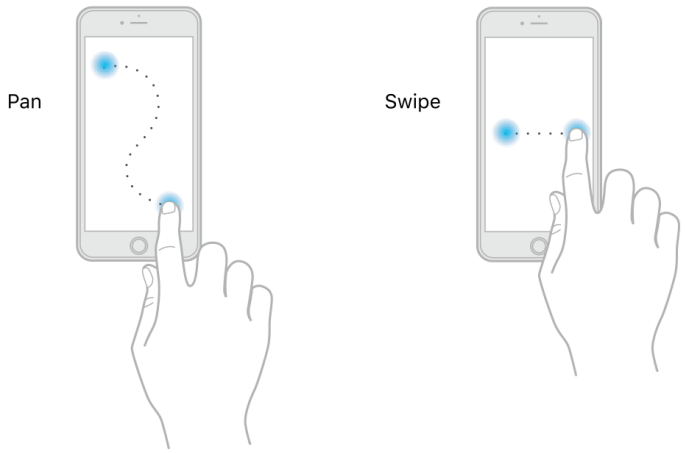
Handling Pan and Swipe Gestures

On This Page

Pan and swipe gestures track the movement of the user's fingers around the screen. These gestures typically track one finger but may track multiple fingers.

- A *pan gesture* occurs any time the user moves one or more fingers around the screen. A screen-edge pan gesture is a specialized pan gesture that originates from the edge of the screen. Use the `UIPanGestureRecognizer` class for pan gestures and the `UIScreenEdgePanGestureRecognizer` class for screen-edge pan gestures.
- A *swipe gesture* occurs when the user moves a single finger across the screen in a specific horizontal or vertical direction. Use the `UISwipeGestureRecognizer` class to detect swipe gestures.

Figure 3-1 Pan and swipe gestures



You can attach a gesture recognizer in one of these ways:

- Programmatically. Call the `addGestureRecognizer:` method of your view.
- In Interface Builder. Drag the appropriate object from the library and drop it onto your view

Handling a Pan Gesture

Use pan gesture recognizers for tasks that require you to track the movement of the user's fingers onscreen. You might use a pan gesture recognizer to drag objects around in your interface or update their appearance based on the position of the user's finger. Pan gestures are continuous, so your action method is called whenever the touch information changes, giving you a chance to update your content.

A pan gesture recognizer enters the `UIGestureRecognizerStateBegan` state as soon as the position of one finger changes. After that initial change, subsequent changes cause the gesture recognizer to enter the `UIGestureRecognizerStateChanged` state. When the user's fingers lift from the screen, the gesture recognizer enters the `UIGestureRecognizerStateEnded` state.

To simplify tracking, use the pan gesture recognizer's `translationInView:` method to get the distance that the user's finger has moved from the original touch location. At the beginning of the gesture, a pan gesture recognizer stores the initial point of contact for the user's finger. (If the gesture involves multiple fingers, the gesture recognizer uses the center point of the set of touches.) Each time that finger moves, the `translationInView:` method reports the distance from the original location. You can also reset the distance using the `setTranslation:inView:` method, which can simplify your code.

Listing 3-1 shows the action method for a pan gesture recognizer that lets the user drag a view around the screen. Touching and dragging the view results in multiple calls to this method to update the position of the view. To simplify the code, the method adds the distance travelled—obtained from the `translationInView:` method—to the view's current position. After updating the view, the method resets the travelled distance back to zero, causing the next call to `translationInView:` to report only the distance from the current location.

Listing 3-1 Dragging a view around the screen

```
@IBAction func panPiece(_ gestureRecognizer : UIPanGestureRecognizer) {
    // Move the anchor point of the view's layer to the touch point
```

```
    // so that moving the view becomes simpler.
    let piece = gestureRecognizer.view
    self.adjustAnchorPoint(gestureRecognizer: gestureRecognizer)

    if gestureRecognizer.state == .began || gestureRecognizer.state == .changed {
        // Get the distance moved since the last call to this method.
        let translation = gestureRecognizer.translation(in: piece?.superview)

        // Set the translation point to zero so that the translation distance
        // is only the change since the last call to this method.
        piece?.center = CGPoint(x: ((piece?.center.x)! + translation.x),
                                y: ((piece?.center.y)! + translation.y))
        gestureRecognizer.setTranslation(CGPoint.zero, in: piece?.superview)
    }
}
```


If the code for your pan gesture recognizer is not called, check to see if the following conditions are true, and make corrections as needed:

- The `userInteractionEnabled` property of the view is set to `YES`. Image views and labels set this property to `NO` by default.
- The number of touches is between the values specified in the `minimumNumberOfTouches` and `maximumNumberOfTouches` properties.
- For a `UIScreenEdgePanGestureRecognizer` object, the `edges` property is configured and touches started at the appropriate edge.

Handling a Swipe Gesture

A `UISwipeGestureRecognizer` object tracks the motion of the user's finger across the screen either horizontally or vertically. A swipe requires the user's finger to move in a specific direction and not deviate significantly from the main direction of travel. (The direction and number of fingers required for the gesture are configurable.) Swipe gestures are discrete, so your action method is called only after the gesture is recognized successfully. As a result, swipes are most appropriate when you care only about the results of the gesture and not about tracking the movement of the user's finger.

Listing 3-2 shows a skeletal action method for a swipe gesture recognizer. You would use a method like this to perform a task when the gesture is recognized. Because the gesture is discrete, the gesture recognizer does not enter the began or changed states.

Listing 3-2 Performing a task in response to a swipe

```
@IBAction func swipeHandler(_ gestureRecognizer : UISwipeGestureRecognizer) {
    if gestureRecognizer.state == .ended {
        // Perform action.
    }
}
```

If the code for your swipe gesture recognizer is not called, check to see if the following conditions are true, and make corrections as needed:

- The `userInteractionEnabled` property of the view is set to `YES`. Image views and labels set this property to `NO` by default.
- The number of touches is equal to the value specified in the `numberOfTouchesRequired` property.
- The direction of the swipe matches the value in the `direction` property.