Documentation of Library KK-Poti

License and Copyright

Published under MIT License

with Copyright (c) 2025-2026 Kay Kasper

See license.txt

Introduction

A library for handling any kinds of analog inputs from potentiometers, attenuators and a lot of other devices. Several options are offered for an easy and performant handling.

The main target was to get an easy understandable code and the chance to concentrate on the real functionality of the project. All functionality for the analog inputs is encapsulated in five separate classes and only changes of the input need attention.

For an easier reading the five classes Poti, StablePoti, MappedPoti, HalfShiftMappedPoti and CenteredPoti are summerized as Poti classes when they are all meant.

Different analog inputs can be handled completely independent in parallel with individual Poti objects.

Advantages

Each of the five existing classes has its specific advantages. Here are only some of them:

- no active waits
- high performance
- low memory usage
- handling current and previous values
- value caching enables stable value analysis
- easy handling in loops with little code
- subclasses for own raw read logic possible
- big number of examples
- intensively tested by manual and automated tests
- value mapping of values to different range (optional)
- reduction of raw value reads (optional)
- stabilization (against often small changes) of values (optional)
- support for potentiometers with a center position (optional)
- two different mapping algorithms (optional)
- compensation of non linear value distribution of potentiometers (optional)

Version history

Version	Released	Comment
1.0.0	20.11.2025	First release with full functionality
1.1.0	31.05.2026	Added new HalfShiftMappedPoti class. Correction of several comment and documentation issues.
1.1.1	12.06.2026	Improved comments and documentation.

Directory

License and Copyright....1
Introduction....1
Advantages....1
Version history....2
Directory....2
Installation....3
Getting started....3
Examples....3
- Simple....4
- Stable....5
- Mapping....5
- HalfShiftMapping....5
- Stretch....6
- Individual....6
- TestPoti....6

Poti....6
- Parameters....7
- Constants....8

StablePoti....8
- Parameters....8

MappedPoti....9
- Parameters....10
- Constants....10

HalfShiftMappedPoti....10
- Parameters....11
- Constants....12

CenteredPoti....12
- Parameters....13

Installation

The library can be installed via Arduino Library Manager or by downloading the archive from directory „library“ and unpacking the archive in IDEs libraries directory.

Getting started

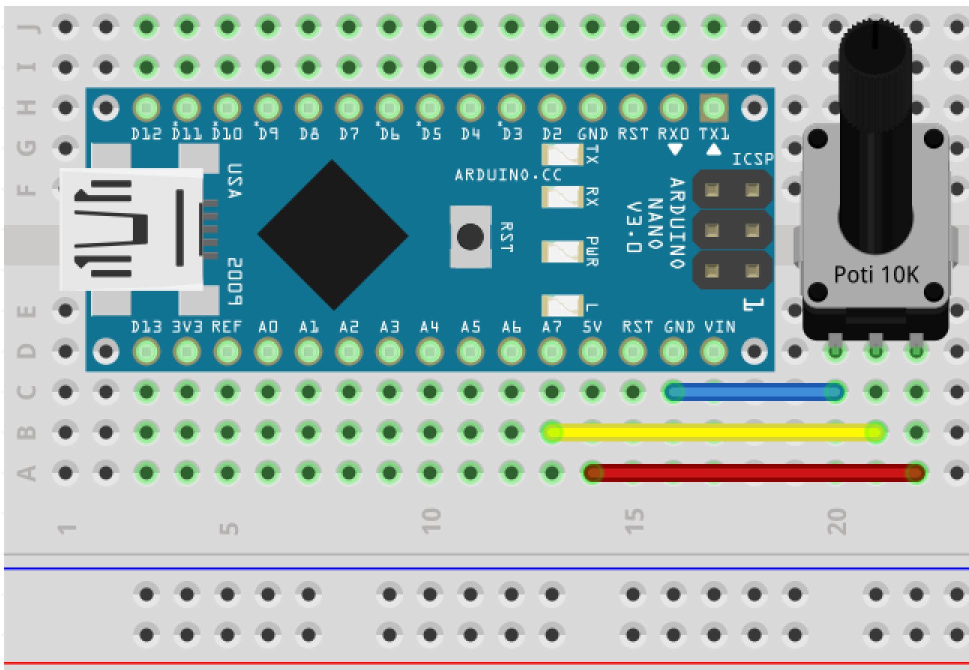
Only a few lines of code are necessary to understand the logic behind:

```
#include <Poti.h>

#define INPUT_PIN A7            // must be analog pin A0 to A7
#define READ_CYCLE_MILLIS 200  // millis between two reads of analog raw value

Poti pot = Poti(INPUT_PIN, READ_CYCLE_MILLIS);

void setup() {
  // for showing relevant information
  Serial.begin(9600);
}


void loop() {
  // react on changing values by writing the relevant information
  if(pot.hasChanged()){
    Serial.print("curVal=");
    Serial.print(pot.getValue());

    Serial.print(", prevVal=");
    if(pot.getPrevValue() == POTI_VALUE_UNDEFINED){
      Serial.print("POTI_VALUE_UNDEFINED");
    }
    else{
      Serial.print(pot.getPrevValue());
    }

    Serial.print("\n");
  }
}
```

See the following examples and more details below.

