VisionPro 3D-Locate

Developer’s Guide

January 5, 2015

Portions of the hardware and software provided by Cognex may be covered by one or more of the U.S. and foreign patents listed below as well as pending U.S. and foreign patents. Such pending U.S. and foreign patents issued after the date of this document are listed on Cognex web site at http://www.cognex.com/patents.

CVL

5495537, 5548326, 5583954, 5602937, 5640200, 5717785, 5751853, 5768443, 5825483, 5825913, 5850466, 5859923, 5872870, 5901241, 5943441, 5949905, 5978080, 5987172, 5995648, 6002793, 6005978, 6064388, 6067379, 6075881, 6137893, 6141033, 6157732, 6167150, 6215915, 6240208, 6240218, 6324299, 6381366, 6381375, 6408109, 6411734, 6421458, 6457032, 6459820, 6490375, 6516092, 6563324, 6658145, 6687402, 6690842, 6718074, 6748110, 6751361, 6771808, 6798925, 6804416, 6836567, 6850646, 6856698, 6920241, 6959112, 6975764, 6985625, 6993177, 6993192, 7006712, 7016539, 7043081, 7058225, 7065262, 7088862, 7164796, 7190834, 7242801, 7251366, EP0713593, JP3522280, JP3927239

VGR

5495537, 5602937, 5640200, 5768443, 5825483, 5850466, 5859923, 5949905, 5978080, 5995648, 6002793, 6005978, 6075881, 6137893, 6141033, 6157732, 6167150, 6215915, 6324299, 6381375, 6408109, 6411734, 6421458, 6457032, 6459820, 6490375, 6516092, 6563324, 6658145, 6690842, 6748110, 6751361, 6771808, 6804416, 6836567, 6850646, 6856698, 6959112, 6975764, 6985625, 6993192, 7006712, 7016539, 7043081, 7058225, 7065262, 7088862, 7164796, 7190834, 7242801, 7251366

OMNIVIEW

6215915, 6381375, 6408109, 6421458, 6457032, 6459820, 6594623, 6804416, 6959112, 7383536

Contents

Preface

This manual describes how the VisionPro 3D-Locate programming interface works and how you use them to solve vision applications.

Style Conventions Used in This Manual

This manual uses the following style conventions for text:

boldface	Used for .NET keywords, function names, class names, structures, enumerations, types, and macros. Also used for user interface elements such as button names, dialog box names, and menu choices.
italic	Used for names of variables, data members, arguments, enumerations, constants, program names, file names. Used for names of books, chapters, and sections. Occasionally used for emphasis.
`courier`	Used for .NET code examples and for examples of program output.
`bold courier`	Used in illustrations of command sessions to show the commands that you would type.
<italic>	When enclosed in angle brackets, used to indicate keyboard keys such as *<Tab>* or *<Enter>*.

Microsoft Windows Support

Cognex VisionPro software runs on different Microsoft Windows operating systems. In this documentation set, these are abbreviated to Windows unless there is a feature specific to one of the variants. Consult the *VisionPro Quick Reference*, available in hardcopy or online from the **Start** menu, for details on the operating systems, hardware, and software supported by that release.

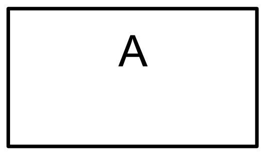
Software Diagramming Conventions

This manual uses the following symbols in class diagrams:

- **Classes** are shown as a box with the class name centered inside the box. For example, a class A with the C++ declaration

  ```
  class A{};
  ```

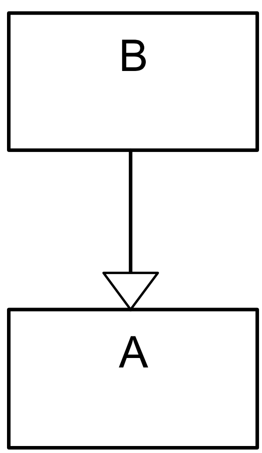
  is shown graphically as follows:

- **Inheritance** relationships between classes are shown using solid-line arrows from the derived class to the base class with a large, hollow triangle pointing toward the base class. For example, a class B that inherits from a class A with the declaration

  ```
  class B : public A {};
  ```

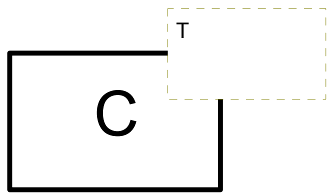
  is shown graphically as follows:

- **Template classes** are shown as a class box with a smaller, dotted-line rectangle representing the template parameter superimposed on the upper right corner of the class box. For example, a template class C with a parameter of type class T with the declaration:

```
template <class T>
class C{};
```

  is shown graphically as follows:



These symbols are based on the Unified Modeling Language (UML), a standard graphical notation for object-oriented analysis and design. See the latest *OMG Unified Modeling Language Specification* (available from the Object Management Group at http://www.omg.org) for more information.

About This Manual

Detailed information about VisionPro 3D-Locate API is provided in the following chapters:

- *3D Vision Overview* introduces concepts related to 3D coordinate spaces, calibration and triangulation.
- *3D Shapes, Graphics and Transforms* describes the mathematical foundations for 3D vision, including transformations and pose representations, as well as support functions such as 3D shapes, fitting functions, and graphics.
- *3D Calibration Tools* provides detailed information on the 3D calibration tools provided by VisionPro 3D Add In, including specific recipes and procedures for calibrating a 3D machine vision system.
- *Locating Objects in 3D* describes the triangulation and fitting tools that enable the generation of 3D poses for objects.

Cognex Offices

The following are the address and phone number of Cognex Corporate Headquarters, and the address of the Cognex web site:

Corporate Headquarters	Cognex Corporation Corporate Headquarters One Vision Drive Natick, MA 01760-2059 (508) 650-3000
Web Site	www.cognex.com

3D Vision Overview 1

VisionPro provides an interactive development environment for configuring acquisition and I/O, two-dimensional vision tools for analyzing images, and capabilities for image and graphics display. VisionPro also provides a full-featured toolkit that you can program in .NET using C#, VB.NET or managed C++.

The optional 3D-Locate API allows you to generate information about objects in three-dimensional space from two-dimensional images.

Creating a complete 3D application requires a combination of standard VisionPro features to be used along with the 3D-Locate API. The application acquires images of an object using multiple cameras and uses various pattern matching and shape-fitting vision tools to locate a set of common features. The 3D-Locate tools use the location of these found features to build data structures representing the object in three-dimensional space, and can return information regarding the pose of the object in the physical world.

This chapter contains the following sections:

- *Some Useful Definitions* on page 12 provides an overview of the chapter and defines some terms that you will encounter as you read.
- *3D Calibration* on page 14 provides an overview of 3D calibration, which lets you map 2D points from an acquired image to locations in a three-dimensional physical space.
- *Triangulation* on page 28 is a capability that lets you determine the three-dimensional location of points and three-dimensional position and orientation of objects in space based on data from multiple 2D images.
- *Robot (Hand-Eye) Calibration* on page 39 is a specialized calibration capability intended for use with robots and robot-mounted machine vision cameras.

The VisionPro installation includes a sample 3D application using Microsoft Visual Studio, located by default at *%VPRO_ROOT%\samples3D\Applications*. Copy the contents to a folder where you have write permission before you execute it.

Some Useful Definitions

This section defines some terms and concepts used in this chapter.

3D Pose The position and orientation of a 3D coordinate system within another 3D coordinate system. A pose comprises 6 degrees of freedom: X-translation, Y-translation, Z-translation, X-rotation, Y-rotation, and Z-rotation.

3D Position The location of a 3D point within a 3D coordinate system. A 3D position is represented by an x-value, y-value, and z-value.

3D Ray Geometric object defined by a 3D position (the starting point of the ray) and orientation. A ray extends infinitely from its origin.

3D-Calibrated Camera A camera for which a 3D calibration (both extrinsic and intrinsic) has been computed.

Raw2D Space Left-handed 2D coordinate space based on the pixels in an acquired image.

Camera3D Space A right-handed 3D coordinate space with its origin at the camera's optical convergence point, X- and Y-axes that are approximately parallel to and oriented in the same direction as the Raw2D coordinate system X- and Y-axes, and a Z-axis that extends along the optical axis away from the camera.

Camera2D Space The plane at Z=1 of Camera3D Space. When Camera2D space is viewed from the camera (Z < 1 and in the direction of the Camera3D positive Z axis) then Camera2D space appears as a left-handed 2D coordinate system. When Camera2D space is viewed in the direction of the Camera3D negative Z axis from a point in front of the camera (where Z > 1), then Camera2D Space appears as a right-handed 2D coordinate system.

Phys3D Space A right-handed 3D coordinate space initially defined by the fiducial mark on the calibration plate specified as having an origin-defining pose-type used to perform 3D calibration. This space can be defined by any coordinate frame in physical space.

Hand3D Space A right-handed 3D coordinate space defined by the end-effector on a robot. The position of the robot hand is reported by the robot controller as the pose of Hand3D space in RobotBase3D space. (Some robot manufacturers and integrators refer to this as *tool space*.)

RobotBase3D Space A right-handed 3D coordinate space defined by the robot manufacturer or integrator. It is typically associated with the robot base (the part of the robot that is rigidly fixed to the physical world).

Checkerboard Feature Extractor Software that locates all of the grid vertices in an image of a Cognex checkerboard calibration plate, along with the fiducial features that define the plate origin. The feature extractor constructs a list of correspondence pairs which associate the location in the image of each feature with its physical position, based on the physical grid pitch value that you supply.

Correspondence In triangulation, the association of a given feature location in one view of an object with the same feature's location in another view of the object.

Correspondence Pair The physical coordinates and image coordinates of a given calibration plate vertex.

Triangulation Establishing a 3D pose or position by computing the intersection points of sets of 3D rays.

3D Calibration

Three-dimensional calibration is a process that establishes a mathematical relationship between the 2D coordinate system associated with the pixels in an acquired image and a 3D coordinate system associated with the physical world in front of the camera. The initial definition of the 3D physical coordinate space is provided by the origin of a calibration plate.

Note Throughout this chapter the term *Raw2D space* refers to the 2D coordinate system established by an acquired image and the term *Phys3D space* refers to a 3D physical coordinate system established by the 3D calibration process. During 3D calibration, Phys3D space is defined by the calibration plate origin at calibration time, which is referred to as *CalPlate3D space*.

For any three-dimensional vision environment, 3D calibration must be performed for each camera, whose configuration is defined by the physical location of the camera in addition to the optical system used to form an image on the image sensor.

Once 3D calibration has been performed, the camera is *3D-calibrated* and has an associated *3D calibration object* of type **Cog3DCameraCalibration**

A 3D-calibrated camera lets you transform a 2D point from Raw2D space into a 3D ray in Phys3D space, as shown in Figure 1.

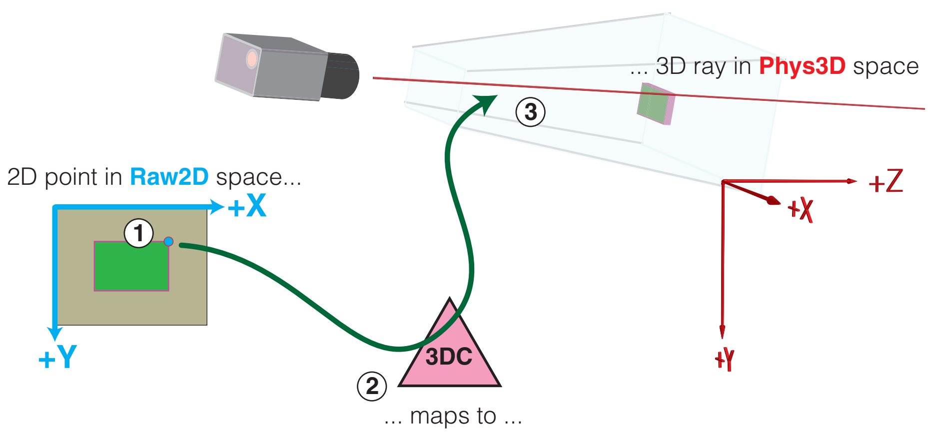


Figure 1. Transforming a 2D point from Raw2D space into a 3D ray in Phys3D space

A 3D calibration object also lets you transform a 2D point from Raw2D space into a 3D point in Phys3D space, as long as you are also able to supply a 3D plane in Phys3D space within which the 3D point lies.

In addition to transforming 2D image points to 3D rays and points, a 3D calibration object can also transform 3D points in Phys3D space to 2D points in Raw2D space, as shown in Figure 2.

