PrimeSense

Giving Devices a Prime Sense

Prime Sensor™ NITE 1.3 Controls Programmer's Guide

Version 1.0

Disclaimer and Proprietary Information Notice

About This Guide

This guide's is intended for software developers using the NITE framework.

This guide contains the following chapters:

- **Chapter 1, Introduction,** on page 1-1, provides basic information about NITE, its concepts and functionality.
- **Chapter 2, Sessions,** on page 2-4, defines the Session concept, with its various states.
- **Chapter 3, NITE Controls,** on page 3-7, defines a Control, and show basic usage.
- **Chapter 4, Flow and NITE tree,** on page 4-10, defines Flow, and introduces the NITE tree.
- **Chapter 5, Compound Controls and Messages,** on page 5-15, describes how to create your own control, and expands on the flow, introducing the Messages that implement it.
- **Chapter 6, NITE Samples,** on page 6-17, describes the samples provided with NITE.
- **Appendix A, Licenses,** on page 6-21, mentions the licenses of products used in NITE.

Table of Contents

Table of Figures

Support

PrimeSense has made a significant investment in the development of the user-friendly software and hardware development environment provided by the Prime Sensor™ Development Kit (PSDK). This development kit is accompanied by documentation that teaches and guides developers through the process of developing applications.

PrimeSense also provides an online technical support service for customers. This service provides useful tips, as well as quick and efficient assistance, in order to enable you to quickly resolve any development issues that may arise.

Getting Technical Support

PrimeSense's primary support email address is support@primesense.com. Each email that is received is automatically forwarded to a relevant support engineer and promptly answered.

Glossary

Term	Description
OpenNI	Standard Natural Interface Framework.
Prime Sensor™	The brand name behind PrimeSense's products. It refers to the reference design for a 3D sensor.
Prime Sensor™ IC	The chip developed by PrimeSense that is implemented in the 3D camera.

This page was intentionally left blank.

1 Introduction

1.1 What is NITE

NITE is a toolbox to allow application to build flows based on the user's hands movement. The hand movement is understood as gestures and is tracked, to provide 'hand points'.

1.2 Prerequisites

NITE works over OpenNI. OpenNI documentation is provided separately.

PrimeSense provides software implementations for the all OpenNI modules. Those implementations, referred to as “NITE algorithms”, are explained in detail in a separate document. In the future, those modules will be implemented in hardware.

Note that the hand tracking module (hand generator) supplied provides a single hand point.

1.3 Software Overview

NITE works with the OpenNI framework and consists of the following layers:

- OpenNI Modules: The OpenNI modules supported by NITE are Gesture Generator and Hands Generator. In addition, NITE provides samples that show a Scene Analyzer module and a User Generator, with its Skeleton Capability.
- OpenNI infrastructure: See OpenNI document.
- Control Management: Receives a stream of points and routes them to the appropriate NITE control.
- Controls: Each control receives a stream of points and translates it into a meaningful action specific to that control. The control then calls a callback from the application that can change the current active control in the control management layer, thus defining the application flow.

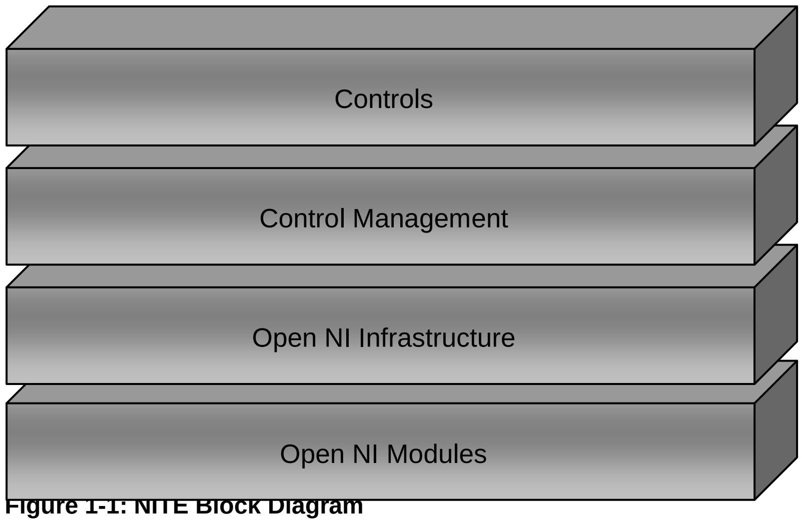


Figure 1-1: NITE Block Diagram

1.4 Programmer Tutorial

This chapter provides a simple step-by-step tutorial describing how to use the NITE infrastructure.

1.4.1 Quick Start

This section describes how to create a simple project using NITE. In order for NITE to run properly, OpenNI must be installed on your machine with at least one instance of each required module (the required modules are described in the Section **Error! Reference source not found.**, **Error! Reference source not found.**).

