

CapTlvate Capacitive Touch

Enhance user experience by adding common gesturing to a capacitive touch slider

A series of tips and tricks for CapTlvate Technology

By Dennis Lehman, Feb 2018

BACKGROUND

Capacitive touch HMI (human machine interface) is becoming more and more common all around us. We all interact with hundreds of different products and devices with this technology, many by only a simple touch or swipe gesture. As an example, a basic HMI with buttons that can recognize single or simultaneous button inputs, such as a keypad, and a slider that is used for a volume control input by detecting the position of the user's finger. For many applications this is all that is needed.

This same HMI can also provide some basic gesturing inputs such as tap, double-tap, tap-hold, swipe left and swipe right using an existing slider sensor without any hardware or design modifications. This is done purely in software and in fact, since a wheel is a slider sensor this applies to wheel sensors as well.

This document focuses on a software implementation that demonstrates how gestures can be easily applied to any CapTlvate capacitive touch slider. Working example code is provided separate from this paper and the reader is encouraged to build it into their application and experiment.

This paper is accompanied by the source code files for the gesture algorithm and a Code Composer Studio demo project that demonstrates gestures using the CAPTIVATE-BSWP.

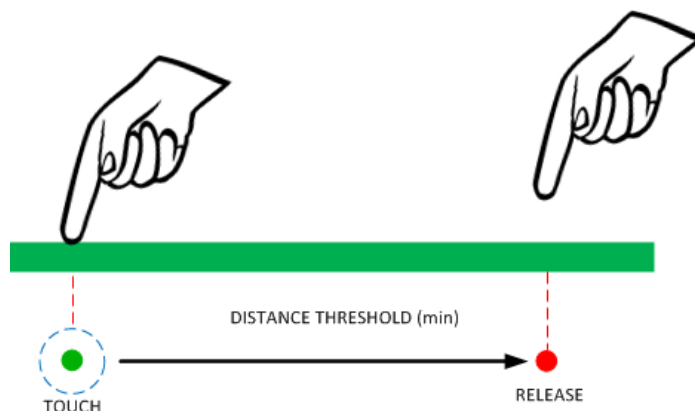
Questions? - please post on e2e.ti.com/

Contents

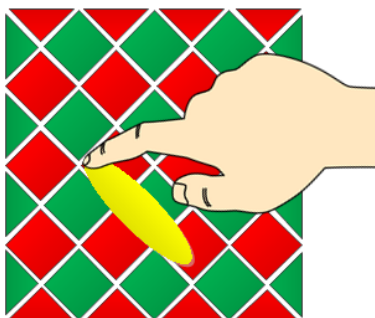
BACKGROUND	1
HOW GESTURING WORKS	3
GESTURE ALGORITHM.....	4
GESTURES.....	5
TAP GESTURE	5
DOUBLE TAP GESTURE	6
TAP-HOLD GESTURE	6
SWIPE GESTURE	7
HOW TO ADD GESTURE CAPABILITY	8
PROJECT SETTINGS AND METRICS	8

HOW GESTURING WORKS

Capacitive touch sensors with one or more degrees of freedom, such as sliders, wheels and trackpads have a behavior that allow them to determine the position of a user's finger touch anywhere on the surface of the sensor. For sliders and wheels, position can only be on one axis.



A trackpad can have a position on two axes, X and Y.



For both types of sensors, gesturing can be thought of as both “touch” and “release” events that occur in a specific order and allocated time. Some gestures may share a common order or sequence, but their length in time determines one gesture from the other. For example, tap, double-tap and tap-hold have common touch and release transitions, but with different release time deadlines.

Swipe gestures have these same timing deadlines but also include a distance requirement. The distance is determined from the position of a touch event to the position of a release event. For sliders and wheels this would be along one axis and for trackpads it would be a vector.

GESTURE ALGORITHM

Fundamentally, the gesture algorithm determines a gesture based on the timing between a touch and release event. To measure time, the example gesture algorithm uses the known scheduled sensor sampling period used by the CapTIvate timer to scan and measure the slider sensor. This period is set by the user with common scan periods for typical HMI applications ranging from 10 to 50ms.

The key here is this scan period creates the base tick rate for the gesture timing.

This means if we know we are going to be sampling our sensors at a known rate, say once every 10ms, we can use this 10ms x N samples to establish our timing thresholds for the gesture parameters. For example, to detect a “tap” we measure the time when a touch is detected and the time when a release is detected, then determine the samples taken during this period. If it meets the tap min and max thresholds, the tap is valid.

In a typical HMI application the sample rate doesn’t change while the MCU is operating in the active mode. However, in low power modes the sample rate may be set lower to reduce power consumption, but this won’t impact the gesture timing because the gesture algorithm only executes while the MCU is in active mode.