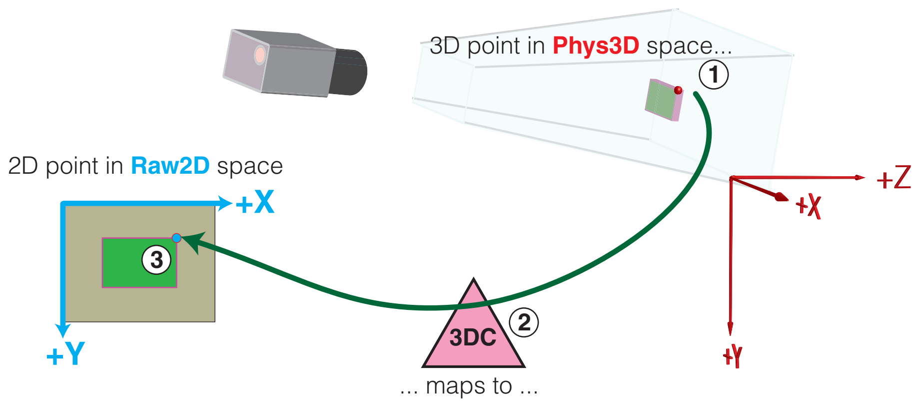


Figure 2. 3D calibration maps 3D points in physical space to 2D points in an image

Note Any single 2D point in Raw2D space always transforms to a single 3D ray in Phys3D space. A single 3D point in Phys3D space always transforms to a single 2D point in Raw2D space. Transforming a single 2D point from Raw2D space to a single 3D point in Phys3D space requires that you constrain the transformation by supplying a 3D plane.

How Does 3D Calibration Work?

Three-dimensional calibration can be done using a single camera or multiple cameras. Specific information on multi-camera calibration can be found in the section *Multi-Camera Calibration* on page 25.

Three-dimensional calibration works by acquiring a series of images of a calibration plate with the plate at different locations and orientations within a working volume using a camera with fixed optical and mechanical configuration. You supply the plate images, the physical spacing of the grid vertices, information about the pose types of the plates in the different images, and identify which plate view corresponds to the origin of the Phys3D space to the 3D camera calibration function.

VisionPro 3D-Locate supports the method **Cog3DCameraCalibrator.Execute()** for computing the 3D calibration. See the Programming Reference in the VisionPro online documentation for more information.

A 3D calibration object provides a transformation between the Raw2D space established by an acquired image and the Phys3D space established by the 3D calibration process.

Calibration Plate Requirements

Cognex 3D calibration supports the use of two types of calibration plates:

- Single-fiducial checkerboard plates
- DataMatrix calibration plates

Each type of plate is described in this section.

Single-Fiducial Checkerboard Plates

You can use a Cognex checkerboard calibration plate to perform 3D calibration. The Cognex checkerboard calibration plate includes a standard fiducial mark that defines the plate origin, as shown in Figure 3.

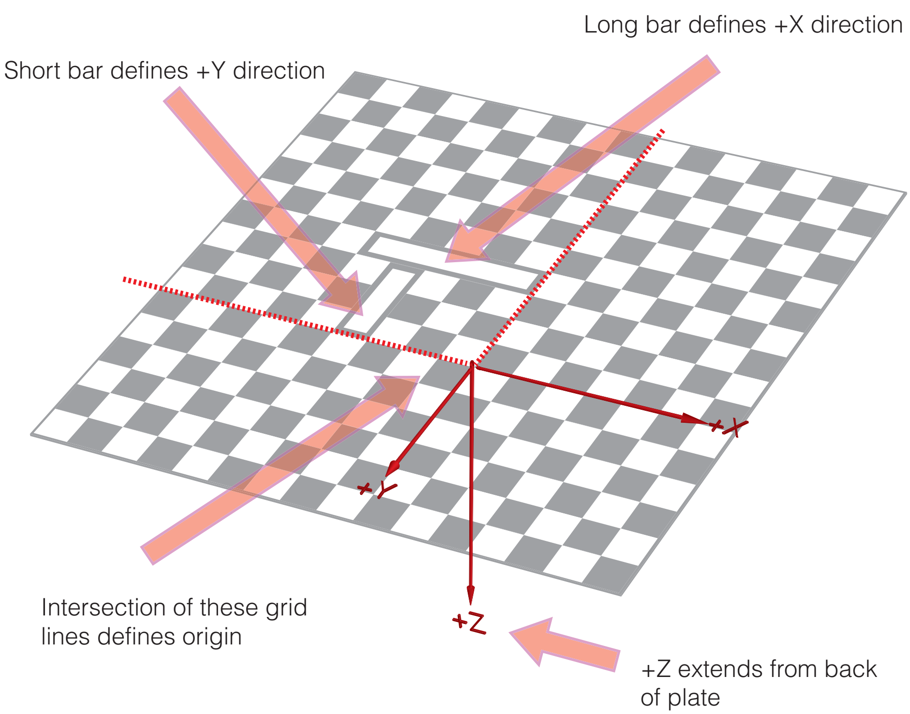


Figure 3. Calibration plate fiducial mark

DataMatrix Checkerboard Plates

Cognex 3D calibration supports the use of checkerboard calibration plates with multiple DataMatrix code fiducial marks. These plates use DataMatrix codes to label the locations of multiple grid vertices on the plate and establish the orientation and handedness of the plate space. The DataMatrix codes can also be used to encode the plate's grid pitch.

Note All Cognex-supplied DataMatrix plates encode the grid pitch; plates that you construct using Cognex-supplied CAD data may not include the grid pitch.

Figure 4 shows how multiple DataMatrix fiducial marks serve to label four vertices on a checkerboard calibration plate (in this case, a plate with a 2mm grid pitch).

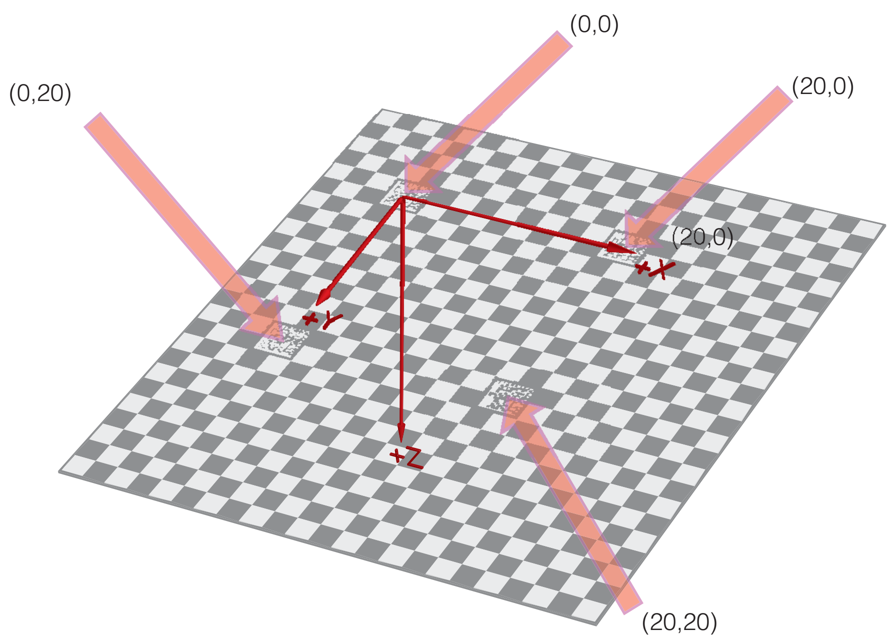


Figure 4. DataMatrix Calibration Plate

The plate origin (0,0) is establlished by the locations encoded in the DataMatrix fiducial marks. For all Cognex-supplied plates, the plate origin is located at the center of the fiducial mark establishing the (0,0) position, but there is no requirement that the plate origin be at the center of a fiducial mark (or even on the plate at all).

In all cases, the location specified by the DataMatrix code is at the center of the symbol, as shown in Figure 5.

The handedness and orientation of the plate coordinate space are defined by the orientation of the DataMatrix mark, as shown in Figure 5. The postive X- and Y-axis directions are established by the orientation of the finder pattern.

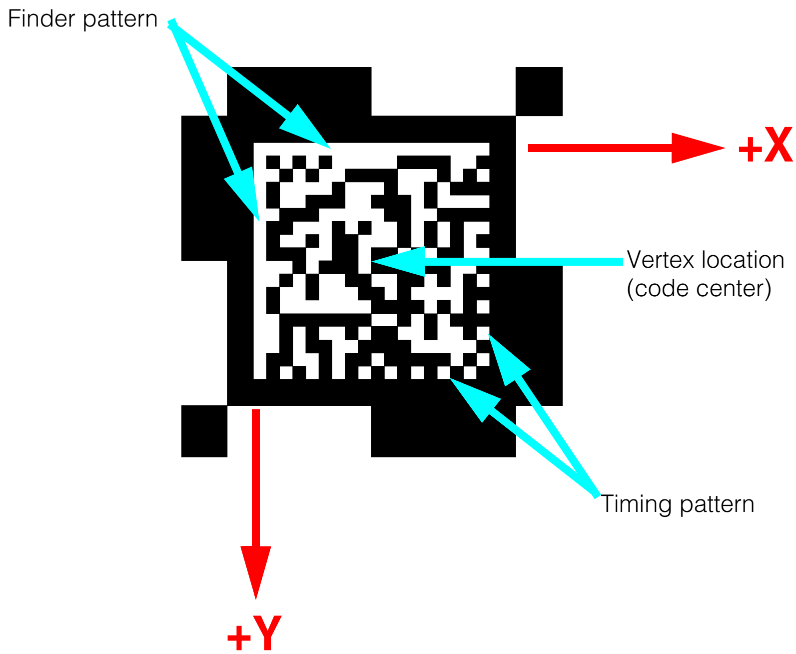


Figure 5. Coordinate axes defined by DataMatrix fiducial mark

As is the case with the single-fiducial plate, the positive Z-axis extends away from the back of the plate.

Note If the plate is mirrored (as would be the case if a transparent plate were viewed from behind), the calibration tool detects that the DataMatrix symbol is mirrored, and inverts the handedness of the coordinate system. In the case where a transparent plate is viewedfrom its back side, the positive Z axis is towards the camera.

Checkerboard calibration plates with DataMatrix fiducial marks provide several important advantages:

- The plates enable the use of the *Exhaustive Multi-Region* mode in 3D camera calibration. This permits calibration in cases where reflections or other occlusions hide regions of grid vertices. Using this mode, the calibration tool can make use of multiple regions of vertices, as long as each region contains at least one fiducial mark.
- The DataMatrix marks encode the plate pitch, so you do not need to include this information as part of your system configuration.
- When performing 3D calibration, as long as *any* fiducial on the plate is visible in an image, that image can be used for calibration. This is much less restrictive than the single-fiducial plate, where the single origin fiducial must be visible.

Note Cognex does not publish the specific encoding used for DataMatrix fiducial marks. Cognex can provide high-accuracy DataMatrix fiducial plates in a wide range of sizes, grid pitches, and materials. Contact your Cognex representative for more information.

Plate Poses

The minimum requirement for 3D calibration is that you acquire four images of the calibration plate, with the plate tilted at between 20° and 30° relative to the image plane and then rotated about the camera's optical axis between images (Cognex recommends a 90° rotation between images). You do not need to specify any information about the position or orientation of the plate in these views; the calibration software automatically calibrates the camera to the physical volume in which the plate is placed.

For best results, however, Cognex recommends that you supply a total of nine plate views. In addition to the four plate views described above, Cognex recommends that you provide five views in which the plates are parallel to each other and separated vertically by a known distance. You need to provide this distance, specified in the same units as the plate grid spacing, to the calibration function.

Note Each image in a plate view (the images from each camera of the plate at a given pose), should include the plate fiducial mark. If you are using a DataMatrix calibration plate, then the image should include *any* plate fiducial mark.

Finally, regardless of how many plate views you use, you must specify which plate view defines the origin of Phys3D space. If you are supplying the optional 5 stacked views, then the view that defines the Phys3D origin must be one of those views (typically it is the plate closest to the center of the working volume, view 5 in Figure 6).

Figure 6 shows both the required (views 1-4) and recommended (views 5-9) plate views for use with 3D calibration.