OpenNI and its modules are automatically installed during NITE installation.

1.4.1.1 Creating an Empty Project that Uses NITE

1.4.1.1.1 Windows

This section describes how to set up your environment for developing your own applications using NITE in Windows.

To create a simple project using NITE:

1 Create a new Visual Studio project or open an existing one with which you want to use the NITE.

2 In the Visual Studio menu, open the Project menu and select Project Properties.

3 In the C/C++ section, under the General node, locate the Additional Include Directories and add the value of the OPEN_NI_INCLUDE environment variable (the default location is C:\Program files\OpenNI\Include) and the value of the XN_NITE_INSTALL_PATH environment variable followed by \Include (the default location is C:\Program files\Prime Sense\NITE\Include).

4 In the Linker section, under the General node, locate the Additional Library Directories and add the value of the OPEN_NI_LIB environment variable (the default location is C:\Program Files\OpenNI\Lib) and the value of the XN_NITE_INSTALL_PATH environment variable followed by \Lib (the default location is C:\Program Files\Prime Sense\NITE\Lib).

5 In the Linker section, under the Input node, locate the Additional Dependencies and add the OpenNI.lib XnVNite.lib libraries to the list.

6 In the Debugging section, locate the Working Directory and set it to ../bin/Debug.

7 Be sure to add the Additional Include and Library directories to both your Release and Debug configurations.

1.4.1.1.2 Linux

Currently, development in LINUX is not supported.

2 Sessions

2.1 What is a session

A session is a state in which the user is in control of the system using hand points. While in the session, hand points are tracked, and have persistent IDs.

2.2 Session states

There are 3 possible states:

2.2.1 Not in session

In this state, we are looking to start a new session by identifying a gesture, which we will refer to as the 'focus gesture'. Once this gesture is recognized, the state changes to ‘in session’.

2.2.2 In session

In this state, a focus gesture was already identified, and hands are being tracked.

2.2.3 Quick refocus

In this state, we were in a session, but no hands points are identified, either intentionally or unintentionally. We don't want to stop the session yet, but give a grace period to restart the session with a different, perhaps easier, gesture, which we will refer to as the 'quick refocus gesture'. This state is optional.

2.3 Session state flow

At the beginning, the state is 'not in session', and we look for a gesture. When a focus gesture is identified, the state changes to 'in session', and we begin tracking the hand that performed the identified focus gesture. When there are no more hands (all hands are lost), the state changes to 'not in state' if the quick refocus state is disabled, or to 'quick refocus state' if it is enabled. In this state, we look for both the focus gesture(s) and the quick refocus gesture. If any of them is identified before a certain configurable timeout is reached, the state changes to 'in session'. If none of them is identified before the timeout, the state changes to 'not in session'.