So it should be obvious that if the sample rate does change, the algorithm parameters will most likely for not work for the new rate so a set of new parameters will need to be determined based on the new sample rate.

The example gesture algorithm is broken into two parts: Part A performs the slider sensor’s touch/no-touch conditions and records the touch position after each measurement. Part B is a state machine which determines the gesture based on the slider’s touch status and state of the state machine.

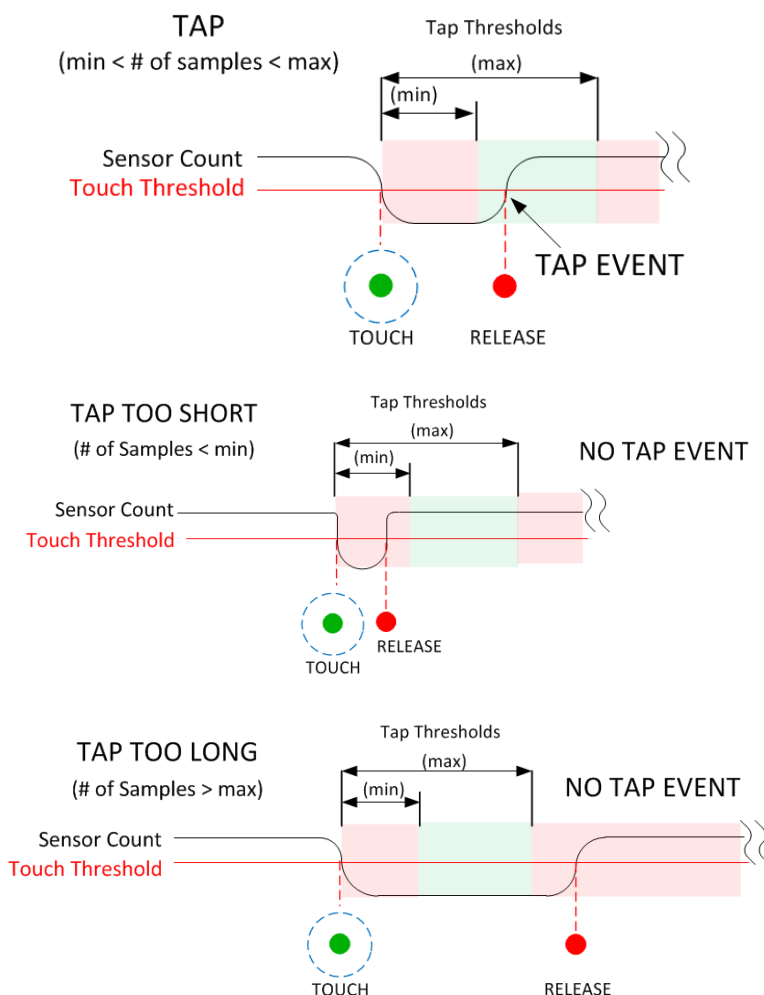
For specific code implementation, refer to the source code file, `gesture.c`

GESTURES

The gesture algorithm supports Tap, Double-Tap, Tap-Hold, Swipe Left and Swipe Right. Adding additional gestures is left up to the reader to develop.

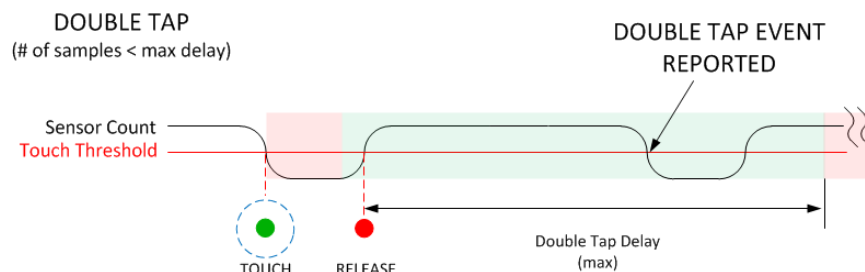
TAP GESTURE

A Tap gesture is defined as a finger that touches the sensor, followed by a release. The length of the tap is controlled by the number of samples between the touch and release. These are the tap's min and max threshold parameters and adjusting these allows the application to respond to a valid tap and ignore the event if the tap is either too short or too long. Adjusting the min and max thresholds allow the designer to create the proper touch and feel for their application.