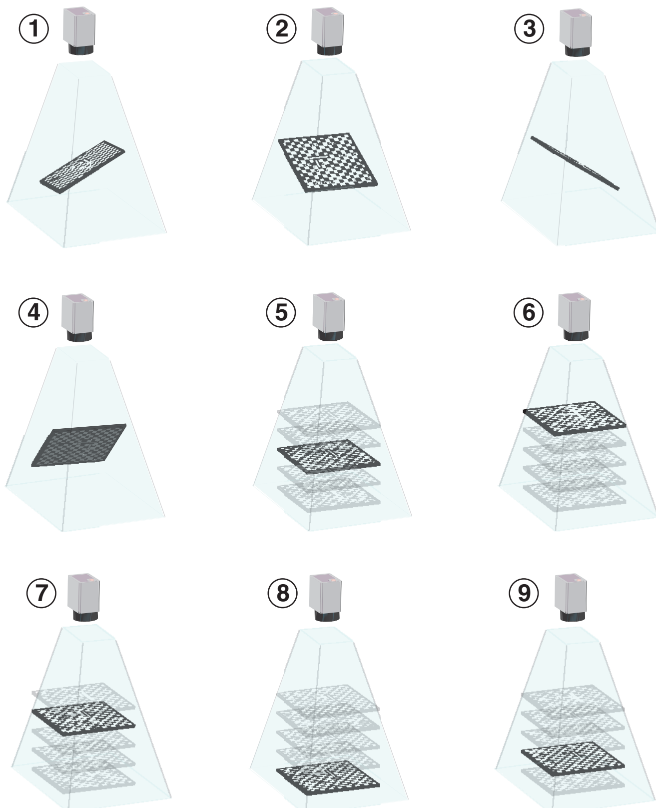


Figure 6. Required (1-4) and recommended (5-9) plate views for 3D calibration

3D Calibration Coordinate Spaces

As described in the section *3D Calibration* on page 14, a 3D calibration object provides a transformation between the Raw2D space established by an acquired image and the Phys3D space established by the calibration plate.

Table 1 provides a more formal definition of the Raw2D and Phys3D spaces.

Space	Description	Origin	Units	Handedness
Raw2D	Raw 2D image space defined by acquired pixels.	Upper-left corner of upper-left pixel in acquired image.	Pixels.	Left-handed. Positive-X extends to the right, positive-Y extends down.
Phys3D	3D physical space. Initially, this is a placeholder space defined by the calibration plate.	Defined by fiducial mark on calibration plate (CalPlate3D space).	Physical units. Initially defined by calibration plate grid spacing and spacing between elevated plate views.	Right handed. X- and Y-axis aligned to calibration grid, Z-axis normal to plate extending away from the camera.

Table 1. Raw2D and Phys3D spaces

The **Cog3DCameraCalibration** class provide functions that map between Raw2D and Phys3D spaces, in either direction. Note that the Raw2D to Phys3D mapping is unusual, in that it maps points in Raw2D space to rays in Phys3D space, while the Phys3D to Raw2D mapping maps points in Phys3D space to points in Raw2D space.

Note All transformations used with 3D calibration are named using the format *SpaceAFromSpaceB*. Such transformations accept values expressed in *SpaceB* and maps the values to *SpaceA*.

Figure 7 shows Raw2D space, Phys3D space, and the Raw2DFromPhys3D transformation that links the spaces.

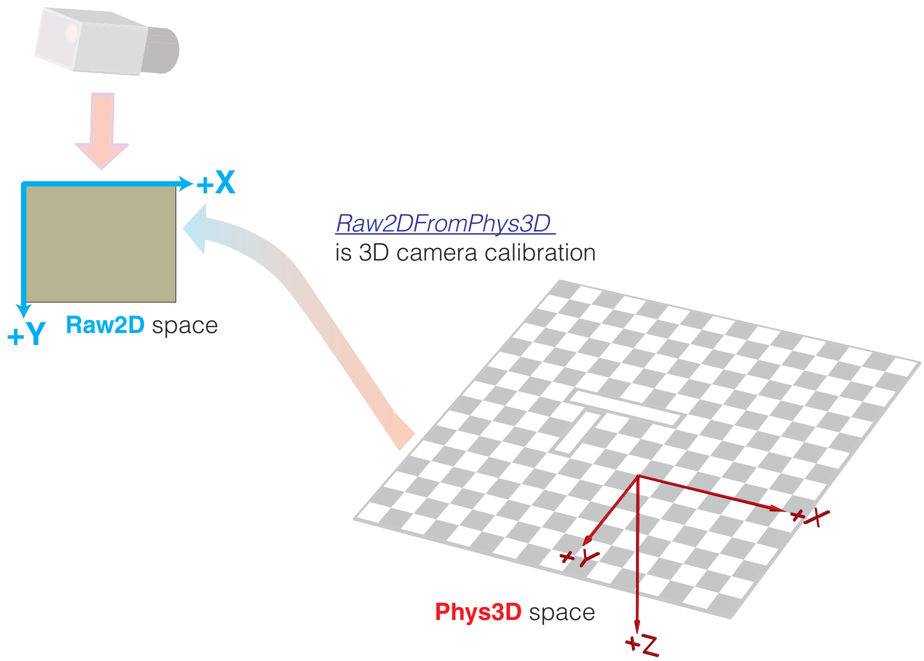


Figure 7. Raw 2D and Physical 3D space

Additional Spaces

All that is required to make basic use of a 3D calibration is the overall 3D calibration and the two spaces listed in the preceding section. The 3D calibration tool also records additional information about the 3D calibration in the form of a pair of additional intermediate coordinate spaces. Table 2 adds these spaces to the two spaces described in Table 1 on page 21.

Space	Description	Origin	Units	Handedness
Raw2D	Raw 2D image space defined by acquired pixels.	Upper-left corner of upper-left pixel in acquired image.	Pixels.	Left-handed. Positive-X extends to the right, positive-Y extends down.
Camera2D	Undistorted 2D space that removes effects of optical distortion and pixel aspect ratio.	0,0,1 in Camera3D space.	N/A	Right-handed if viewing in front of the camera. X- and Y-axes parallel to and in the same direction as Camera3D X- and Y-axes
Camera3D	Idealized 3D space with Z-axis corresponding to optical axis of camera.	Point of optical convergence within lens.	Physical units.	Right -handed. X- and Y-axis roughly parallel to Raw2D X- and Y-axis. Z-axis extends away from front of camera along optical axis.
Phys3D	3D physical space.	Defined by fiducial mark on calibration plate.	Physical units. Initially defined by calibration plate grid spacing and spacing between elevated plate views.	Right handed. X- and Y-axis aligned to calibration grid, Z-axis normal to plate extending away from the camera.

Table 2. Supplemental spaces.

The transformations between the four coordinate spaces listed in Table 2 provide important information about the computed 3D calibration:

- **Cog3DCameraCalibrationIntrinsics::MapPointFromCamera2DToRaw2D**

  The transformation from Camera2D to Raw2D space provides the *intrinsic* part of the overall 3D calibration. The camera intrinsics form a nonlinear transformation that removes the effect of optical distortion, pixel aspect ratio, and any irregularity in the spacing of pixels within the image sensor.

- **Cog3DCameraCalibration::Camera3DFromPhys3D**

  The transformation to Phys3D from Camera3D space provides the *extrinsic* part of the overall 3D calibration. The extrinsic are a rigid 6-degree-of-freedom linear transformation between the Phys3D space and the Camera3D space.

The transformation between Camera3D and Camera2D space is a simple projection. Any 3D point (X,Y,Z) in Camera3D space can be mapped to a 2D point in Camera2D space by dividing its X- and Y-values by its Z-value.

Figure 8 shows all four of the spaces, and their associated transformations, generated during 3D calibration.

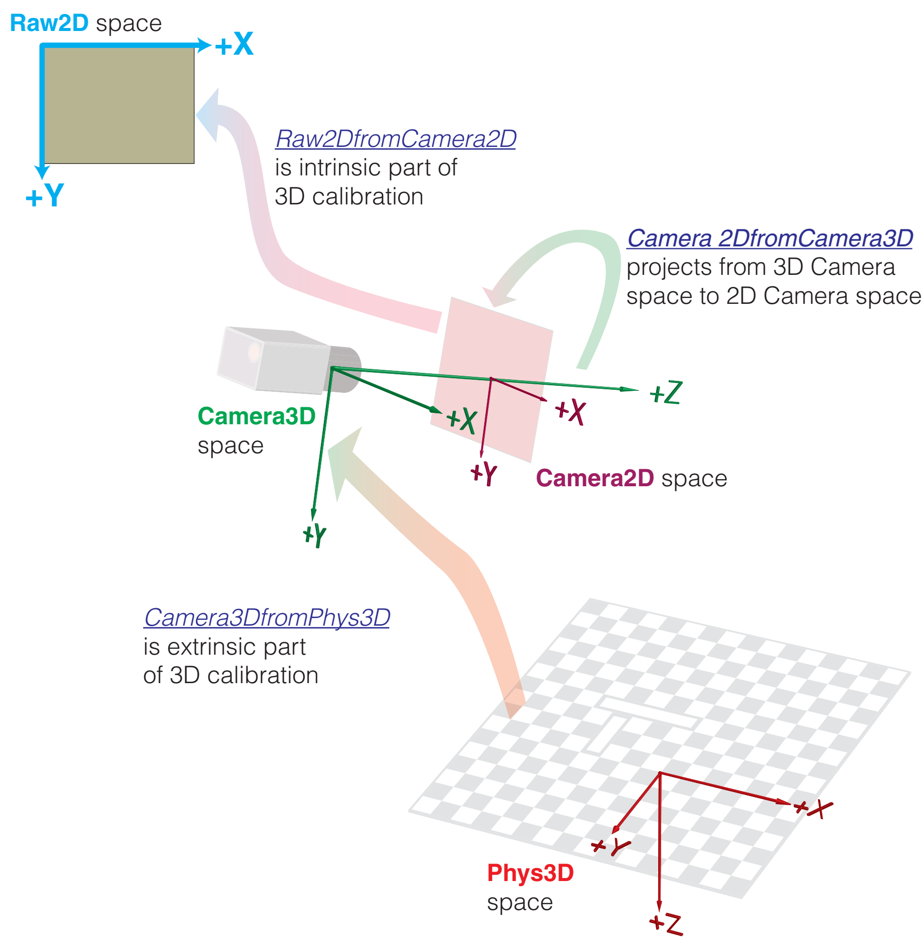


Figure 8. Spaces and transformations computed by 3D calibration

Multi-Camera Calibration

While single-camera 3D calibration is supported, and can be used effectively in some applications, the highest accuracy and greatest application flexibility is provided by using multiple cameras.

Multi-camera 3D calibration is performed in exactly the same way, and produces the same type of results, as single-camera calibration. The only differences are that multiple, simultaneously acquired images (1 per camera) are acquired of each plate pose, as shown in Figure 9, and a separate calibration object is computed for each camera.

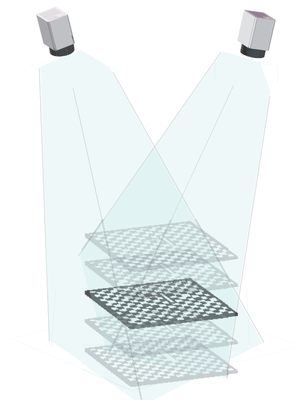


Figure 9. Multi-camera calibration

Note For single-camera calibration, all images must include well-focused views of all calibration plate features. For multi-camera calibration, if some images from some cameras do not include calibration features, it is generally still possible to compute an accurate calibration. For best results, however, every image from every camera should include well-focused views of all calibration plate features.

A multi-camera 3D calibration is computed with a single function call, and this call takes as arguments all of the image data from all of the cameras, properly indexed. Because a single function call has access to the input data from all cameras at the same time, a highly accurate calibration is computed.

The resulting calibration provides a 3D calibration object for each calibrated camera. While each camera has a unique Raw2D, Camera2D, and Camera3D space, they share the same Phys3D space since they were viewing the same 3D physical coordinate space defined by the calibration plate.

Figure 10 shows the spaces associated with a 3D calibration using two cameras.

