// AccelStepper.h

//

/// \mainpage AccelStepper library for Arduino

///

/// This is the Arduino AccelStepper 1.2 library.

/// It provides an object-oriented interface for 2 or 4 pin stepper motors.

///

/// The standard Arduino IDE includes the Stepper library

/// (http://arduino.cc/en/Reference/Stepper) for stepper motors. It is

/// perfectly adequate for simple, single motor applications.

///

/// AccelStepper significantly improves on the standard Arduino Stepper library in several ways:

/// \li Supports acceleration and deceleration

/// \li Supports multiple simultaneous steppers, with independent concurrent stepping on each stepper

/// \li API functions never delay() or block

/// \li Supports 2 and 4 wire steppers

/// \li Supports stepper drivers such as the Sparkfun EasyDriver (based on 3967 driver chip)

/// \li Very slow speeds are supported

/// \li Extensive API

/// \li Subclass support

///

/// The latest version of this documentation can be downloaded from

/// http://www.open.com.au/mikem/arduino/AccelStepper

///

/// Example Arduino programs are included to show the main modes of use.

///

/// The version of the package that this documentation refers to can be downloaded

/// from http://www.open.com.au/mikem/arduino/AccelStepper/AccelStepper-1.3.zip

/// You can find the latest version at http://www.open.com.au/mikem/arduino/AccelStepper

///

/// Tested on Arduino Diecimila and Mega with arduino-0018 on OpenSuSE 11.1 and avr-libc-1.6.1-1.15,

/// cross-avr-binutils-2.19-9.1, cross-avr-gcc-4.1.3\_20080612-26.5.

///

/// \par Installation

/// Install in the usual way: unzip the distribution zip file to the libraries

/// sub-folder of your sketchbook.

///

/// This software is Copyright (C) 2010 Mike McCauley. Use is subject to license

/// conditions. The main licensing options available are GPL V2 or Commercial:

///

/// \par Open Source Licensing GPL V2

/// This is the appropriate option if you want to share the source code of your

/// application with everyone you distribute it to, and you also want to give them

/// the right to share who uses it. If you wish to use this software under Open

/// Source Licensing, you must contribute all your source code to the open source

/// community in accordance with the GPL Version 2 when your application is

/// distributed. See http://www.gnu.org/copyleft/gpl.html

///

/// \par Commercial Licensing

/// This is the appropriate option if you are creating proprietary applications

/// and you are not prepared to distribute and share the source code of your

/// application. Contact info@open.com.au for details.

///

/// \par Revision History

/// \version 1.0 Initial release

///

/// \version 1.1 Added speed() function to get the current speed.

/// \version 1.2 Added runSpeedToPosition() submitted by Gunnar Arndt.

/// \version 1.3 Added support for stepper drivers (ie with Step and Direction inputs) with \_pins == 1

///

///

/// \author Mike McCauley (mikem@open.com.au)

// Copyright (C) 2009 Mike McCauley

// $Id: AccelStepper.h,v 1.2 2010/10/24 07:46:18 mikem Exp mikem $

#ifndef AccelStepper\_h

#define AccelStepper\_h

#if ARDUINO >= 100

#include "Arduino.h"

#else

#include "WProgram.h"

#include "stdlib.h"

#include "wiring.h"

#endif

// These defs cause trouble on some versions of Arduino

#undef round

/////////////////////////////////////////////////////////////////////

/// \class AccelStepper AccelStepper.h <AccelStepper.h>

/// \brief Support for stepper motors with acceleration etc.

///

/// This defines a single 2 or 4 pin stepper motor, or stepper moter with fdriver chip, with optional

/// acceleration, deceleration, absolute positioning commands etc. Multiple

/// simultaneous steppers are supported, all moving

/// at different speeds and accelerations.

///

/// \par Operation

/// This module operates by computing a step time in milliseconds. The step

/// time is recomputed after each step and after speed and acceleration

/// parameters are changed by the caller. The time of each step is recorded in

/// milliseconds. The run() function steps the motor if a new step is due.

/// The run() function must be called frequently until the motor is in the

/// desired position, after which time run() will do nothing.

///

/// \par Positioning

/// Positions are specified by a signed long integer. At

/// construction time, the current position of the motor is consider to be 0. Positive

/// positions are clockwise from the initial position; negative positions are

/// anticlockwise. The curent position can be altered for instance after

/// initialization positioning.

///

/// \par Caveats

/// This is an open loop controller: If the motor stalls or is oversped,

/// AccelStepper will not have a correct

/// idea of where the motor really is (since there is no feedback of the motor's

/// real position. We only know where we \_think\_ it is, relative to the

/// initial starting point).