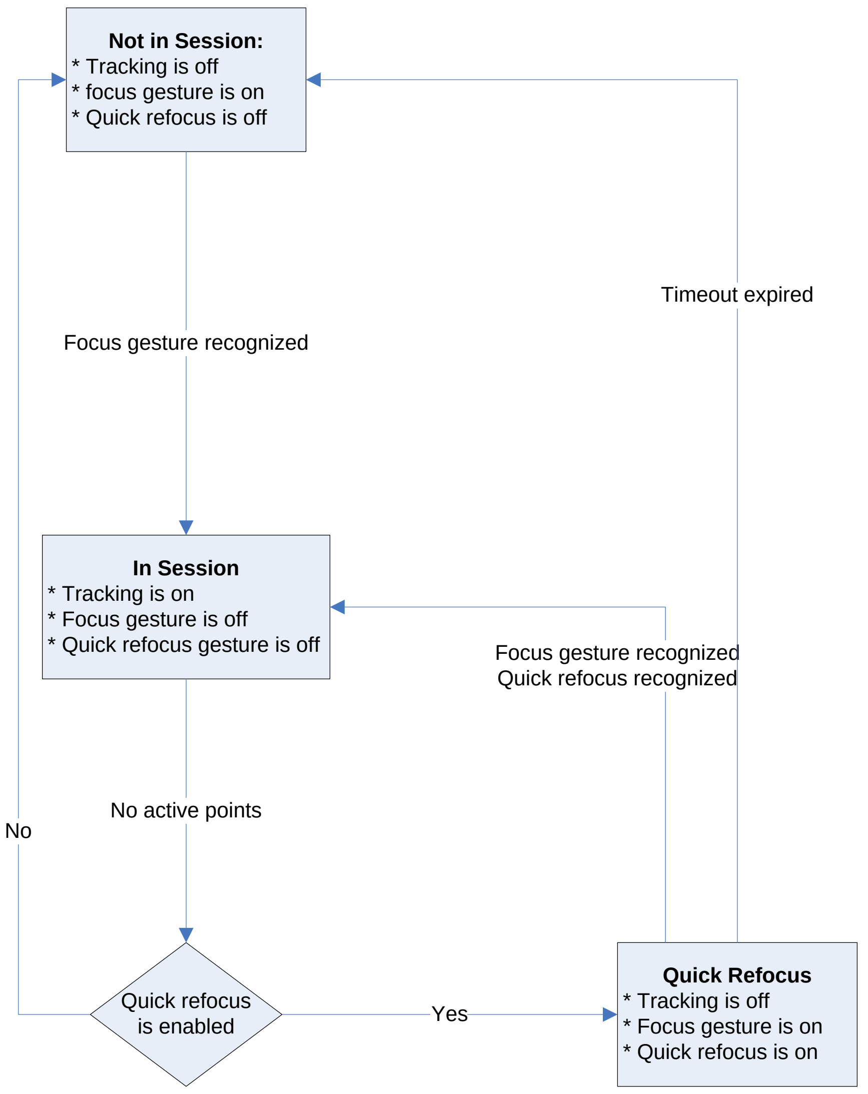


Figure 2-1: Session state automaton

2.4 Relation to OpenNI

OpenNI modules may be used for recognition of gesture and tracking of hand points.

2.5 Session Manager

XnVSessionManager is the NITE object that handles the session. It performs the logic described in section 2.3, turns on and off the gesture recognition when needed.

When in 'in session' state, and there are active hand points, it groups them together and sends them to interested controls.

2.6 Non-OpenNI gestures

In order to use a user-defined gesture, which isn't an OpenNI module, the gesture should inherit from XnVGesture and pass its object to XnVSessionManager’s SetGesture(XnVGesture* pGesture).

2.7 Usage examples

2.7.1 Session callbacks

```
void XN_CALLBACK_TYPE SessionStart(const XnPoint3D& pFocus, void* UserCxt)
{
    // Session started - do whatever
}
void XN_CALLBACK_TYPE SessionEnd(void* UserCxt)
{
    // Session ended - do whatever
}
```

2.7.2 Initialization

```
xn::Context context; // OpenNI.
XnVSessionManager sessionManager;
XnStatus rc = sessionManager.Initialize(&context, "Wave,Click", "RaiseHand");
```

2.7.3 Registration to session callbacks

```
sessionManager.RegisterSession(NULL, &SessionStart, &SessionEnd);
```

2.7.4 Main loop

```
while (1)
{
    context.WaitAndUpdateAll(); // OpenNI - can use any Wait function
    sessionManager.Update(&context);
}
```

3 NITE Controls

3.1 What are controls

Controls are objects that receive specific types of data, and perform some action on that data.

The most common controls in NITE are the point controls.

Point controls receive the current active hand points from some source (for example, the session manager), and try to understand these points as some action.

Controls have events, to which callbacks can be registered. When a control recognizes the movement it is supposed to, it will call all registered events.

3.2 Point Controls

Point controls allow registration to when new points are created, existing points move and when points that existed disappear.

3.3 How to use controls

Controls are actually listeners, getting data every frame. They are connected to a source of their required data.

For example, when a Point Controls is connected to a Session Manager, it gets the active hand points each frame, and can perform its action.

Each control allows registration to control-specific events. The registered callback functions will be called when that event happens.

For instance, the Push Detector has a 'Push' event, so all registered callbacks are called when the Push Detector recognizes a push motion.

3.4 List of controls in NITE

3.4.1 Push Detector

This point control tries to recognize hand point movement as a push motion, which is a continuous motion towards the sensor and back again.

For example, in a picture album application, push can be used to select an album.

3.4.2 Swipe Detector

This point control tries to recognize hand point movement as a swipe motion, either up, down, left or right. A swipe motion is a short movement in a specific direction, followed by the hand resting.

For example, in a picture album application, swipes to the left and right can be used to move from one picture to another.

3.4.3 Steady Detector

This point control tries to recognize when a hand point is steady for some time. Steady means the hand doesn't move (or almost doesn't move), it's movement delta variance being almost 0.

Steady is mostly useful between other controls, to make sure the next control starts from a resting hand.

3.4.4 Wave Detector

This point control tries to recognize hand point movement as a wave motion. A wave is a number of direction changes within a timeout. By default 4 direction changes are needed to identify a wave.

3.4.5 CircleDetector

This point control tries to recognize hand point movement as a circular motion.

The Circle Detector needs a full circle in either direction in order to start generating output. Clockwise direction is considered the 'positive' direction, and anti-clockwise is considered the 'negative'.

3.4.6 SelectableSlider1D

This point control tries to recognize hand point movement as a slider, aligned to any of the 3 axes - X (left-right), Y (up-down) or Z (close-far). This slider is divided equally into a number of areas, each area defining a single item in the slider. It can be used to implement menus, with each item being a single menu option.