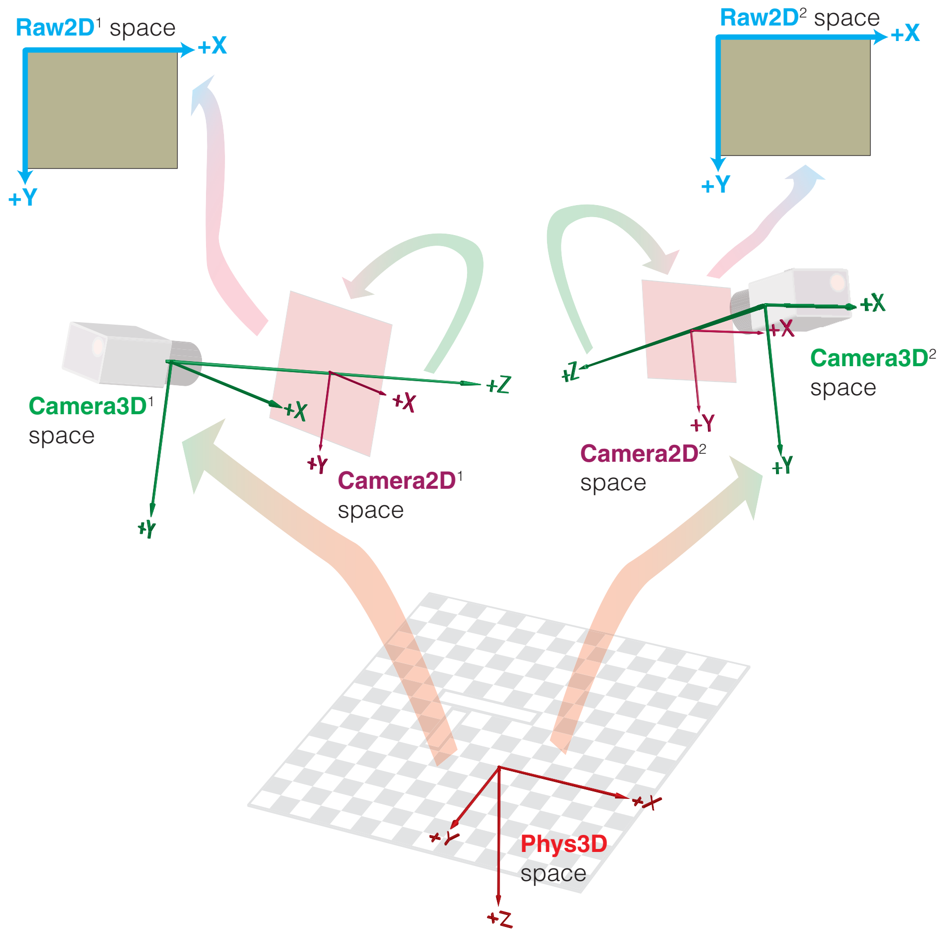


Figure 10. 3D camera calibrations from multiple cameras share common 3D physical space

The design of 3D calibration guarantees that all 3D-calibrated cameras that were calibrated at the same time map Raw2D image points to the same Phys3D space. This capability is the basis of triangulation, which is discussed in the next section.

Triangulation

Because a 3D-calibrated camera can transform a 2D point in Raw2D space to a 3D ray in Phys3D space, two (or more) 3D-calibrated cameras arranged so that they can view the same object from different locations can generate multiple intersecting rays in Phys3D space that all correspond to the same feature of the object. Triangulation is the process of using this property of multiple 3D-calibrated cameras to generate 3D position and pose information from multiple acquired images.

There are several basic requirements that must be met in order to successfully use triangulation to convert 2D locations into a 3D location:

- The image acquired from each of the 3D-calibrated cameras must include the feature that you wish to locate.
- You must be able to use a 2D vision tool to obtain an accurate 2D feature location in the acquired image; the 2D location of the feature must not be affected by 3D effects such as perspective and foreshortening.
- The part cannot have moved between the times the images were acquired. For best results, the images should be acquired simultaneously.
- The 3D-calibrated cameras must have been calibrated together so that their calibrations all refer to the same Phys3D space. In addition, the optical configuration, including focus position and aperture, must not have changed for any camera since the cameras were calibrated.
- The 3D-calibrated cameras must view the part from different directions.

Figure 11 shows two 3D-calibrated cameras acquiring images of the same part at the same time.

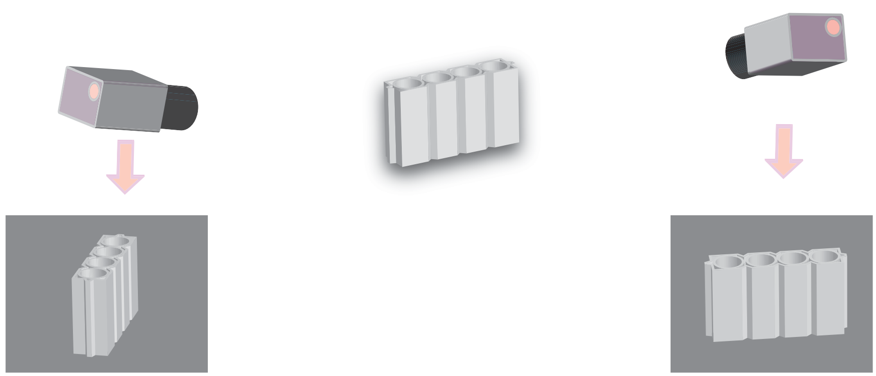


Figure 11. Simultaneous multi-camera acquisition

Once the images have been acquired, your application must determine the location of a given feature on the object in both images. In the example shown in Figure 11, the application could use a corner on the lower edge of the end of the part, as shown in Figure 12.

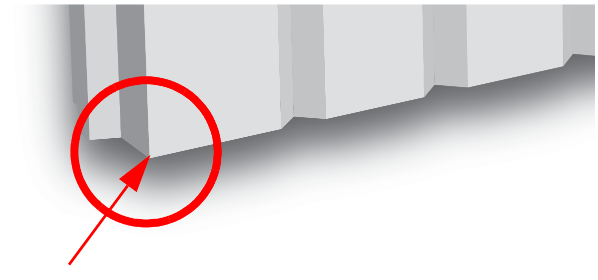


Figure 12. Part feature

Using 2D vision tools, the application first locates the same part feature (the corner) in Raw2D space in both acquired images. The application then calls the triangulation tool's **Cog3DTriangulator.Execute()** function providing the following inputs:

- The two feature locations in Raw2D space
- The two **Cog3DCameraCalibration** objects produced during 3D calibration

The function returns the 3D location of the feature in Phys3D space. Figure 13 shows the triangulation process.

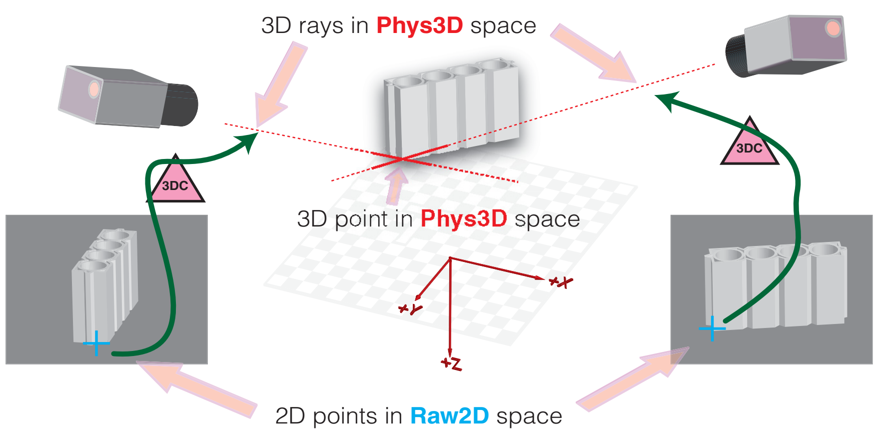


Figure 13. Triangulating a 3D point from two 2D points from two 3D-calibrated cameras

Estimating 3D Object Pose

While triangulation can provide 3D locations of points, most 3D vision applications are interested in the 3D pose of an object. If your application can meet the following basic requirements, you can use the 3D API to determine the 3D pose of an object based on the location of 2D features in acquired images.

- Your part has features that can be reliably and accurately found in 2D images from one or more 3D-calibrated cameras.
- A sufficient number of features can be located to allow pose estimation.
- You can provide a 3D Model comprised of a set of 3D model features. The model features must be expressed in a single 3D coordinate space named Model3D space.

3D Model

Consider the part shown in Figure 12 on page 29. You could supply a 3D Model of the part by giving the 3D locations of the eight outermost corners of the part, as shown in Figure 14.

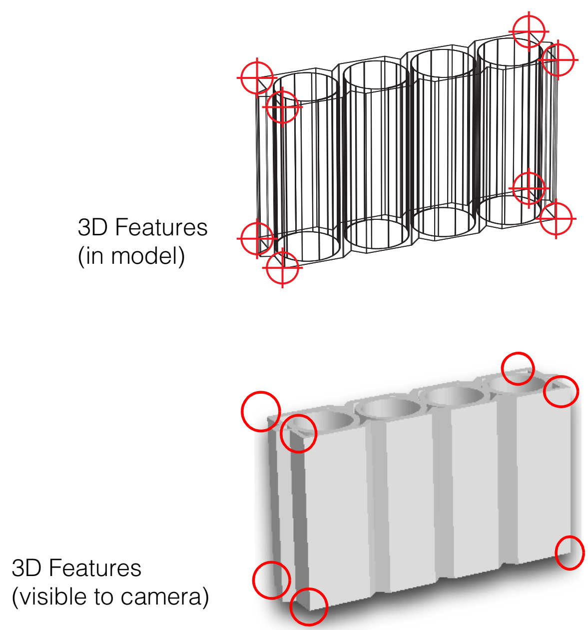


Figure 14. 3D Features

Even though not all eight of the 3D model features that define the 3D model are visible to the camera, the first requirement is still met because seven features are visible, and the seven visible features include three non-collinear points, which is sufficient for 3D pose estimation.

Note As shown in Figure 15, the 3D features that make up a 3D model are maintained as an indexed list. In this example, the model contains eight 3D points.

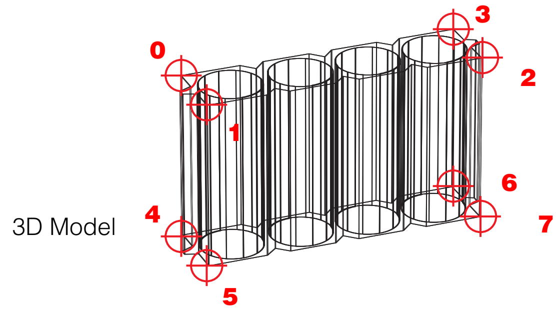


Figure 15. 3D Model points are indexed

Direct 3D Pose Estimation

Consider the simple example of a part with a 3D model consisting of three points. If you can correspond the locations of those three points with the 2D locations of those features in an image acquired from a 3D-calibrated camera, the VisionPro 3D-Locate API can generate an accurate estimate of the pose of the model in 3D physical space using direct pose estimation.

Direct pose estimation is based on the fact that given a point in an image from a 3D-calibrated camera, VisionPro can compute the 3D ray in 3D physical space that corresponds to that point, as shown in Figure 1 on page 14.

To perform direct pose estimation, the 3D-Locate API computes the 3D rays for each of the supplied 2D points, then determines the 3D model pose that best fits the specified 3D points in the model to the computed 3D rays. This process is shown in Figure 16.

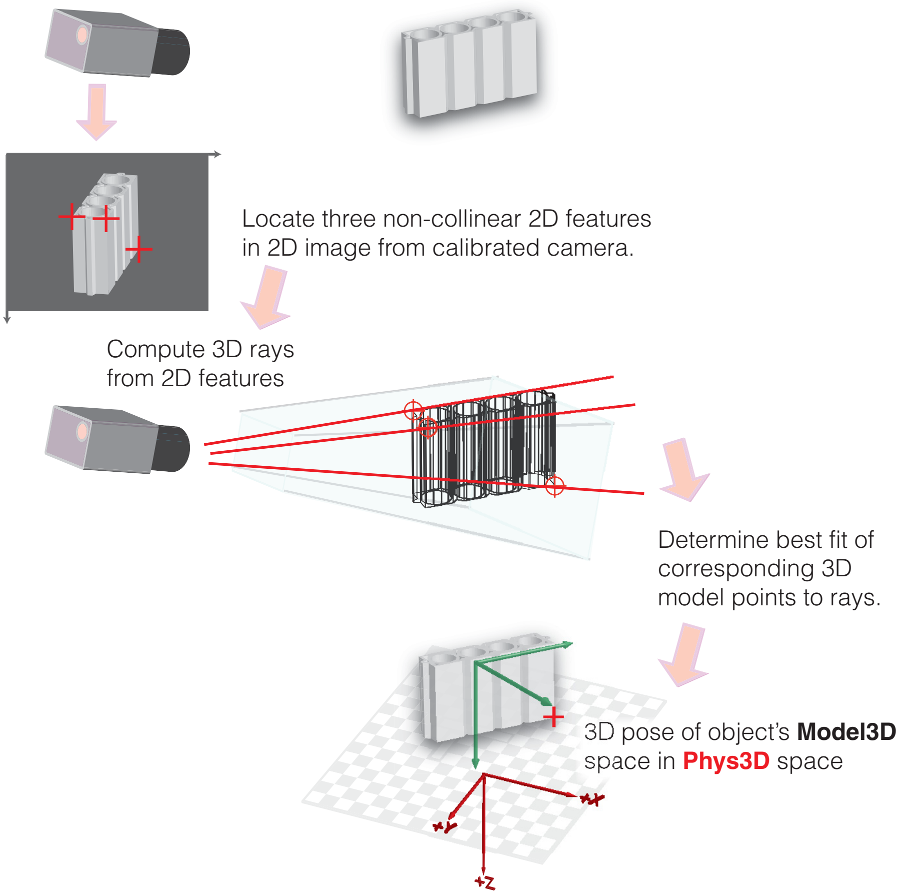


Figure 16. Direct 3D pose estimation from 2D-3D correspondence

Feature Correspondence

The VisionPro 3D-Locate API uses the **Cog3DCrsp2D3D** class to hold information about a single 2D-3D *feature correspondence* (the correspondence between one 2D feature in one image from one camera and a single 3D feature in a 3D model). The classes and methods that implement 3D pose estimation, 3D model generation, and part correspondence take lists of **Cog3DCrsp2D3D** objects as input. It is the job of your VisionPro 3D application to create, initialize, and manage these **Cog3DCrsp2D3D** objects.