///

/// The fastest motor speed that can be reliably supported is 1000 steps per

/// second (1 step every millisecond). However any speed less than that down

/// to very slow speeds (much less than one per second) are supported,

/// provided the run() function is called frequently enough to step the

/// motor whenever required.

class AccelStepper

{

public:

/// Constructor. You can have multiple simultaneous steppers, all moving

/// at different speeds and accelerations, provided you call their run()

/// functions at frequent enough intervals. Current Position is set to 0, target

/// position is set to 0. MaxSpeed and Acceleration default to 1.0.

/// The motor pins will be initialised to OUTPUT mode during the

/// constructor by a call to enableOutputs().

/// \param[in] pins Number of pins to interface to. 1, 2 or 4 are

/// supported. 1 means a stepper driver (with Step and Direction pins)

/// 2 means a 2 wire stepper. 4 means a 4 wire stepper.

/// Defaults to 4 pins.

/// \param[in] pin1 Arduino digital pin number for motor pin 1. Defaults

/// to pin 2. For a driver (pins==1), this is the Step input to the driver. Low to high transition means to step)

/// \param[in] pin2 Arduino digital pin number for motor pin 2. Defaults

/// to pin 3. For a driver (pins==1), this is the Direction input the driver. High means forward.

/// \param[in] pin3 Arduino digital pin number for motor pin 3. Defaults

/// to pin 4.

/// \param[in] pin4 Arduino digital pin number for motor pin 4. Defaults

/// to pin 5.

AccelStepper(uint8\_t pins = 4, uint8\_t pin1 = 2, uint8\_t pin2 = 3, uint8\_t pin3 = 4, uint8\_t pin4 = 5);

/// Constructor. You can have multiple simultaneous steppers, all moving

/// at different speeds and accelerations, provided you call their run()

/// functions at frequent enough intervals. Current Position is set to 0, target

/// position is set to 0. MaxSpeed and Acceleration default to 1.0.

/// Any motor initialization should happen before hand, no pins are used or initialized.

/// \param[in] forward void-returning procedure that will make a forward step

/// \param[in] backward void-returning procedure that will make a backward step

AccelStepper(void (\*forward)(), void (\*backward)());

/// Set the target position. The run() function will try to move the motor

/// from the current position to the target position set by the most

/// recent call to this function.

/// \param[in] absolute The desired absolute position. Negative is

/// anticlockwise from the 0 position.

void moveTo(long absolute);

/// Set the target position relative to the current position

/// \param[in] relative The desired position relative to the current position. Negative is

/// anticlockwise from the current position.

void move(long relative);

/// Poll the motor and step it if a step is due, implementing

/// accelerations and decelerations to achive the ratget position. You must call this as

/// fequently as possible, but at least once per minimum step interval,

/// preferably in your main loop.

/// \return true if the motor is at the target position.

boolean run();

/// Poll the motor and step it if a step is due, implmenting a constant

/// speed as set by the most recent call to setSpeed().

/// \return true if the motor was stepped.

boolean runSpeed();

/// Sets the maximum permitted speed. the run() function will accelerate

/// up to the speed set by this function.

/// \param[in] speed The desired maximum speed in steps per second. Must

/// be > 0. Speeds of more than 1000 steps per second are unreliable.

void setMaxSpeed(float speed);

/// Sets the acceleration and deceleration parameter.

/// \param[in] acceleration The desired acceleration in steps per second

/// per second. Must be > 0.

void setAcceleration(float acceleration);

/// Sets the desired constant speed for use with runSpeed().

/// \param[in] speed The desired constant speed in steps per

/// second. Positive is clockwise. Speeds of more than 1000 steps per

/// second are unreliable. Very slow speeds may be set (eg 0.00027777 for

/// once per hour, approximately. Speed accuracy depends on the Arduino

/// crystal. Jitter depends on how frequently you call the runSpeed() function.

void setSpeed(float speed);

/// The most recently set speed

/// \return the most recent speed in steps per second

float speed();

/// The distance from the current position to the target position.

/// \return the distance from the current position to the target position

/// in steps. Positive is clockwise from the current position.

long distanceToGo();

/// The most recently set target position.

/// \return the target position

/// in steps. Positive is clockwise from the 0 position.

long targetPosition();

/// The currently motor position.

/// \return the current motor position

/// in steps. Positive is clockwise from the 0 position.

long currentPosition();

/// Resets the current position of the motor, so that wherever the mottor

/// happens to be right now is considered to be the new position. Useful

/// for setting a zero position on a stepper after an initial hardware

/// positioning move.