Examples

Based on a typical easy electrical design (like shown in the following picture with Arduino Nano and a potentiometer connected to analog pin A7)



several examples are included to show the usage and possibilities.

Simple

Simple example to show the current value of the connected potentiometer. The example will show the instability of measured values over the time, when the potentiometer is not touched. In the example are the available stabilisation methods not configured, means in use.

Prerequisite is an potentiometer connected with variable voltage pin to analog input pin. Output will be written to Serial.

```
/*
  Copyright (c) 2025 Kay Kasper
  under the MIT License (MIT)
*/

#include <Poti.h>

#define INPUT_PIN A7            // must be analog pin A0 to A7
#define READ_CYCLE_MILLIS 200   // millis between two reads of analog raw value

Poti pot = Poti(INPUT_PIN, READ_CYCLE_MILLIS);

// the setup function is called once for initialization
void setup() {
  Serial.begin(9600);
}


// for showing relevant information
void printValues(){
  Serial.print("curVal=");
  if(pot.getValue() == POTI_VALUE_UNDEFINED){
    Serial.print("POTI_VALUE_UNDEFINED");
  }
  else{
```

```
    Serial.print(pot.getValue());
  }

  Serial.print(", prevVal=");
  if(pot.getPrevValue() == POTI_VALUE_UNDEFINED){
    Serial.print("POTI_VALUE_UNDEFINED");
  }
  else{
    Serial.print(pot.getPrevValue());
  }

  Serial.print("\n");
}


// the loop function runs over and over again forever
void loop() {
  // react on changing values by writing the relevant information
  if(pot.hasChanged()){
    printValues();
  }
}
```

Stable

Example to show the current value of the connected potentiometer with and without stabilization methods. Methods can be compared in their consequences. In the example both Poti objects use the same INPUT_PIN. This should normally not be done, but is without problems possible.

Prerequisite is an potentiometer connected with variable voltage pin to analog input pin. Output will be written to Serial.

Mapping

Example to generate mapped values from raw input based on an analog input pin.

The mapping algorithem will, in case of not used stretching, try to map an equal number of raw input values to each mapping value.

The variable value of the analog pin will change the speed of blinking of the build in LED.

The example reduces the effect of instability of measured values over the time, when the potentiometer is not touched by using a mapping. In the example are the other available stabilisation methods not configured, means not in use.

Prerequisite is an potentiometer connected with variable voltage pin to analog input pin. Output will be written to Serial. LED is assumed to be always available at pin LED_BUILTIN.

HalfShiftMapping

Example to generate mapped values from raw input based on an analog input pin.

The mapping algorithm will be most suitable for mapping predefined and expected analog values with equal difference between mapping values. E.g. for 0V, 1V, ... , 4 V and 5V to 6 mapping values.

The variable value of the analog pin will change the speed of blinking of the build in LED.

The example reduces the effect of instability of measured values over the time, when the potentiometer is not touched by using a mapping. In the example are the other available stabilisation methods not configured, means not in use.

Prerequisite is an potentiometer connected with variable voltage pin to analog input pin. Output will be written to Serial. LED is assumed to be always available at pin LED_BUILTIN.

Stretch

Example to show the mapped value of the connected potentiometer with and without the stretching method. The stretching can be compared in its consequences. In the example both Poti objects use the same INPUT_PIN. This should normally not be done, but is without problems possible.

Prerequisite is an potentiometer connected with variable voltage pin to analog input pin. Output will be written to Serial.

Individual

Example to show the possibility to create subclasses with own logic for mapping or stabilization of the values measured from the connected potentiometer. The function getRawValue() is therefore overwittten.

Prerequisite is an potentiometer connected with variable voltage pin to analog input pin. Output will be written to Serial.

TestPoti

Example that tests the functionality of Poti, StablePoti, MappedPoti, HalfShiftmappedPoti and CenteredPoti classes. Several checks and performance measurements are done continously in the loop.

Output will be written to Serial.

Poti

Needs include of header file „Poti.h“.