Note The **Cog3DCrsp2D3D** class is described in detail in Chapter 4, *Locating Objects in 3D* on page 81 as well as the VisionPro Programming Reference in the VisionPro online documentation.

Each feature correspondence includes the following basic information:

- The 2D position of a feature in an image acquired by a 3D-calibrated camera
- The index of the 3D feature in a 3D model that corresponds to the 2D position of the feature in the image.
- The index of the camera used to acquire this image, if you are acquiring images simultaneously from multiple cameras that are 3D-calibrated to the same 3D physical space.
- The index of the part instance in this image, if multiple instances of the part are present.
- Detailed information about the 3D feature, including its type (point, line segment, circle, or cylinder) and subfeatures. The use of non-point 3D features is introduced in the section *Non-Point 3D Features* on page 37.

Using Feature Correspondences for Pose Estimation

As described in the previous section, all that is required for direct pose estimation are the correspondences between 3D model features and 2D features as visible in an image from a 3D-calibrated camera.

VisionPro 3D-Locate uses feature correspondences as a simple way of encoding and managing this information. They are also used as input by the direct pose estimator.

Figure 17 shows how you would use two feature correspondences to represent how the 2D features in the image shown in Figure 16 on page 32 correspond to the 3D features in the 3D model.

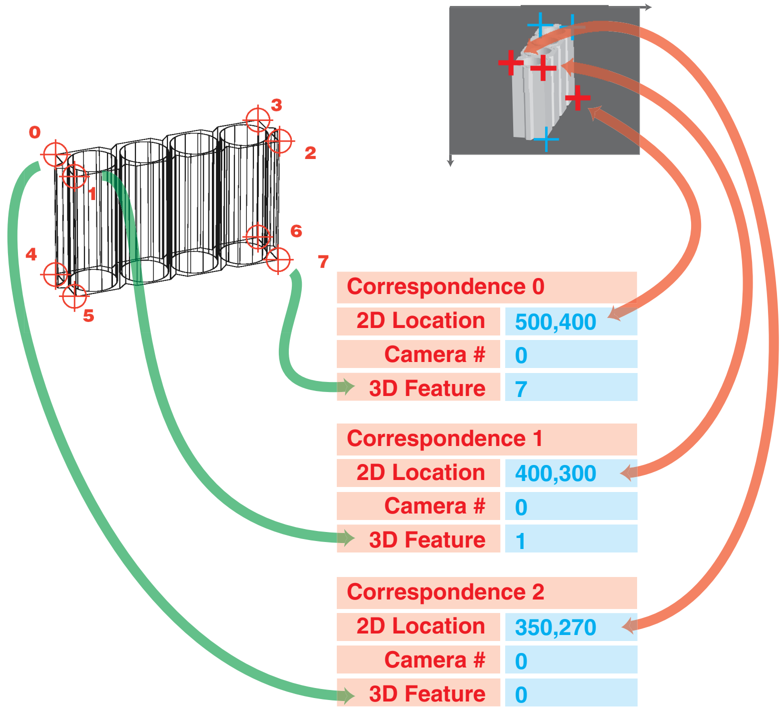


Figure 17. Feature correspondences for single-camera pose estimation using three 3D Model feature points.

Note The feature correspondences shown in Figure 17 do not include the part instance index or feature type fields.

When you use feature correspondences to perform direct pose estimation, keep in mind that:

- You must create a feature correspondence for each 2D feature for which a corresponding 3D feature exists.
- Only one correspondence can exist for a given 2D feature. A given 2D feature can only correspond to one 3D feature.
- A given 3D feature will typically be associated with multiple feature correspondences, one for each image in which the feature is visible.

Multi-Camera Direct 3D Pose Estimation

The example shown in Figure 16 on page 32. uses a single camera. You can get better accuracy and robustness, particularly in cases where not all features may be seen in all parts, by using multiple cameras. You can use any number of cameras for pose estimation, as long as

- The cameras are all 3D-calibrated to the same 3D space as described in the section *Multi-Camera Calibration* on page 25.
- All of the feature correspondences describe 2D features from images of the same part or parts acquired at the same time. You cannot use 2D features from images where the part moved between the time that the images were acquired.

Figure 18 shows the use of multiple cameras for direct pose estimation.

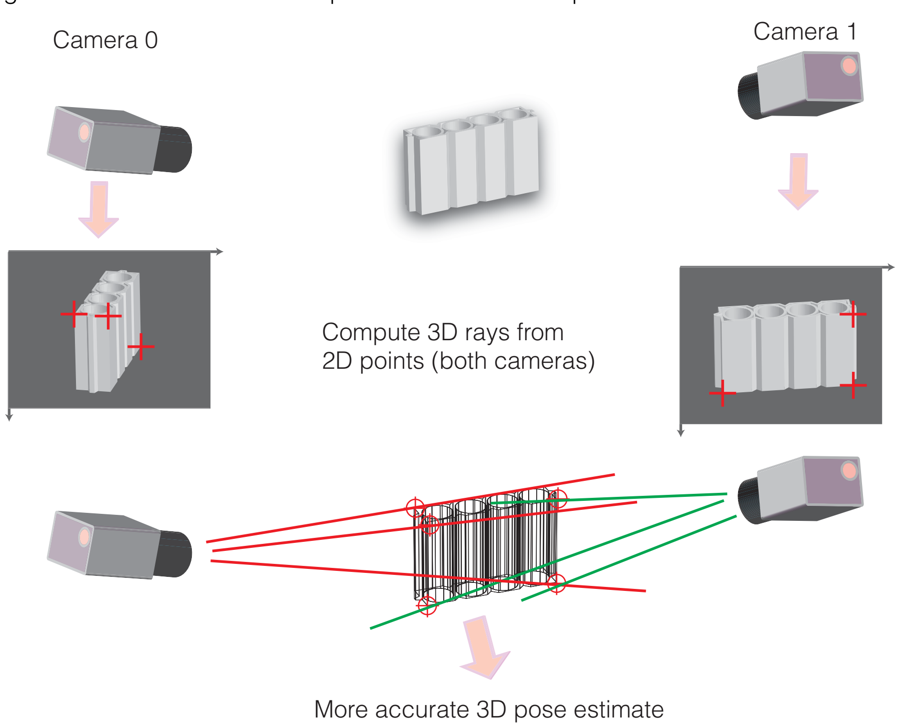


Figure 18. Using multiple cameras for direct 3D pose estimation.

By adding feature correspondence information from additional cameras, your application can obtain more accurate pose estimates, and because multiple cameras can provide for multiple views of a given part feature, your application will be more robust in situations where not all part features are visible to all cameras at all times.

If you are using multiple cameras, you use additional feature correspondence pairs to describe the additional correspondences, taking care to set the camera index correctly.

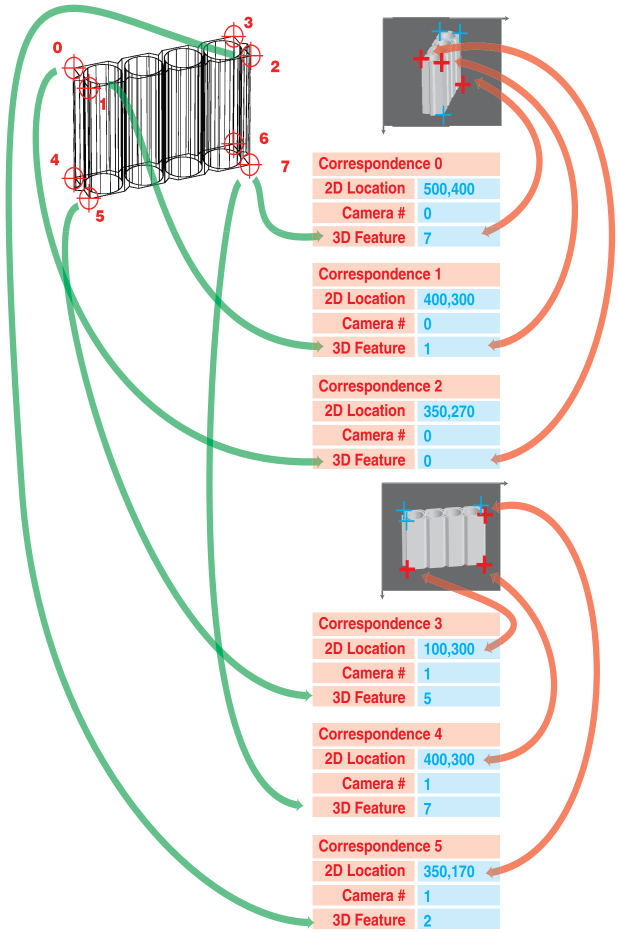


Figure 19. Feature correspondences for multi-camera pose estimation

Multiple Parts

The VisionPro 3D-Locate API supports 3D pose estimation for multiple parts simultaneously, as long as each feature correspondence pair correctly identifies the part instance of the correspondence. For most real-world applications, which use 2D vision tools to locate part features in images, it can be difficult or impossible to guarantee that this part instance correspondence is correct, as shown in Figure 20.

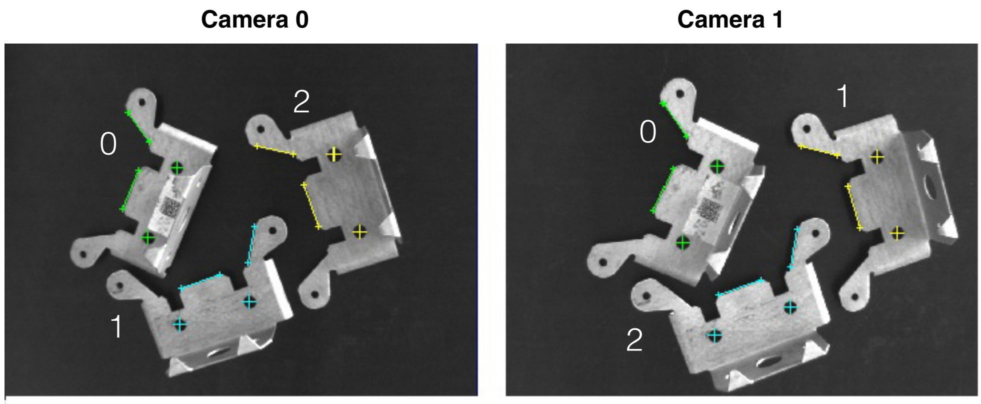


Figure 20. Part correspondence problem

VisionPro 3D-Locate includes an automatic part corresponder that can detect and correct the part correspondence information in feature correspondences from different images, as long as you have a 3D model of your part *and* the 2D features are correctly corresponded to the 3D model feature.

For detailed information on the part corresponder, see Chapter 4, *Locating Objects in 3D* on page 81.

Non-Point 3D Features

All of the examples described in this section use 2D and 3D points as features. VisionPro 3D-Locate also allows you to work with complex 3D feature types, including lines, line segments, circles, and cylinders. Using these complex feature types allows you to make use of the edges, holes, and cylinders in your part, simplifying the task of 2D feature location, and greatly improving the accuracy of 3D pose estimation.

