Analog input and hysteresis

INF471C - Modal - Connected Objects and the Internet of Things (IoT) (2021-2022)
Analog input tutorial
Ilja Tihhanovski, Taltech CPS 3

First approach to analog input problem leaded to next code:

```
//brightness thresholds
#define THR_DARK_DEFAULT 150    //dark threshold
#define THR_BRIGHT_DEFAULT 410  //light threshold

//pins for LEDs
#define PIN_RED 11
#define PIN_BLUE 12
#define PIN_GREEN 13

int darkThreshold;
int brightThreshold;

void setup() {
  //Setup pins
  pinMode(PIN_RED, OUTPUT);
  pinMode(PIN_BLUE, OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);

  darkThreshold = THR_DARK_DEFAULT;
  brightThreshold = THR_BRIGHT_DEFAULT;

}

char getButtonPressed()
{
  int bv = analogRead(A1);
  if(bv < 100)
    return 't';
  if(bv > 450 && bv < 550)
    return 'b';
  return ' ';   //assume that nothing is pressed
}

int getBrightness()
{
  return analogRead(A0);
}

void loop(){
  //Initialize output values for all three leds to zero
  byte r = LOW;
  byte g = LOW;
  byte b = LOW;
```

```
  //Get "brigtness value" from analog input
  int brightness = getBrightness();

  //Compare and set one of led values to HIGH
  if(brightness > brightThreshold)
    r = HIGH;
  else
  {
    if(brightness < darkThreshold)
      b = HIGH;
    else
      g = HIGH;
  }

  //Switch the leds
  digitalWrite(PIN_RED, r);
  digitalWrite(PIN_GREEN, g);
  digitalWrite(PIN_BLUE, b);

  char button = getButtonPressed();
    //When pressing the TOP button, the current value of the light sensor will be
recorded as threshold for the switch between RED and GREEN
  if (button == 't')
    darkThreshold = brightness;


    //When pressing the BOTTOM button, the current value of the light sensor will
be recorded as threshold for the switch between BLUE and GREEN.
  if (button == 'b')
    brightThreshold = brightness;
}
```

Listing 1: naïve approach to analog input handling

Problem of such approach appears when analog input value is near switching threshold. Input value will tremble (be above or below threshold many times per second) and it will result in very frequent switches of our output.

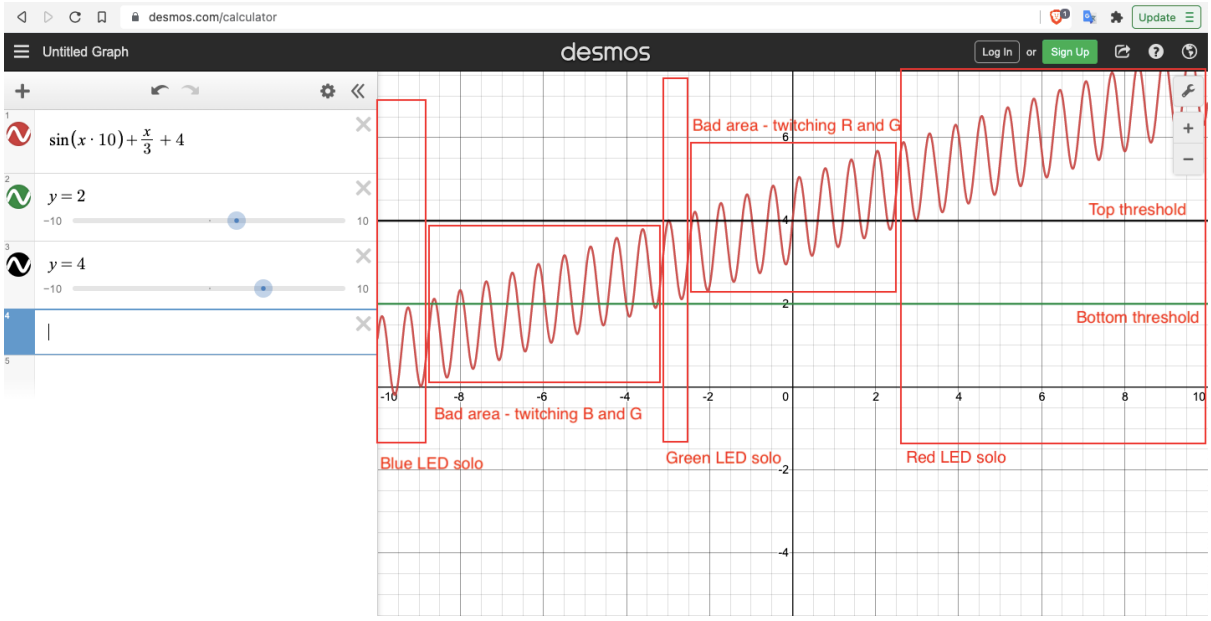

Figure 1: simplified model of analog input trembling. y = 2 and y = 4 are example thresholds, sinusoid is input signal. Here we can observe multiple switching between approximately x ∈(-9, -3) for green threshold and x ∈(-2.5, 2.5) for black threshold. [1]

In our example (LEDs switching) we observed both LEDs being lid (something like PWM) which was undesirable but not damaging. But in real life it could lead to damage, for example if we want to operate relays based on some analog input.

