EE_read(sin_offset_addr)+384;
AATcos -= EE_read(cos_offset_addr)+384;

//Digital filters--> Fc=Fsample/(m*2*pi); Fsample=~250/s
sinFiltered += (AATsin-sinFiltered)/m;
cosFiltered += (AATcos-cosFiltered)/m;

//Calculate calibrated angle; scale for 15360 = 360 degrees
angle=(atan2(sinFiltered/EE_read(sin_pp_addr),cosFiltered/EE_read(cos_pp_addr))/pi+1)*7679;
analogWrite(1, angle/60); //Scale 0-255; output PWM on PB1 (Arduino Write for simplicity)
  angle = angle/256; //Scale for 0-59 for LED array
  if (angle != angleOld) {
    dir=(angle>angleOld)^(abs(angle-angleOld)>30); //cw=1; ccw=0; XOR fixes 59/0 crossing
    strip.clear(); //Clear LEDs; set cw=Red, ccw=Green
    strip.setPixelColor(angle, dir*32*brightness, !dir*32*brightness, 0);
    angleOld = angle;
    cycleCounter = 0; //Reset stop counter
  }
    if (cycleCounter >= stopCycles) { //Wash out direction colors to white if stopped
      strip.setPixelColor(angle, 16*brightness, 16*brightness, 12*brightness);
  }
  else {
    cycleCounter++; //Increment stop counter
  }
  strip.show();
} //End main loop (add delay here for faster processors)

/*Functions*/
//EEPROM read
unsigned char EE_read(unsigned char addr) {
  while (EECR & (1 << EEPE)); //Check for write in progress
  EEAR = addr;
  EECR |= (1 << EERE); //Start EEPROM read
  return (EEDR);
}
//EEPROM write
void EE_write(unsigned char addr, unsigned char ucData) {
  while (EECR & (1 << EEPE)); //Wait for completion of previous write
  EECR = (0 << EEPM1) | (0 << EEPM0); //Set programming mode
  EEAR = addr;
  EEDR = ucData;
  EECR |= (1 << EEMPE); //Write 1 to EEMPE
  EECR |= (1 << EEPE); //Start EEPROM write by setting EEPE
}
//ADC subroutine
int getADC() {
  ADCSRA |= _BV(ADSC); //Start conversion
  while ((ADCSRA & _BV(ADSC))); //Wait for conversion
  return ADC;
}
//Read uncalibrated angle (direct ADC access uses less memory than Arduino "analogRead")
//Retuns uncalibrated angle as a scaled 0-60 integer
  unsigned char readAngle () {
  ADMUX = 3; //Read sensor
  AATsin = getADC();
  ADMUX = 2;
  AATcos = getADC();
  return (atan2(float(AATsin-512), float(AATcos-512))/pi+1)*30;
}
```

NVE CORPORATION

Limited Warranty and Liability
Information in this document is believed to be accurate and reliable. However, NVE does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. In no event shall NVE be liable for any indirect, incidental, punitive, special or consequential damages (including, without limitation, lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Right to Make Changes
NVE reserves the right to make changes to information published in this document including, without limitation, specifications and product descriptions at any time and without notice.

Use in Life-Critical or Safety-Critical Applications
Unless NVE and a customer explicitly agree otherwise in writing, NVE products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical devices or equipment. NVE accepts no liability for inclusion or use of NVE products in such applications and such inclusion or use is at the customer's own risk. Should the customer use NVE products for such application whether authorized by NVE or not, the customer shall indemnify and hold NVE harmless against all claims and damages.

Applications
Applications described in this document are illustrative only. NVE makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NVE products, and NVE accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NVE product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customers. Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NVE does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customers. The customer is responsible for all necessary testing for the customer's applications and products using NVE products in order to avoid a default of the applications and the products or of the application or use by customer's third party customers. NVE accepts no liability in this respect.

An ISO 9001 Certified Company

NVE Corporation
11409 Valley View Road
Eden Prairie, MN 55344-3617

©NVE Corporation
All rights are reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner.

Manual No.: SB-00-072