Its allows registration to events for when the hand is over a different item, when that item is selected (by movement in either of the 2 axes that aren't the primary axis), and for each frame, a value between 0 and 1, indicating where the hand point is relative to the set ends of the slider.

3.4.7 SelectableSlider2D

This point control tries to recognize hand point movement as a 2D slider, on the X-Y plane.

It allows registration to events for when the hand is over a different item, when that item is selected (by movement along the Z axis), and for each frame, two values between 0 and 1, indicating where the hand point is in both axes, relative to the ends of the slider.

3.5 Primary Point

All the aforementioned controls work on one of the hand points, not all of them. This point is the Primary Point. It is initially the first one recognized (possibly the one that performed the focus gesture). If the primary point is no longer available (hand point isn't identified anymore), and other points exist, one of those points will take over as the primary point. The primary point is set by the session manager.

In addition to the 3 types of events described in 3.2, there are 4 more types, that are specific to the primary point. These are for primary point creation (most likely, session has started, or control turned active), primary point update, primary point destruction (most likely, session has ended, or control turned inactive) and primary point replaced, which means the primary point is no longer available, but another hand point exists and has taken over as the primary point.

3.6 Full APIs

API reference for NITE is provided separately.

3.7 Examples

```
void XN_CALLBACK_TYPE Push_Pushed(XnFloat fVelocity, XnFloat fAngle, void* cxt)
{
    // A push was detected. Do whatever.
}

XnVPushDetector pushDetector;
pushDetector.RegisterPush(NULL, &Push_Pushed);
sessionManager.AddListener(&pushDetector);
```

4 Flow and NITE tree

4.1 NITE flow

When NITE controls are connected directly to a Session Manager, it means that all the controls are always active, meaning they get hand points whenever there are any hand points.

A flow can be defined, so that controls receive data only when needed, when they are meaningful.

4.2 Flow objects

Flow objects are object that determine the flow of data, thus setting the current state of the application.

4.3 NITE tree

The flow of data (primarily points) between NITE objects can be described by a graph, most commonly that graph being a tree.

This NITE tree describes the flow of data at any single point in time.

4.4 List of Flow objects in NITE

4.4.1 Flow Router

The flow router object sends all the data it receives to a single object that is connected to it. This single object may be changed. An object that is connected to the flow router it considered to be 'active', as it will receive the data. An object that was connected to the flow router but isn't anymore is considered to be 'inactive'.

4.4.2 Broadcaster

The broadcaster object sends all data it receives to all objects that are connected to it.

4.4.3 Point Denoiser

This object works specifically on points. It performs a smoothing algorithm on all points, elimination small movements that are most likely creates by the depth or by the user's hand not being entirely stable.

4.4.4 Point Area

This object works specifically on points. It passes on only points that are within the configured 3D area.

4.4.5 Virtual Coordinates

This object works specifically on points. It learns the user's natural plane, and then transforms all consequent points to be relative to that plane.

4.5 Examples

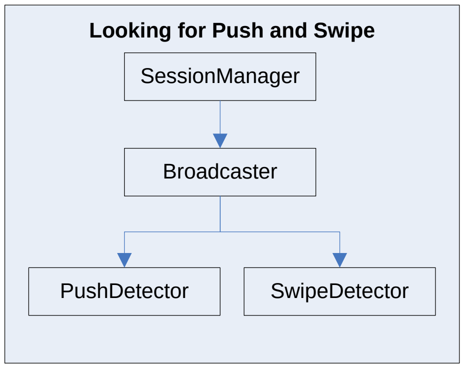


Figure 4-1: Using Broadcaster

```
XnVBroadcaster broadcaster;
XnVPushDetector pushDetector;
XnVSwipeDetector swipeDetector;

broadcaster.AddListener(&pushDetector);
broadcaster.AddListener(&swipeDetector);
sessionManager.AddListener(&broadcaster);
```