Using average

Then I tried my idea of using some average value of input to get smoother value.
It was not really hysteresis, more like low pass filter.

```
//brightness thresholds
#define THR_DARK_DEFAULT 150    //dark threshold
#define THR_BRIGHT_DEFAULT 410  //light threshold

//pins for LEDs
#define PIN_RED 11
#define PIN_BLUE 12
#define PIN_GREEN 13
#define AVG_COUNT 10


int darkThreshold;
int brightThreshold;
int history[AVG_COUNT];

void setup() {
  //Setup pins
  pinMode(PIN_RED, OUTPUT);
  pinMode(PIN_BLUE, OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);

  darkThreshold = THR_DARK_DEFAULT;
  brightThreshold = THR_BRIGHT_DEFAULT;
```

```
  //initialize history
  for(auto i = 0; i < AVG_COUNT; history[i++] = 0);
}

char getButtonPressed()
{
  int bv = analogRead(A1);
  if(bv < 100)
    return 't';
  if(bv > 450 && bv < 550)
    return 'b';
  return ' ';   //assume that nothing is pressed
}

int getBrightness()
{
  //shift history
  long int sum = 0;
  for(auto i = 1; i < AVG_COUNT; i++)
  {
    history[i - 1] = history[i];
    sum += history[i];
  }
  int s = analogRead(A0);
  history[AVG_COUNT - 1] = s;
  sum += s;

  return sum / AVG_COUNT;
}

void loop(){
  //Initialize output values for all three leds to zero
  byte r = LOW;
  byte g = LOW;
  byte b = LOW;

  //Get "brigtness value" from analog input
  int brightness = getBrightness();

  //Compare and set one of led values to HIGH
  if(brightness > brightThreshold)
    r = HIGH;
  else {
    if(brightness < darkThreshold)
      b = HIGH;
    else
      g = HIGH;
  }

  //Switch the leds
```

```
  digitalWrite(PIN_RED, r);
  digitalWrite(PIN_GREEN, g);
  digitalWrite(PIN_BLUE, b);

  char button = getButtonPressed();
  if (button == 't')
    darkThreshold = brightness;

  if (button == 'b')
    brightThreshold = brightness;
}
```

Listing 2. Using average value to smooth trembling of analog input.

I saved every new input to an array and then used average value of entire array contents: (see modified method `getBrightness()`).

Such approach did not improved result much. Also this approach uses more memory to store data needed to calculate average. The more data we will store (increase `AVG_COUNT`), the smoother our "analog input" will be yet the larger memory we will use for that.
The idea of this approach also do not solve our problem. Yes, we will have smoother graph of input values, but it will still tremble around threshold and unnecessary switching of output will still occur.

Implementing hysteresis

Third approach was to implement something more similar to hysteresis.

```
//brightness thresholds
#define THR_DARK_DEFAULT 150    //dark threshold
#define THR_BRIGHT_DEFAULT 410  //light threshold
#define THR_BORDER 10

//pins for LEDs
#define PIN_RED 11
#define PIN_BLUE 12
#define PIN_GREEN 13

int darkThreshold;
int brightThreshold;

void setup() {
  //Setup pins
  pinMode(PIN_RED, OUTPUT);
  pinMode(PIN_BLUE, OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);

  darkThreshold = THR_DARK_DEFAULT;
  brightThreshold = THR_BRIGHT_DEFAULT;
}

char getButtonPressed()
```

```
{
  int bv = analogRead(A1);
  if(bv < 100)
    return 't';
  if(bv > 450 && bv < 550)
    return 'b';
  return ' ';   //assume that nothing is pressed
}

int getBrightness()
{
  return analogRead(A0);
}

bool wasBright = false;

bool isBright(int brightness)
{
  if (wasBright)
    wasBright = brightness > (brightThreshold - THR_BORDER);
  else
    wasBright = brightness > brightThreshold;
  return wasBright;
}

bool wasDark = false;

bool isDark(int brightness)
{
  if (wasDark)
    wasDark = brightness < (darkThreshold + THR_BORDER);
  else
    wasDark = brightness < darkThreshold;
  return wasDark;
}

void loop(){
  //Initialize output values for all three leds to zero
  byte r = LOW;
  byte g = LOW;
  byte b = LOW;

  //Get "brigtness value" from analog input
  int brightness = getBrightness();

  if (isBright(brightness))
    r = HIGH;
  else
  {
    if (isDark(brightness))
      b = HIGH;
```

```
    else
      g = HIGH;
  }

  //Switch the leds
  digitalWrite(PIN_RED, r);
  digitalWrite(PIN_GREEN, g);
  digitalWrite(PIN_BLUE, b);

  char button = getButtonPressed();
  if (button == 't')
    darkThreshold = brightness;

  if (button == 'b')
    brightThreshold = brightness;

}
```

Listing 3: switching with hysteresis

Here we introduced concept of memory of previous state of system. If system reached some state (bright threshold was reached) then leaving this state will need larger change.
For example, if our brightThreshold = 400 and THR_BORDER = 10, then entering “bright” state will occure at brightness level of 400, but leaving “bright” state will need brightness to drop below 391:

```
bool wasBright = false;

bool isBright(int brightness)
{
  if (wasBright)
    wasBright = brightness > (brightThreshold - THR_BORDER);
  else
    wasBright = brightness > brightThreshold;
  return wasBright;
}
```

Listing 4: memory of state

This approach had much better visual results.
Value of THR_BORDER depends on characteristics of signal (dispersion?) and I think it should maybe depend on values of bright and dark thresholds itself.

References

1. Desmos graphing calculator. https://www.desmos.com/calculator