The Poti class is used for easy handling of potentiometers, attenuators and other kinds of analog input signals.

Analog input signals are directly returned without stabilization by reading a raw value from an anlog input pin.

Normally it is interresting to check the measured value only, when it changed. Then typically actions have to be started. Therefore the logic in the Poti class is to ask in the processing loop only with function hasChanged(), if something has changed between now and the previous call of hasChanged().

The evaluation of value changes is done only in the function hasChanged() and it is the only function, that calls getRawState(), to read the raw input value. Therefore the function

has to be called in the programm loop at least once, to identify changes as fast as necessary.

Only when hasChanged() deliveres true, a different value than before must be analyzed. Between calls of hasChanged() the values are stable and will not change. The functions getValue() and getPrevValue() will always return the value, that was identified and internally set by the last call of hasChanged().

When function hasChanged() is called the first time, it always returns true. This is because the internally used previous value is defined as POTI_VALUE_UNDEFINED. This makes it possible to react immediately when the programm starts in the loop and no other initialization logic is necessary.

Raw input value measurements (by function getRawValue()) is based on a slow analog to digital conversion (about 100 nanos in Arduinos). Therefore the

number of calls of getRawState() calls can optionally be reduced to a minimum with an included logic that implements a minimum time delay between the calls of getRawValue().

The function getRawValue() can be overwritten by a subclass, so that a possibility is given to use whatever input and translate it into different "analog" values.

Normally it is not necessary to configure the used analog input pin for analog read. But if it is necessary, this configuration must be done in the setup() before the first time hasChanged() is called.

If the analog input is comming from an potentiometer, this means with an average potentiometer, that the measured analog values may change from one to the next measurement even if the potentiometer was not touched in the meantime. The variation of the raw values depends on thermical topics, mechanical topic (e.g. vibrations), varying power supply voltage (even a blinking LED may be a reason) and so on.

Linear and logarithmical potentiometers are supported by Poti class. But "linear" means in most cases not, that the measured values are equally distibuted. Normally are the values slowly changing at the beginning and the end of the movement. In the middle position normally values are changing fast with little movement. This has to be taken into account.

Parameters

Name	Description
inputPin	Analog pin for reading the analog raw value. Possible pin configuration must be done before hasChanged() calls. Values for Arduino e.g. A0 to A7
readCycleMillis	Minimum time in milliseconds that must have been waited between succeeding calls of getRawValue(). Values from 0 to 255. Value 0 means no waits and getRawvalue() is called by each hasChanged() call

Constants

Name	Description
POTI_VALUE_UNDEFINED	Special value that is delivered, if no value has been

	measured till that moment

StablePoti

Needs include of header file „StablePoti.h“.

Based on the Poti class and all its advantages the StablePoti class adds some functionallity to stabalize the measured raw values for a better and easier usage of the values. StablePoti is a subclass of the Poti class.

If the analog input is comming from an potentiometer, this means with an average potentiometer, that the measured analog values may change from one to the next measurement even if the potentiometer was not touched in the meantime. The variation of the raw values depends on thermical topics, mechanical topic (e.g. vibrations), varying power supply voltage (even a blinking LED may be a reason) and so on. Therefore two methods are implemented internally to stabilize the output value.

The first method for stabilizing the internal value is the building of an average of several raw values before the processing by the next method is continued. This method is as option controlled by defining additional measurements with a time difference of 1 millisecond by parameter addNumRawAvg. Additional measurements based on addNumRawAvg are prioritized to the delay in reading raw values based on leadCycleMillis.

The second method for stabilizing the (by first method given) internal new value is done by taking the previously given and stabilized value into account. New and previous values are added with given weights into one new external usable value. This method is as option controlled by parameter weightPrev, that defines the weight of the previous value based on a fixed given weight of 4 for the new value. Side effect of this average calculation is a reduced speed in the change of output values. The real analog value will be reached with delay. The higher the weight of the previous value is, the stronger is the delay.

Parameters

Name	Description
inputPin	Analog pin for reading the analog raw value. Possible pin configuration must be done before hasChanged() calls. Values for Arduino e.g. A0 to A7
readCycleMillis	Minimum time in milliseconds that must have been waited between succeeding calls of getRawValue(). Values from 0 to 255. Value 0 means no waits and getRawvalue() is called by each hasChanged() call
weightPrev	Weight of the previous value, when the new output value is calculated as combined value. Values 0 to 12. Value 0 means no weighting logic
addNumRawAvg	Additional nummer of raw value measurements for building an average with first measurement. Values 0 to 7. Value 0 means no average calculation

MappedPoti

Needs include of header file „MappedPoti.h“.