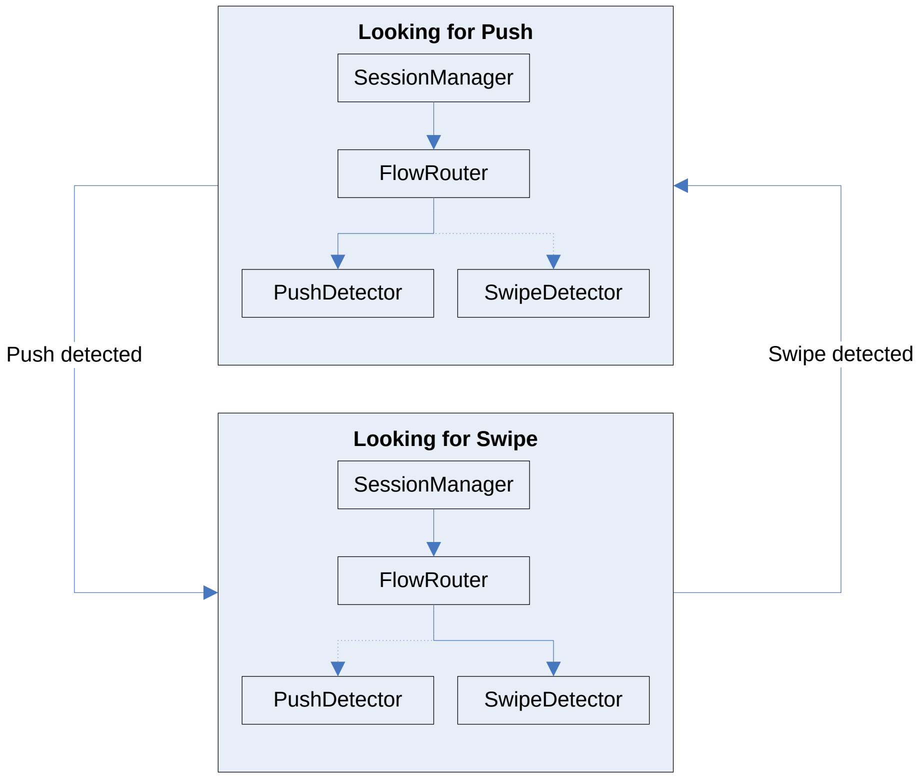


Figure 4-2: Using Flow Router

```
XnVFlowRouter g_flowRouter;
XnVPushDetector g_pushDetector;
XnVSwipeDetector g_swipeDetector;

void XN_CALLBACK_TYPE Push_Pushed(XnFloat fVelocity, XnFloat fAngle, void* cxt)
{
    g_flowRouter.SetActive(&g_swipeDetector);
}
void XN_CALLBACK_TYPE Swipe_SwipeUp(XnFloat fVelocity, XnFloat fAngle, void* cxt)
{
    g_flowRouter.SetActive(&g_PushDetector);
}
…
    g_pushDetector.RegisterPush(NULL, &Push_Pushed);
    g_swipeDetector.RegisterSwipeUp(NULL, &Swipe_SwipeUp);
    g_flowRouter.SetActive(&g_pushDetector);
    sessionManager.AddListener(&g_flowRouter);
```

4.6 Multi-threaded Support

Multi-threaded support is provided in the XnVMessageListener object, which is the base of all NITE controls. An XnVMessageListener object has a single thread in which it runs, which is by default, the thread in which it was created.

XnVMessageListener’s multi-threaded support performs the following:

- If the event was received in the execution thread, it is handled immediately.
- If the event was received in a thread that is not the execution thread, it is added to an internal operation queue.

When XnVMessageListener is run in a thread different from the one of the generator that sends it messages, ,then that thread must call the Run() method, which reads events from the operation queue and handles them.

There is also the option to have the XnVMessageListener allocate a specific thread, in which the main loop is read from the queue. This is done by calling RunAsThread(). This thread is terminated upon destruction of XnVMessageListener.

This capability is deactivated by default, and can be optionally activated for each XnVMessageListener specifically and separately.

4.7 Multi-process Support

Multi-process support is provided by two objects, encapsulating a single server (XnVMultiProcessFlowServer) and multiple clients. This object is an NITE control, which receives messages and updates a shared memory section. Each client is an XnVMultiProcessFlowClient object, which is an can be used instead of the Session Manager as the head of the NITE tree. It reads from the shared memory section and sends events to all its registered listeners.

Multi-process support supplies the client with points and session events only. In addition, all communication is one-sided: Server to client only. Client can't end the session in the server without additional implementation by the user.

4.8 Log

NITE uses the OpenNI Log mechanism, and defines the following new masks:

- XNV_NITE_MASK_CREATE: Logs the creation and destruction of Generators and Listeners.
- XNV_NITE_MASK_FLOW: Logs any message sent by any Generator and any message received by any Listener.
- XNV_NITE_MASK_CONNECT: Logs the connection and disconnection of Listeners to/from Generators and the change of active controls in Flow Routers.
- XNV_NITE_MASK_POINTS: Logs the creation and destruction of points.
- XNV_NITE_MASK_SESSION: Logs the changes of the Session state.
- XNV_NITE_MASK_MT_QUEUE: Logs access to the multi-thread queue in multi-threaded mode.

- XNV_NITE_MASK_EVENTS: Logs control events.

5 Compound Controls and Messages

5.1 Messages

The flow of data between NITE controls is done through messages, of type XnVMessage. Messages can be sent (by XnVMessageGenerators) and be listened to (by XnVMessageListeners). The base of all the point controls, which is XnVPointControl, is a XnVMessageListener.