The use of complex feature types is described in detail in Chapter 4, *Locating Objects in 3D* on page 81.

Using Feature Correspondences to Generate 3D Models

If, at setup time, you have a collection of feature correspondences for multiple features of your part, but you do not have a 3D model that defines the 3D positions of those features relative to each other (that is, in a single Model3D coordinate space), you can use the VisionPro 3D-Locate API to automatically generate the 3D model for you.

Note You must have feature correspondence pairs from multiple cameras to use the model generation function.

3D model generation uses triangulation to compute the 3D position of a 3D model feature based on multiple feature correspondences that give the 2D location of the feature in multiple images acquired simultaneously from 3D-calibrated cameras.

Robot (Hand-Eye) Calibration

Robot calibration is a specialized type of calibration that is suited for the specific application where a single machine vision camera is mounted on the end-effector of a robot.

Hand-eye calibration makes use of two additional 3D coordinate spaces:

- *RobotBase3D space* is a 3D physical space defined by the robot manufacturer or integrator. It is typically associated with the robot base (the part of the robot that is rigidly fixed to the physical world).
- *Hand3D space* is a 3D physical space associated with the robot's end efffector. The robot controller reports the location of the end effector by specifying the pose of Hand3D space in RobotBase3D space.

Hand-eye calibration computes the rigid 3D transformation that maps 3D points from Hand3D space to the Camera3D space of the effector-mounted camera. It is this mapping that permits your application to ultimately link feature locations in an image acquired from the end-effector-mounted camera with 3D locations in the robot's RobotBase3D space.

Note Hand-eye calibration can be done for systems using a single camera or multiple cameras. Also, hand-eye calibration can also calibrate robots in which the camera is stationary and the robot moves a stage carrying the calibration plate. In this case, the transformation between RobotBase3D space and Camera3D space is calibrated.

Calibration Phase

To perform hand-eye Calibration, place a standard checkerboard calibration plate at a fixed location. Then move the robot end-efffector to a series of positions (“stations”) from which it can view the calibration plate. At each station, acquire an image of the plate and record the pose of the end effector’s Hand3D space in RobotBase3D space (this information is typically provided by the robot controller software).

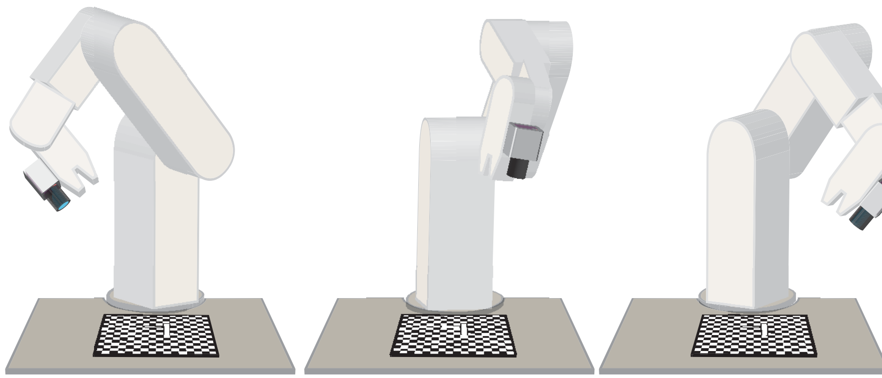


Figure 21. Hand-eye calibration

To perform the hand-eye calibration, call the calibration function with two items of data from each station:

- The correspondence pair list of the plate vertex locations generated by the checkerboard feature extractor, as described in the section *How Does 3D Calibration Work?* on page 15.
- A rigid 3D transformation giving the pose of Hand3D space in RobotBase3D space

Figure 22 shows the inputs from a single station.

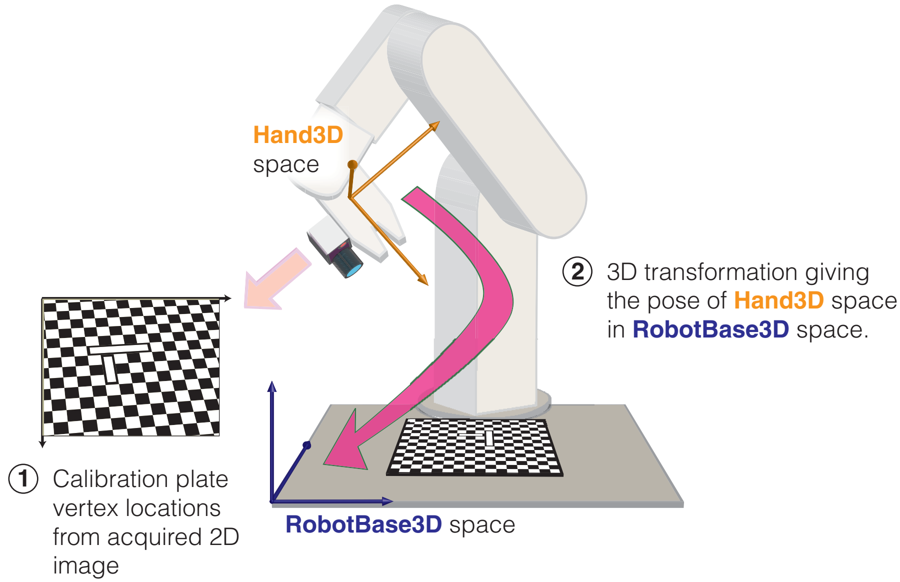


Figure 22. Per-station inputs for hand-eye calibration

Calibration Outputs

A successful hand-eye calibration produces a rigid 3D transformation that transforms 3D points from Hand3D space to Camera3D space, as shown in Figure 23.

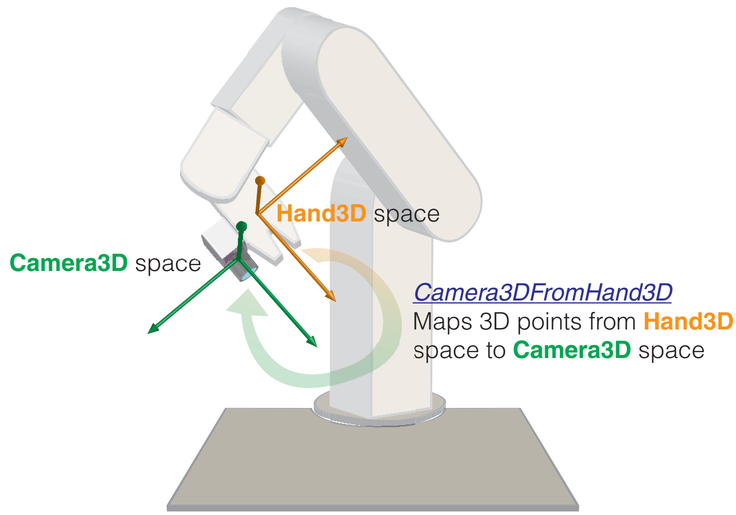
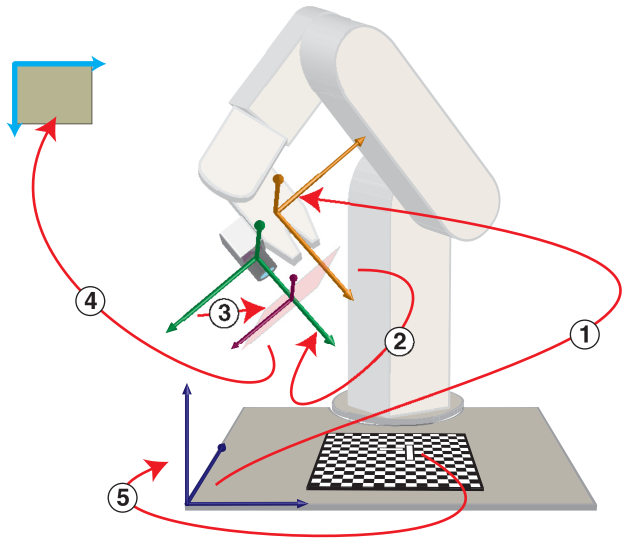


Figure 23. Hand-eye calibration outputs

Hand-eye calibration takes intrinsic parts of camera calibrations (obtained from 3D camera calibration) as inputs. The same data set acquired for hand-eye calibration may be used for camera calibration first to obtain the camera intrinsics.

Most single-camera robot vision applications that use hand-eye calibration will ultimately use both the hand-eye calibration and the intrinsic part of the 3D camera calibration, as shown in Figure 24.



1. **Hand3D** space from **RobotBase3D** space (supplied by robot)
2. **Camera3D** space from **Hand3D** space (result of hand-eye calibration)
3. **Camera2D** space from **Camera3D** space (projection)
4. **Raw2D** space from **Camera2D** space (intrinsic part of original camera calibration)
5. **RobotBase3D** space from **CalPlate3D** space (result of hand-eye calibration)

Figure 24. Spaces and transformations associated with hand-eye calibration

The **Cog3DHandEyeCalibrationResult** object contains both the hand-eye calibration and the camera intrinsics.

3D Shapes, Graphics and Transforms

2

This chapter describes the basic VisionPro 3D-Locate building blocks that provide the mathematical foundation for your 3D applications. The framework includes the following items:

- 3D shapes that provide a rich set of classes for representing and manipulating geometric objects in three dimensions.
- 3D transformations that allow you to describe the pose of 3D objects in 3D space, and to map points and shapes between different 3D coordinate spaces. The 3D transformations include a rich set of representations for 3D rotations, which allow you to characterize the rotation of objects and coordinate systems in three dimensions.
- 3D shape projection tools that allow you to create a 2D representation of a 3D shape using a 3D camera calibration. Such a 2D projection can be used to display a graphical representation of a 3D shape using a 2D display device.

This chapter contains the following sections:

- *Some Useful Definitions* on page 46, provide an overview of the chapter and define some terms that you will encounter as you read.
- *3D Shapes* on page 47 describes the 3D shape classes and interfaces, and also describes the shape projection functions.
- *3D Transformations* on page 51 provides information on the classes that implement 3D rigid transformations.

Some Useful Definitions

This section defines some terms and concepts used in this chapter.

3D Pose The position and orientation of a 3D coordinate system within another 3D coordinate system. A pose comprises 6 degrees of freedom: X-translation, Y-translation, Z-translation, X-rotation, Y-rotation, and Z-rotation.

3D Position The location of a 3D point within a 3D coordinate system. A 3D position is represented by an x-value, y-value, and z-value.

Camera3D Space A right-handed 3D coordinate space with its origin at the camera's optical convergence point, X- and Y-axes that are approximately parallel to and oriented in the same direction as the Raw2D coordinate system X- and Y-axes, and a Z-axis that extends along the optical axis away from the camera.

Space The plane at Z=1 of Camera3D Space. When space is viewed from the camera (Z < 1 and in the direction of the Camera3D positive Z axis) then space appears as a left-handed 2D coordinate system. When space is viewed in the direction of the Camera3D negative Z axis from a point in front of the camera (where Z > 1), then Space appears as a right-handed 2D coordinate system.

Hand3D Space A right-handed 3D coordinate space defined by the end-effector on a robot. The position of the robot hand is reported by the robot controller as the pose of Hand3D space in RobotBase3D space. (Some robot manufacturers and integrators refer to this as *tool space*.)

Model3D Space A 3D coordinate space implicitly defined by a collection of 3D points that describe a 3D object (model).

Phys3D Space A right-handed 3D coordinate space initially defined by the fiducial mark on the calibration plate used to perform 3D calibration. This space can be defined by any coordinate frame in physical space.

RobotBase3D Space A right-handed 3D coordinate space defined by the robot manufacturer or integrator. It is typically associated with the robot base (the part of the robot that is rigidly fixed to the physical world).

3D Shapes

VisionPro 3D-Locate includes a 3D shape framework as part of the mathematical foundation for 3D vision applications. The key features of the 3D shape framework are:

- Concrete implementations of the following 3D shapes:

  Cog3DAlignedBbox
  Cog3DBox
  Cog3DCircle
  Cog3DCylinder
  Cog3DLine
  Cog3DLineSeg
  Cog3DPlane
  Cog3DRay
  Cog3DRectangle
  Cog3DSphere

- Basic geometric functions such as area, volume, distance, intersection, parallelism, anti-parallelism, and nearest point for all the concrete shapes.
- Polymorphism of the concrete shapes that allows you to work with concrete shapes through an abstract base class reference. This allows you to create code to obtain the volume of a set of shapes without needing to check the type of each shape.