Based on the Poti and StablePoti classes and all its advantages the MappedPoti class adds some functionallity for mapping the analog values to a configurable new and different range of values. MappedPoti is a subclass of StablePoti.

Mapping reduces the precision of the analog values to a lower number of values and can also be seen as another stabilization method for easier handling of analog values based on potentiometers.

When mapping is used, linear potentiometer shall be used. Logarithmical potentiometers are not suitable.

Every returned value of getRawValue() is in relation to a position of a specific potentiometer. In this implementation it is always assumed, that the lowest raw value is 0 and the position for this value is when the potentiometer is turned completely to the left. When turned completely to the right side the maximum value (e.g. 1023 or 4095, depending on specific microcontroller) is read. The lowest mapping values of 0 always includes the analog value of 0 and the highest mapping value always includes the highest analog value.

The range of mapping values is defined as 0 to numMapping - 1. "numMapping" is an instantiation parameter and allows values from 2 to 100. If the parameter value is uneven, then the potentiometer is assumed to have the middle mapping value around the middle analog value (e.g. 511 for maximum analog value 1023). The mapping is implemented in standard case (instantiation parameter stretch=0) by assuming a linear distribution of analog values and by calculation how many analog values need to be summerized in one mapping value. With ideal linear potentiometers each mapping value would mean the same distance/way that the potentiometer must be turned. During the calculation rounding errors may occour.

Example for standard mapping by setting numMapping = 4 and stretch=0:
0-255 -> mapping 0; 256-511 -> mapping 1;
512-767 -> mapping 2; 768-1023 -> mapping 3

As already described in Poti class, linear potentiometers are in reality not linear. "linear" means in most cases not, that the measured values are equally distibuted. Normally are the values slowly changing at the beginning and the end of the movement. In the middle position normally values are changing fast with little movement.

The MappedPoti class implements a method to compensate the not linear distribution of analog values. With instantiation parameter "stretch" the distribution of analog values to mapping values can be influenced. The higher the configured stretching is, the more values in the middle and the less values at the edges of the potentiometer are summerized in one mapping value. As a consequence the movement in the middle is bigger and to the edges is smaller for one mapping value than with linear mapping.

In general it is still relevant ot have stable analog input values for MappedPoti because often changing analog values at the border of two mapping values may cause permanent changing of mapping values. Therefore MappedPoti is based on StablePoti.

The external view of the MappedPoti is based on mapping values. Additional functions have been created for handling the mapping. The hasChanged() function reacts only on changed mapping values. When changes occured, the analog value of getValue() is the one, that was relevant for the change. Analog values can change in the background without reflection in the value functions. The mapping values are always based on the stored analog values.

Very important for the mapping is the knowledge of the highest possible analog value (defined as maxAnalogVal). For getting and setting this information are functions available. Default is 1023, typically for many Arduino microcontroller. If the maximum number is different to the default, then the function setMaxAnalogValue() must be called before first use of function hasChanged() to set the real maximum number (e.g. 4095).

Parameters

Name	Description
inputPin	Analog pin for reading the analog raw value. Possible pin configuration must be done before hasChanged() calls. Values for Arduino e.g. A0 to A7
readCycleMillis	Minimum time in milliseconds that must have been waited between succeeding calls of getRawValue(). Values from 0 to 255. Value 0 means no waits and getRawvalue() is called by each hasChanged() call
weightPrev	Weight of the previous value, when the new output value is calculated as combined value. Values 0 to 12. Value 0 means no weighting logic
addNumRawAvg	Additional nummer of raw value measurements for building an average with first measurement. Values 0 to 7. Value 0 means no average calculation
numMapping	Number of mapping values. Range is from 2 to 100. Default is 2, so mapping is always used.
stretch	Factor for stretching analog values during the mapping calculation. Values from 0 (no use, linear) to 20. Value 20 ist highest stretching.

Constants

Name	Description
POTI_MAPPING_UNDEFINED	Special value that is delivered, if no mapping can be delivered, because no value has been measured till that moment

HalfShiftMappedPoti

Needs include of header file „HalfShiftMappedPoti.h“.

Based on MappedPoti class, but with a different mapping logic. HalfShiftMappedPoti is a subclass of MappedPoti. See MappedPoti first.

The difference in the mapping logic is:

MappedPoti class implements a logic, where (when no stretching is used) the algorithm tries to map the same number of analog raw input values to one mapping value.

HalfShiftMappedPoti class implements a logic, where (when no stretching is used) the algorithm tries to map different number of analog raw input values to one mapping value. The first and last mapping value gets only half of the number of analog raw input values as the other mapping values (between first and last mapping value).