Each control receives a message, checks if that message is of the expected type, and handles that message. In most cases, it will extract the actual data from the message, and either supply it for handling by a subclass, or analyze it and supply the subclass (or user callbacks) with specific occcurances.

For instance, an XnVPointControl receives an XnVPointMessage. It extracts an XnVMultipleHands object from it, which is a container of all the active hand points currently identified (each being a XnVHandPointContext). It also analyzes it, to find which points are new (thus calling the OnPointCreate callback), which already existing (resulting in the OnPointUpdate callback), and which points that were previously tracked are no longer identified (the OnPointDestroy callback). It also finds out which point is the primary point, and continues to call the relevant primary point callbacks.

5.2 How to create a control based on existing controls

Controls can be grouped together to create new controls. This can be done by holding a NITE tree inside your control, which receives its data from a previous node.

For example, the Boxes sample defines a control that when active looks for either Steady or Swipe, and also for Push, in order to leave that control. This is the Box control, and the advantage of having it as a single control is that having 3 boxes is as easy as one, creating additional instances of the same compound control

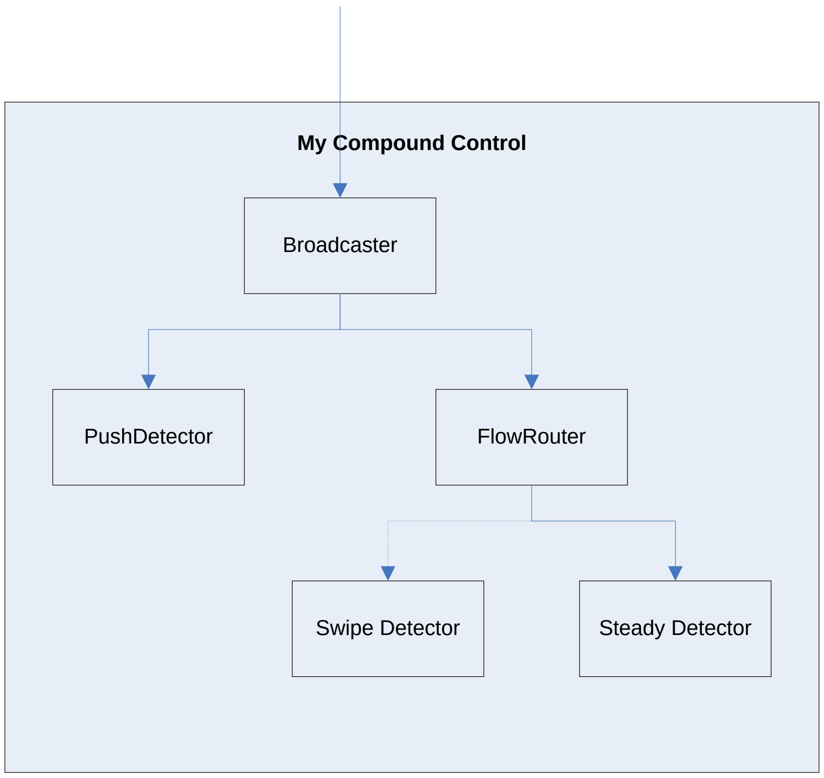


Figure 5-1: Compound Control

5.3 How to create a control that does something new

New controls can be created, based on existing messages, or on the events the basic controls externalize.

For example, the PointViewer sample receives point messages, saves them in a history buffer for each point, and draws them. The drawing happens in the function that handles the message, after letting its Point Control take care of the callbacks, in which the history buffers are constructed.

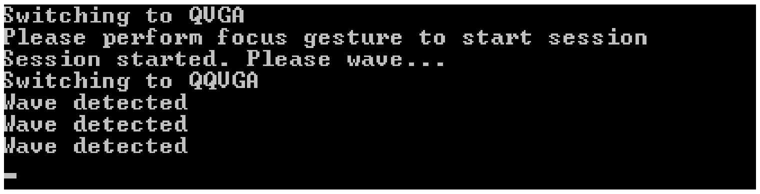
In that example, if we only wanted to store the history buffer without drawing it, or making the drawing the user's responsibility, we could only use the Point Control's callbacks, not dealing with the messages at all.

6 NITE Samples

This chapter describes the samples provided with NITE. The source code of these samples is provided in the NITE package.

6.1 SingleControl Sample

The example below demonstrates the identification of waves based on hand points. Output is all textual.



```
Switching to QVGA
Please perform focus gesture to start session
Session started. Please wave...
Switching to QQVGA
Wave detected
Wave detected
Wave detected
_
```

Figure 6-1: Wave Sample

6.2 CircleControl Sample

The example below depicts the identification of circular motions. A circle is drawn as the output of a circular motion.