3D Shape Class Architecture

All concrete 3D shape classes inherit from the **Cog3DShapeBase** interface, and implement other interfaces that define common behavior for different shape types:

- **ICog3DVertex** provides functionality for shapes that can be described as a set of vertices.
- **ICog3DCurve** provides functionality for shapes that can be described by 3D line segments, arcs, and other 1D shapes.
- **ICog3DSurface** provides functionality for shapes that can be described by surfaces.
- **ICog3DVolume** provides functionality for shapes that can be described as a volume or collection of volumes.

Figure 25 shows the inheritance and implementation for a **Cog3DAlignedBox**: .

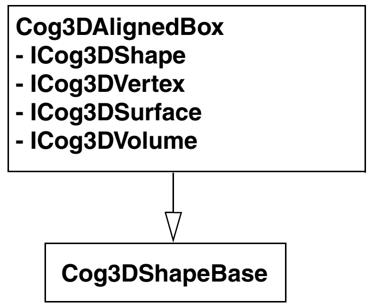


Figure 25. Cog3DAlignedBox class inheritance

3D Shape State Type

VisionPro 3D-Locate shape types that implement **ICog3DShape** must implement the **ShapeState** property. The 3D shape framework defines four state types that correspond to the four shape type interfaces:

Shape type class	State type enumeration
ICog3DVertex	*Cog3DShapeStateConstants.Vertex*
ICog3DSurface	*Cog3DShapeStateConstants.Surface*
ICog3DCurve	*Cog3DShapeStateConstants.Curve*
ICog3DVolume	*Cog3DShapeStateConstants.Volume*

A shape's shape type controls how the shape's geometry is interpreted by various memebers of **Cog3DShape**. Figure 26 shows how the state type effects how the distance to shape measurement is made between a point an a **Cog3DBox**.

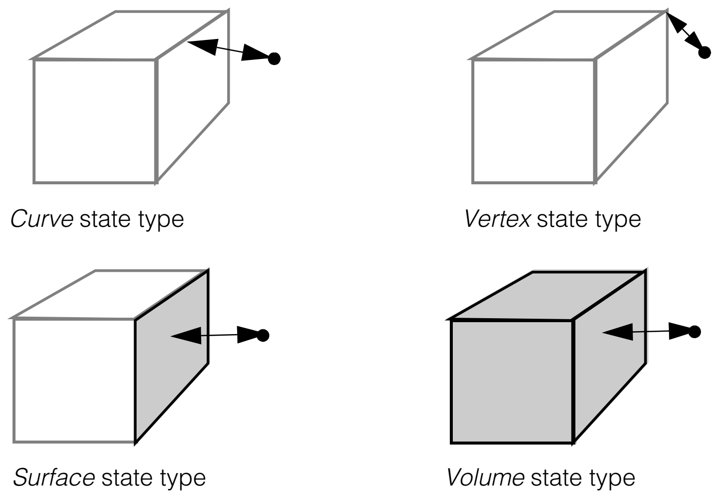


Figure 26. Shape State Type

3D Shape Geometric Operations

VisionPro 3D-Locate supports the **Cog3DShapeGeometricOperations** class, which includes methods performing shape intersection, projection to plane, parallelism/perpendicularity checking, and others.

Projecting 3D Shapes for Display

You can project 3D shapes into a 2D image if the image was acquired from a 3D calibrated camera.

Note You must supply the 3D camera calibration object associated with the 2D image space into which you wish to project a 3D shape. For more information on 3D camera calibration, see the chapter *3D Calibration Tools* on page 55.

The 3D shape projection function, **Cog3DShapeProjector.Execute()** generates standard 2D graphics that correspond to the projected 3D shape and appends them to a **CogGraphicCollection** that you can display. The **Cog3DShapeProjector** class supports properties to control the graphical appearance of the shapes.

Projection Shape Representation

When projecting a 3D shape you can choose to project the vertices of the shape or the shape's curves (edges). Shape projection does not support the projection of surfaces or volumes, and it does not support hidden line removal.

3D Transformations

The following table summarizes the .NET classes that support 3D rigid transformations, rotation, and poses:

Functionality	.NET Classes
3D Rotation	**Cog3DTransformRotation**
3D Rotation – Euler angles	**Cog3DEulerZYX, Cog3DEulerXYZ, Cog3DEulerZYXMovingAxes, Cog3DEulerXYZMovingAxes**
3D Rotation – quaternions	**Cog3DQuaternion**
3D Rotation – axis angle	**Cog3DAxisAngle**
3D Rotation – matrix	**Cog3DMatrix3x3**
3D Translation	**Cog3DVect3**
3D Point	**Cog3DVect3**
3D Rigid Transform	**Cog3DTransformRigid**
3D Linear Transform	**Cog3DTransformLinear**
3D Composed Transform	**Cog3DTransformComposed**
3D Pose Estimation	**Cog3DPoseEstimatorUsing2DPoints** **Cog3DPoseEstimatorUsing3DPoints** **Cog3DPoseEstimatorUsingCrsp2D3Ds**

3D Rigid Transforms

The VisionPro 3D-Locate vision framework provides the **Cog3DTransformRigid** class that implement a 3D rigid body transform or *3D rigid transform*. A 3D rigid transform provides a mapping between two 3D Cartesian coordinate spaces. The mapping consists of a 3D rotation and a 3D translation. There is no scaling, aspect, or shearing in a rigid transform.

Note The 3D rigid transformation class is intended to let you map points, vectors, poses, and shapes between 3D coordinate spaces. These classes do not provide non-rigid, non-linear, or 2D-to-3D transformations.

3D Rotation

3D rotations are encapsulated in the **Cog3DTransformRotation** class, which allows construction from and conversion to many different representations of 3D rotations: Euler angles, quaternions, 3x3 matrices, and others. Which representation you use depends on the requirements of your overall application.

All these representations are described here:

http://en.wikipedia.org/wiki/Rotation_representation_(mathematics)

3D Fitting

VisionPro 3D-Locate provides the following classes that let you fit 3D shapes to collections of 2D or 3D points.

Class	Description
Cog3DCircleFitterUsing2DPoints	This class provides 3D circle fitting from 2D points. A 3D circle is fitted from one or multiple sets of 2D image points. The sets of image points can come from different cameras or from a single camera providing multiple views of the circle. The 3D circle fitter computes the pose of the circle which minimizes the sum squared image error with respect to the given 2D image points from calibrated camera(s).
Cog3DCircleFitterUsing3DPoints	This class provides 3D circle fitting from 3D points. The 3D circle fitter computes the pose of a 3D circle based on the specified 3D input points. The fitting technique is controlled by the **Cog3DRobustFitParameters**. See the Remarks section of the **Cog3DRobustFitTechniqueConstants** enum and the **Cog3DRobustFitParameters** class for details.
Cog3DCylinderFitterUsing2DPoints	This class provides 3D cylinder fitting from 2D points. A 3D cylinder is fitted from multiple sets of 2D image points. The sets of image points can come from different cameras or from a single camera. The 3D cylinder fitter computes the pose of the cylinder which minimizes the sum squared image error with respect to the given 2D image points from calibrated camera(s).

Class	Description
Cog3DLineFitterUsing2DPoints	This class provides 3D line fitting from 2D points. A 3D line is fitted from multiple sets of 2D image points. The sets of image points can come from different cameras or from a single camera providing multiple views of the line. The 3D line fitter computes the pose of the line which minimizes the sum squared image error with respect to the given 2D image points from calibrated camera(s).
Cog3DLineFitterUsing3DPoints	This class provides 3D line fitting from 3D points. The 3D line fitter computes the pose of a 3D line based on the specified 3D input points. The fitting technique is controlled by the **Cog3DRobustFitParameters**. See the Remarks section of the **Cog3DRobustFitParameters** enum and the **Cog3DRobustFitParameters** class for details.
Cog3DPlaneFitterUsing3DPoints	This class provides 3D plane fitting from 3D points. The 3D plane fitter computes the pose of a 3D plane based on the specified 3D input points. The fitting technique is controlled by the **Cog3DRobustFitParameters**. See the Remarks section of the **Cog3DRobustFitParameters** enum and the **Cog3DRobustFitParameters** class for details.

The sample code installed in %VPRO_ROOT%*samples3D\Programming\Runtime\Fitting* contains a Visual Studio 2010 solution with several shape fitting examples. Copy the contents to a folder where you have write permission before you execute it.

3D Calibration Tools

3

This chapter describes 3D calibration, a process that establishes a mathematical relationship between the 2D coordinate system associated with the pixels in an acquired image and a 3D coordinate system associated with the physical world in front of the camera.

This chapter contains the following sections:

Some Useful Definitions

This section defines some terms and concepts used in this chapter.

3D Pose The position and orientation of a 3D coordinate system within another 3D coordinate system. A pose comprises 6 degrees of freedom: X-translation, Y-translation, Z-translation, X-rotation, Y-rotation, and Z-rotation.

3D Position The location of a 3D point within a 3D coordinate system. A 3D position is represented by an x-value, y-value, and z-value.

3D-Calibrated Camera A camera for which a 3D calibration (both extrinsic and intrinsic) has been computed.

Raw2D Space Left-handed 2D coordinate space based on the pixels in an acquired image.

Camera3D Space A right-handed 3D coordinate space with its origin at the camera's optical convergence point, X- and Y-axes that are approximately parallel to and oriented in the same direction as the Raw2D coordinate system X- and Y-axes, and a Z-axis that extends along the optical axis away from the camera.

Camera2D Space The plane at Z=1 of Camera3D Space. When Camera2D space is viewed from the camera (Z < 1 and in the direction of the Camera3D positive Z axis) then Camera2D space appears as a left-handed 2D coordinate system. When Camera2D space is viewed in the direction of the Camera3D negative Z axis from a point in front of the camera (where Z > 1), then Camera2D Space appears as a right-handed 2D coordinate system.

Phys3D Space A right-handed 3D coordinate space initially defined by the fiducial mark on the calibration plate specified as having an origin-defining pose-type used to perform 3D calibration. This space can be defined by any coordinate frame in physical space.

Hand3D Space A right-handed 3D coordinate space defined by the end-effector on a robot. The position of the robot hand is reported by the robot controller as the pose of Hand3D space in RobotBase3D space. (Some robot manufacturers and integrators refer to this as *tool space*.)

RobotBase3D Space A right-handed 3D coordinate space defined by the robot manufacturer or integrator. It is typically associated with the robot base (the part of the robot that is rigidly fixed to the physical world).

Checkerboard Feature Extractor Software that locates all of the grid vertices in an image of a Cognex checkerboard calibration plate, along with the fiducial features that define the plate origin. The feature extractor constructs a list of correspondence pairs which associate the location in the image of each feature with its physical position, based on the physical grid pitch value that you supply.

Correspondence Pair The physical coordinates and image coordinates of a given calibration plate vertex.

Viewset A set of *n* images acquired simultaneously from *n* cameras viewing the same scene from different locations. Multiple viewsets of a calibration plate are used to perform 3D camera calibration.

3D Calibration Basics

3D camera calibration establishes an accurate correspondence between 2D locations in an image from a 3D calibrated camera and a 3D location in physical space in front of the camera. This correspondence takes the form of a linked series of transformations between a pair of 2D coordinate spaces and a second pair of 3D coordinate spaces.

Note For more detailed information, see the section *3D Calibration Coordinate Spaces* on page 20.

Figure 27 shows all four of the spaces, and their associated transformations, generated during 3D camera calibration.