You can see the difference of half shifted mapping by setting numMapping = 4 and stretch=0 in this example (and comparing with the same example of MappedPoti class):
0-170 -> mapping 0; 171-511 -> mapping 1;
512-852 -> mapping 2; 853-1023 -> mapping 3

The algorithm of the HalfShiftMappedPoti class is ideal for expected raw analog input values, that have a fixed difference (like steps). E.g. 1,66V steps and a range from 0V to 5V with 4 mapping values will handle the expected values 0V, 1.66V, 3.33V and 5V with an input variablility of up to +/-0,83V per value.

The available stretching and stabilization of analog raw input values is good for adjusting different potentiometer to the mapping logic in the MappedPoti class, because normal potentiometer are not linear and quite unstable. But the typical use case for the HalfShiftMappedPoti class is based on steps than on variable analog values, so the stretching and other stabilization should be unnecessary.

All other explanations for MappedPoti class also apply to HalfShiftMappedPoti class.

Parameters

Name	**Description**
inputPin	Analog pin for reading the analog raw value. Possible pin configuration must be done before hasChanged() calls. Values for Arduino e.g. A0 to A7
readCycleMillis	Minimum time in milliseconds that must have been waited between succeeding calls of getRawValue(). Values from 0 to 255. Value 0 means no waits and getRawvalue() is called by each hasChanged() call
weightPrev	Weight of the previous value, when the new output value is calculated as combined value. Values 0 to 12. Value 0 means no weighting logic
addNumRawAvg	Additional nummer of raw value measurements for building an average with first measurement. Values 0 to 7. Value 0 means no average calculation
numMapping	Number of mapping values. Range is from 2 to 100. Default is 2, so mapping is always used.
stretch	Factor for stretching analog values during the mapping calculation. Values from 0 (no use, linear) to 20. Value 20 ist highest stretching.

Constants

Name	**Description**
POTI_MAPPING_UNDEFINED	Special value that is delivered, if no mapping can be

	delivered, because no value has been measured till that moment

CenteredPoti

Needs include of header file „CenteredPoti.h“.

Based on the Poti, StablePoti and MappedPoti classes and all its advantages the CenteredPoti class adds some functionality specially for potentiometers that are used in cases where a center position is necessary (e.g. for treble, bass, balance in amplifiers). The main functionality in this class is to transform all given values and standard mappings (starting with 0 to a max value) to a different range from -x ... 0 ... +x where x = (numMapping-1)/2. CenteredPoti class is a subclass of MappedPoti class.

Mapping is used and linear potentiometer shall be used. Logarithmical potentiometers are not suitable.

For understanding the mapping, the description of the MappedPoti class is relevant. Centered potentiometer must have always an uneven number of mapping values, so that a symmetric distribution of mapping value on the left and right side is possible. The central mapping value will always be defined as value 0 compared to the linear mapping values from 0 to maxMapping-1 in MappedPoti class. The centered mapping values on the left side have negative signs and the more the potentiometer is turned left, the more negative becomes the value. The centered mapping values on the right side have positive signs and the more the potentiometer is turned right, the more positive becomes the value.

Very important for the mapping is the knowledge of the highest possible analog value (defined as maxAnalogVal). For getting and setting this information are functions available. Default is 1023, typically for many Arduino microcontroller. If the maximum number is different to the default, then the function setMaxAnalogValue() must be called before first use of function hasChanged() to set the real maximum number (e.g. 4095).

Parameters

Name	Description
inputPin	Analog pin for reading the analog raw value. Possible pin configuration must be done before hasChanged() calls. Values for Arduino e.g. A0 to A7
readCycleMillis	Minimum time in milliseconds that must have been waited between succeeding calls of getRawValue(). Values from 0 to 255. Value 0 means no waits and getRawvalue() is called by each hasChanged() call
weightPrev	Weight of the previous value, when the new output value is calculated as combined value. Values 0 to 12. Value 0 means no weighting logic
addNumRawAvg	Additional nummer of raw value measurements for building an average with first measurement. Values 0 to 7. Value 0 means no average calculation
numMapping	Number of mapping values. Range is from 2 to 100. Default is 2, so mapping is always used.
stretch	Factor for stretching analog values during the mapping calculation. Values from 0 (no use, linear) to 20. Value 20 ist highest stretching.
centerTol	Tolerance on left and right side of centerVal. Center is defined as 2*centerTol + 1 values. Values from 10 to 255. Default is 10.
centerVal	Analog value of the physical center position of the turning knob. If 0 is given, a standard value is calculated internally as maxAnalogVal/2.