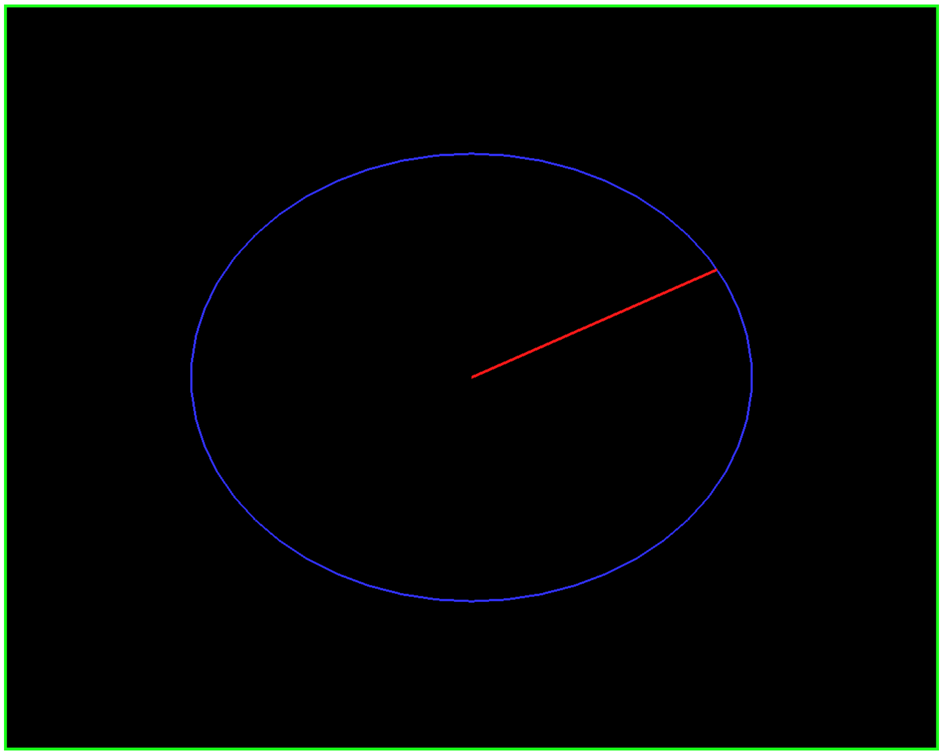


Figure 6-2: Circle Sample

6.3 PointViewer Sample

The example below depicts the tracking of hand points.

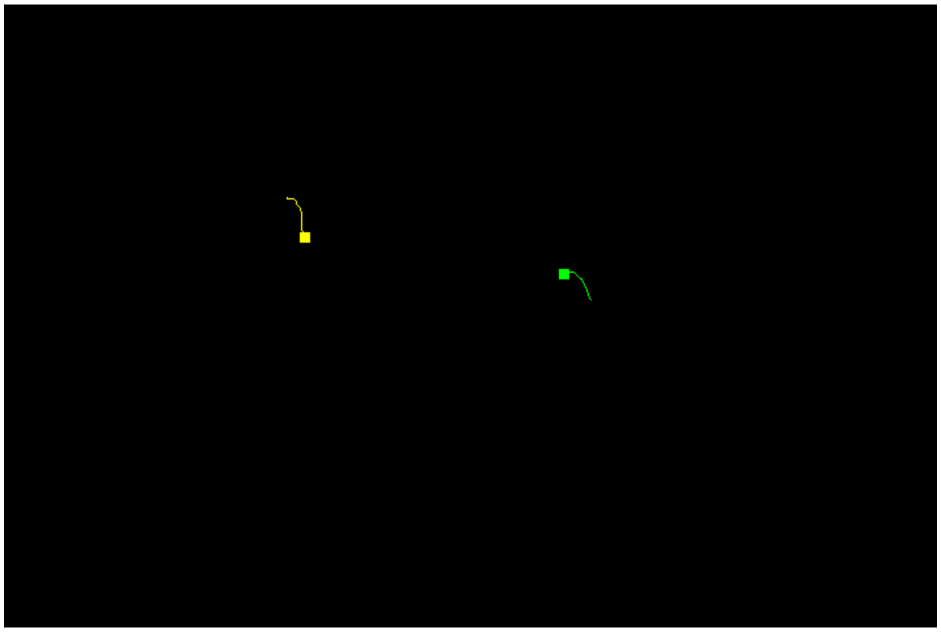


Figure 6-3: PointViewer Sample

6.4 Boxes Sample

The example below depicts the definition of user-specific listeners and a user-defined flow.