/// \param[in] position The position in steps of wherever the motor

/// happens to be right now.

void setCurrentPosition(long position);

/// Moves the motor to the target position and blocks until it is at

/// position. Dont use this in event loops, since it blocks.

void runToPosition();

/// Runs at the currently selected speed until the target position is reached

/// Does not implement accelerations.

boolean runSpeedToPosition();

/// Moves the motor to the new target position and blocks until it is at

/// position. Dont use this in event loops, since it blocks.

/// \param[in] position The new target position.

void runToNewPosition(long position);

/// Disable motor pin outputs by setting them all LOW

/// Depending on the design of your electronics this may turn off

/// the power to the motor coils, saving power.

/// This is useful to support Arduino low power modes: disable the outputs

/// during sleep and then reenable with enableOutputs() before stepping

/// again.

void disableOutputs();

/// Enable motor pin outputs by setting the motor pins to OUTPUT

/// mode. Called automatically by the constructor.

void enableOutputs();

protected:

/// Forces the library to compute a new instantaneous speed and set that as

/// the current speed. Calls

/// desiredSpeed(), which can be overridden by subclasses. It is called by

/// the library:

/// \li after each step

/// \li after change to maxSpeed through setMaxSpeed()

/// \li after change to acceleration through setAcceleration()

/// \li after change to target position (relative or absolute) through

/// move() or moveTo()

void computeNewSpeed();

/// Called to execute a step. Only called when a new step is

/// required. Subclasses may override to implement new stepping

/// interfaces. The default calls step1(), step2() or step4() depending on the

/// number of pins defined for the stepper.

/// \param[in] step The current step phase number (0 to 3)

virtual void step(uint8\_t step);

/// Called to execute a step using stepper functions (pins = 0) Only called when a new step is

/// required. Calls \_forward() or \_backward() to perform the step

virtual void step0(void);

/// Called to execute a step on a stepper drover (ie where pins == 1). Only called when a new step is

/// required. Subclasses may override to implement new stepping

/// interfaces. The default sets or clears the outputs of Step pin1 to step,

/// and sets the output of \_pin2 to the desired direction. The Step pin (\_pin1) is pulsed for 1 microsecond

/// which is the minimum STEP pulse width for the 3967 driver.

/// \param[in] step The current step phase number (0 to 3)

virtual void step1(uint8\_t step);

/// Called to execute a step on a 2 pin motor. Only called when a new step is

/// required. Subclasses may override to implement new stepping

/// interfaces. The default sets or clears the outputs of pin1 and pin2

/// \param[in] step The current step phase number (0 to 3)

virtual void step2(uint8\_t step);

/// Called to execute a step on a 4 pin motor. Only called when a new step is

/// required. Subclasses may override to implement new stepping

/// interfaces. The default sets or clears the outputs of pin1, pin2,

/// pin3, pin4.

/// \param[in] step The current step phase number (0 to 3)

virtual void step4(uint8\_t step);

/// Compute and return the desired speed. The default algorithm uses

/// maxSpeed, acceleration and the current speed to set a new speed to

/// move the motor from teh current position to the target

/// position. Subclasses may override this to provide an alternate

/// algorithm (but do not block). Called by computeNewSpeed whenever a new speed neds to be

/// computed.

virtual float desiredSpeed();

private:

/// Number of pins on the stepper motor. Permits 2 or 4. 2 pins is a

/// bipolar, and 4 pins is a unipolar.

uint8\_t \_pins; // 2 or 4

/// Arduino pin number for the 2 or 4 pins required to interface to the

/// stepper motor.

uint8\_t \_pin1, \_pin2, \_pin3, \_pin4;

/// The current absolution position in steps.

long \_currentPos; // Steps

/// The target position in steps. The AccelStepper library will move the

/// motor from teh \_currentPos to the \_targetPos, taking into account the

/// max speed, acceleration and deceleration

long \_targetPos; // Steps

/// The current motos speed in steps per second

/// Positive is clockwise

float \_speed; // Steps per second

/// The maximum permitted speed in steps per second. Must be > 0.

float \_maxSpeed;

/// The acceleration to use to accelerate or decelerate the motor in steps

/// per second per second. Must be > 0

float \_acceleration;

/// The current interval between steps in milliseconds.

unsigned long \_stepInterval;

/// The last step time in milliseconds

unsigned long \_lastStepTime;

// The pointer to a forward-step procedure

void (\*\_forward)();

// The pointer to a backward-step procedure

void (\*\_backward)();

};

#endif