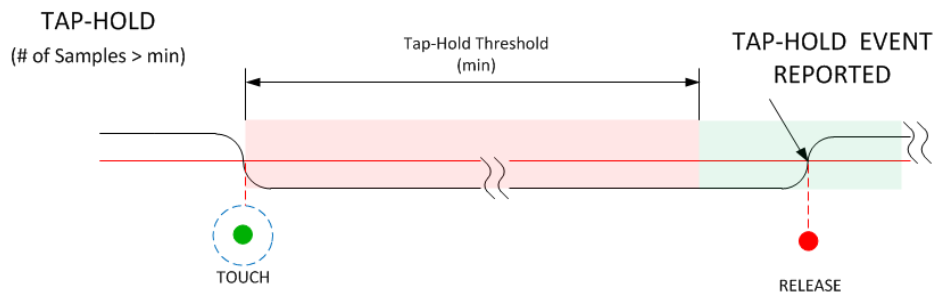
DOUBLE TAP GESTURE

A double-tap gesture is defined as a tap followed by a delay, then a second tap. A valid double-tap event is when the second tap follows the first tap, but before the end of the double-tap delay maximum limit. Adjusting the double-tap delay controls the response rate of the double-tap.



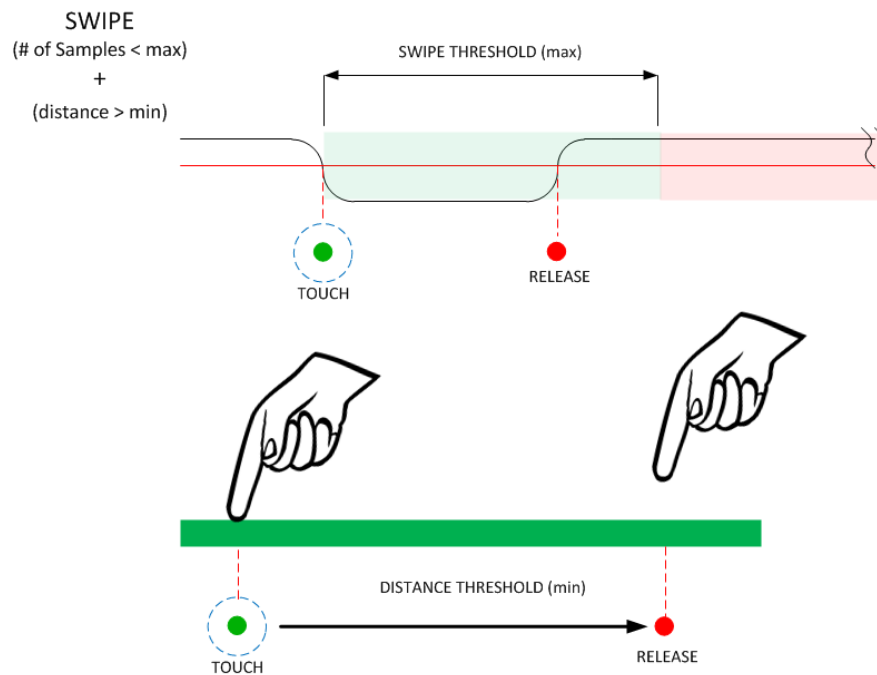
TAP-HOLD GESTURE

A tap-hold gesture is defined as a finger that touches the sensor, but not releasing until sometime later, after the tap-hold minimum threshold. Adjusting the tap-hold parameter controls the length of time a finger must touch the sensor before releasing. Releasing the finger before the threshold could be interpreted as a single tap by the algorithm, depending on the tap max parameter.



SWIPE GESTURE

A swipe gesture is defined as a finger that touches the sensor at one position on the sensor, then the finger moves to another location before releasing. For a valid swipe, the finger must move a minimum distance as well as within a maximum swipe time. If the swipe is too slow or the distance is too short, the gesture is ignored.



HOW TO ADD GESTURE CAPABILITY

The typical method to integrate gesturing is using a software callback function. A user can place their application specific code within a callback and register the callback as part of the application's initialization. In this way, each time a sensor is measured and processed, the user's callback function executes automatically. This is the perfect spot to include a call to the gesture algorithm.

The example project that accompanies this paper illustrates these steps:

- Add `gesture.c` and `gesture.h` to your project
- If you don't already have a callback for your slider, create one
- If you add a callback, don't forget to register the callback during initialization
- In the slider callback, add a line that calls `sliderGesture()`

Or if you want to see an example project already setup, use project:

- Import "Slider-Gestures-demo" project into CCS

That's it!

The `sliderGesture()` function returns the gesture.

The example includes some debug code in the callback that allows the gesture to be trapped, which can be useful while setting up the gesture parameters. Note, you will have to change the optimization settings for the file "`demo.c`" = *Off* in order for the breakpoints to work properly.

Change the file `demo.c` optimization settings = off so the breakpoints will work reliably.

PROJECT SETTINGS AND METRICS

The example gesture project was developed with the following tools and settings:

- Code Composer Studio 7.4.0.00015
- Compiler optimization setting = -O3
- Started with CAPTIVATE-BSWP example in CapTivate Design Center 1.60.00.00
- Gesture algorithm memory usage =

FILE: `gestures.obj`
CODE size (bytes): 330
CONST size (bytes): 30