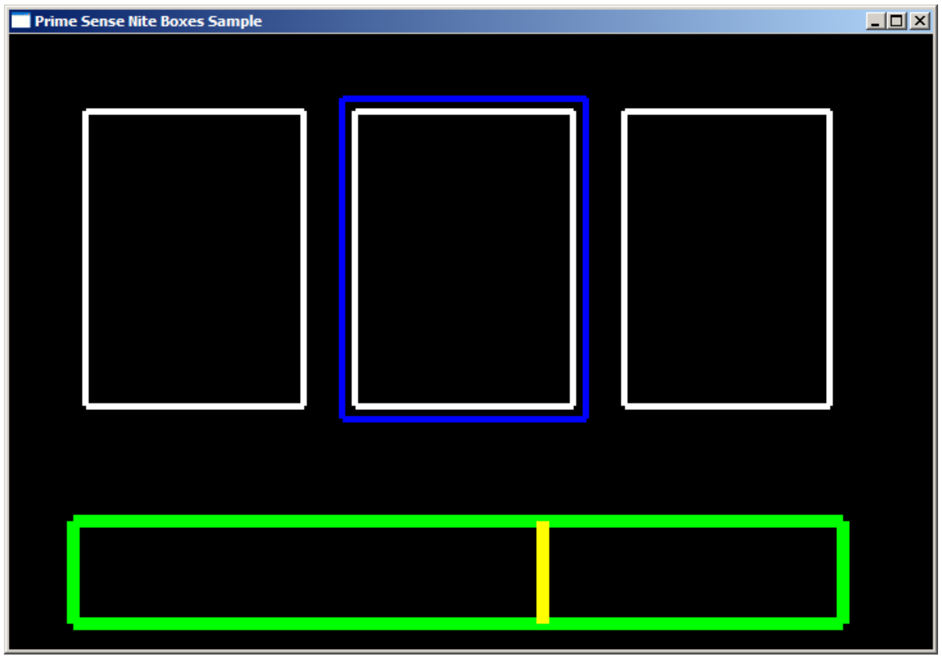


Figure 6-4: Boxes Sample

6.5 Stick Figure Sample

The example below depicts the extraction of users, their pixels and a skeleton representation.

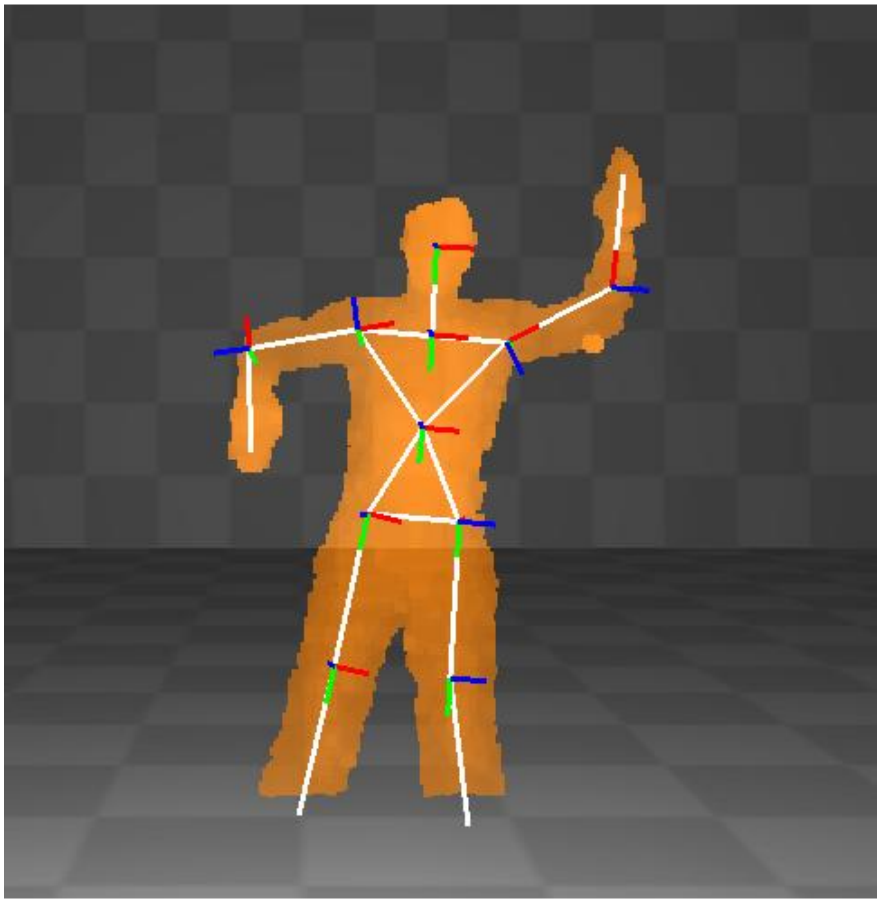


Figure 6-5: Stick Figure Sample

This page was intentionally left blank.

A Licenses

This software includes components from OpenCV and the PhysBAM libraries. Respective copyrights are reproduced below.

OpenCV

PhysBAM

This page was intentionally left blank.