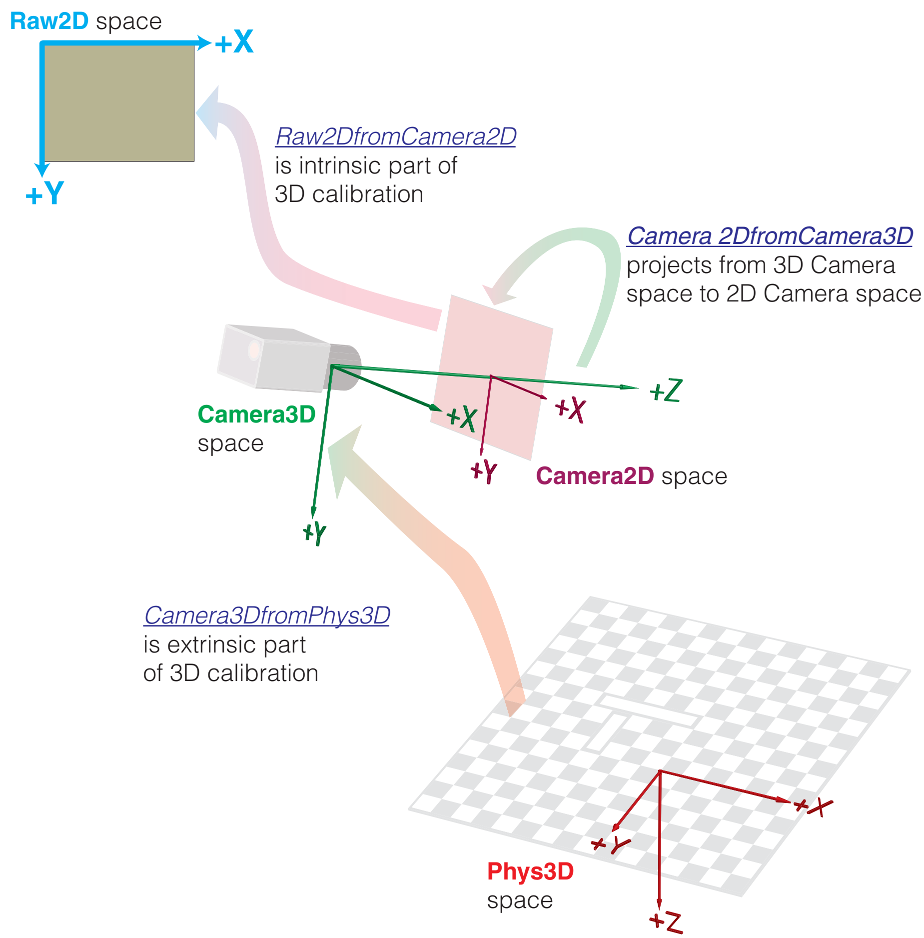


Figure 27. Spaces and transformations computed by 3D camera calibration

Table 3 defines and describes the spaces shown in Figure 27.

Space	Description	Origin	Units	Handedness
Raw2D	Raw 2D image space defined by acquired pixels.	Upper-left corner of upper-left pixel in acquired image.	Pixels.	Left-handed. Positive-X extends to the right, positive-Y extends down.
Camera2D	Undistorted 2D space that removes effects of optical distortion and pixel aspect ratio.	0,0,1 in Camera3D space.	N/A	Right-handed if viewing in front of the camera. X- and Y-axes parallel to and in the same direction as Camera3D X- and Y-axes
Camera3D	Idealized 3D space with Z-axis corresponding to optical axis of camera.	Point of optical convergence within lens.	Physical units.	Right -handed. X- and Y-axis roughly parallel to Raw2D X- and Y-axis. Z-axis extends away from front of camera along optical axis.
Phys3D	3D physical space.	Defined by fiducial mark on calibration plate.	Physical units. Initially defined by calibration plate grid spacing and spacing between elevated plate views.	Right handed. X- and Y-axis aligned to calibration grid, Z-axis normal to plate extending away from the camera.

Table 3. 3D calibration coordinate spaces

It is the first and last spaces shown in Table 3 that enable 3D vision: the mapping of points from the Raw2D space of an acquired image to rays in Phys3D space, and points in Phys3D space to points in Raw2D space.

3D Camera Calibration

VisionPro 3D-Locate allows you to generate a 3D calibration for a single camera or for all the cameras in your production environment simultaneously.

The *image-based* method of calibration allows you to pass all the acquired viewsets of the calibration plate directly to the **Cog3DCameraCalibrator.Execute()** method, along with information about the heights and pose types of each viewset.

VisionPro 3D-Locate also supports a *feature-based* method where you use *checkerboard feature extractor* classes explicitly to locate all the grid vertices in the viewsets of the plate along with the fiducial features the define the plate origin. The feature extractor constructs a list of correspondence pairs which associate the location in the image of each feature with its physical position, based on the physical grid pitch value that you supply. You then pass the correspondence pairs instead of image sets to the **Cog3DCameraCalibrator.Execute()** method.

Note The calibration sample code installed in *%VPRO_ROOT%\samples3D\Programming\Setup\Calibration* includes examples of calibration using both the images and features. Copy the contents to a folder where you have write permission before you execute it.

Cognex recommends that camera calibration be performed with a Cognex calibration plate. If you are performing camera calibration using your own custom calibration plate, you must use feature-based calibration, which is illustrated in the sample code for calibration.

Note Failure to use a precisely manufactured calibration plate will result in poor calibration results. In particular, do not attempt to perform calibration using a calibration plate that you print using a laser or inkjet printer.

Perform the following steps to perform camera calibration:

1. Position, focus, and securely mount the cameras required for your application.

   See the topic *Camera Positioning* on page 62 for more information.

2. Acquire a series of viewsets of the calibration plate, where the images in each viewset have been acquired simultaneously. A viewset contains images from the cameras of the calibration plate in a given pose. You must acquire a minimum of four image sets.

   See the topic *Acquiring the Viewsets* on page 63 for more information.

3. For each image in each viewset, extract the correspondence pairs of 2D image and 3D physical locations of each plate vertex.

4. Pass all of the extracted correspondence pairs, along with information about the plate pose type for each viewset, to the calibration function. You do not need to specify the pose itself, only the type.
5. Measure the accuracy of the computed calibration and, optionally, establish a camera calibration validation baseline.

   See the topic *Assessing the Calibration Quality* on page 69 for more information.

Camera Positioning

Before attempting to calibrate your cameras, define a 3D working volume sufficiently large enough to contain the objects that your application is locating or measuring, allowing for the expected variation in object position.

Position your cameras so that each camera can view the entire working volume. This requires that you select optics that provide a wide enough field of view and sufficient depth of field that well-focused images of objects placed anywhere in the working volume can be obtained by all cameras.

Note There is no requirement that all cameras have the same image size, focal length, or other configuration, with one exception: All cameras must have the same handedness. If your configuration uses any mirrors, then the same number of mirrors modulo 2 must be in the optical paths of each camera. You cannot perform 3D camera calibation if some cameras are mirrored but others are not.

Once you have selected the cameras and optics, and then positioned and focused the cameras, you must perform the following steps before acquiring the viewsets:

- Lock the positions and orientations of the cameras with respect to each other. If there is any movement of any camera relative to any other camera during or after calibration, the calibration will become invalid or inaccurate.
- Lock the focus, focal length (for a zoom lens), and aperture of all cameras. If the focus, aperture, or focal length changes for any camera during or after calibration, the calibration will become invalid or inaccurate. Note that this requirement means that you must establish system lighting configuration before calibration; you will not be able to adjust image exposure by changing the lens aperture after calibration.
- Lock the position and orientation of all reflective surfaces (mirrors) or refractive bodies (filters or prisms) in the optical path of any camera. Keep in mind that the optical characteristics of any transparent filters, guards, covers, or windows in the optical path may affect the calibration accuracy.

Acquiring the Viewsets

To successfully perform 3D camera calibration, you must acquire at least four viewsets that show a standard Cognex checkerboard calibration plate at four specific poses. For best accuracy, Cognex recommends that you acquire an additional five viewsets, also with a calibration plate at specific poses.

Note The four required plate poses are approximate; you do not need to precisely position the plate. The five optional plate poses, however, require that you position the plate with very precise relative heights.

You can acquire three types of viewsets during 3D calibration:

- Viewsets of tilted, rotated plates. Four viewsets of this type are required to perform calibration. This pose type is called *Cog3DCalibrationPlatePoseTypeConstants.PoseTilted*.
- Z-offset elevated viewsets. These viewsets are optional, but acquiring this type of viewset improves calibration accuracy. This pose type is called *Cog3DCalibrationPlatePoseTypeConstants.PoseElevated*.
- A single viewset that defines Phys3D space. You can position the calibration plate so that its origin fiducial is at the position that you want to use to define Phys3D space or you can identify one of the other viewsets as the origin viewset. In all cases you can specify a different origin for 3D physical space after calibration is complete. A single viewset that defines the origin of Phys3D is required to perform calibration. This pose type is called *Cog3DCalibrationPlatePoseTypeConstants.PoseDefineWorldCoord*.

Each type of viewset is discussed later in this chapter. Keep the following guidelines in mind when acquiring all viewsets:

- The calibration plate surface should lie within the working volume at all times.
- For single-camera calibration, all images in a viewset must include well-focused views of all calibration plate features. For multi-camera calibration, if some images in some viewsets do not include calibration features, it is generally still possible to compute an accurate calibration. For best results, however, every image in every viewset should include well-focused views of all calibration plate features.
- Depending on how your cameras are positioned, some images in some viewsets may contain no image features. In general, this does not cause a problem for camera calibration, as long as a given camera shares a view of at least some plate poses with another camera.
- Cognex strongly recommends that you save all acquired viewset images to a file archive so that you can experiment with different calibration options without needing to reacquire the images.

Acquiring the Tilted Viewsets (Required)

To acquire the *Cog3DCalibrationPlatePoseTypeConstants.PoseTilted* viewsets, follow these steps:

1. Establish an axis of rotation through the working volume. The axis of rotation should be approximately the average of the optical axes of the cameras that you are calibrating, and it should be approximately centered within the working volume, as shown in Figure 28.

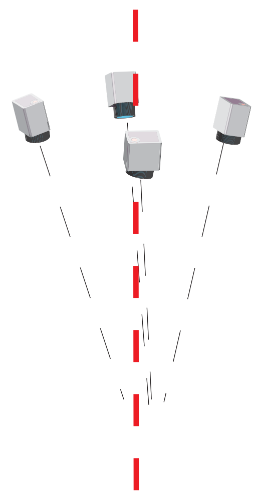


Figure 28. Rotation axis for tilted viewsets

2. Place the calibration plate so that the axis passes through the plate origin, the plate is facing toward the cameras, and the plate is tilted at about 20° from a plane normal to the axis, as shown in Figure 29, and acquire the first viewset.

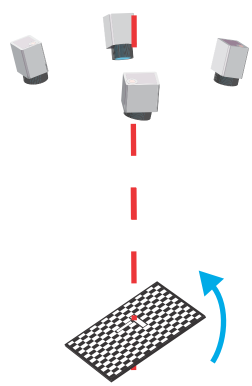


Figure 29. Tilting the calibration plate

3. Rotate the plate about the axis by 90° and acquire another viewset. Repeat the rotation twice more, acquiring a viewset at 180° and 270°. Figure 30 shows the acquisition of all four of the required tilted viewsets:

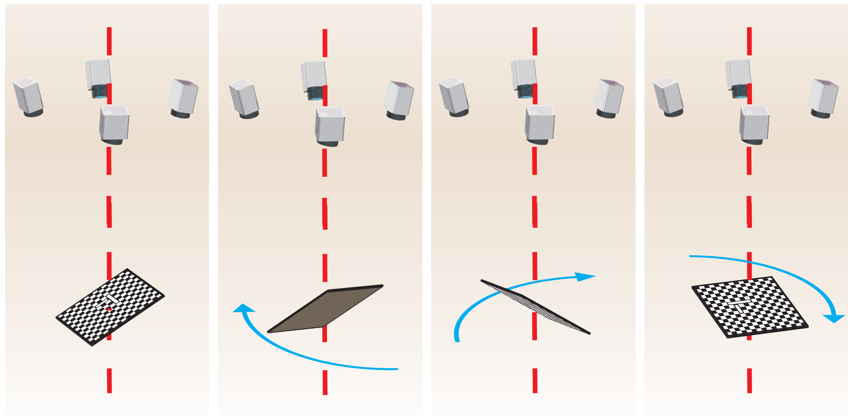


Figure 30. Rotating the calibration plate

Elevated Viewsets (Optional)

Acquiring *Cog3DCalibrationPlatePoseTypeConstants.PoseElevated* viewsets is not required to perform 3D camera calibration, but acquiring these viewsets will result in a more accurate calibration.

To acquire the elevated viewsets, follow these steps:

1. Using the same axis established for the tilted viewsets, place the calibration plate precisely normal to this axis, centered within the working volume, and acquire a viewset, as shown in Figure 31.

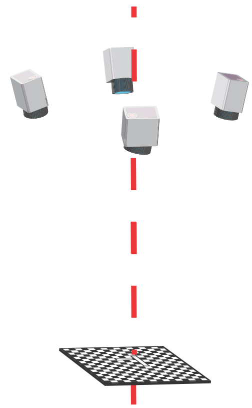


Figure 31. Anchor plate for elevated viewsets.

2. Using an accurate positioning spacer, acquire viewsets of the same plate at evenly spaced positions above and below the first viewset, as shown in Figure 32. You must precisely record the actual spacing between each plate, and the plate must be kept parallel to the first elevated viewset pose.

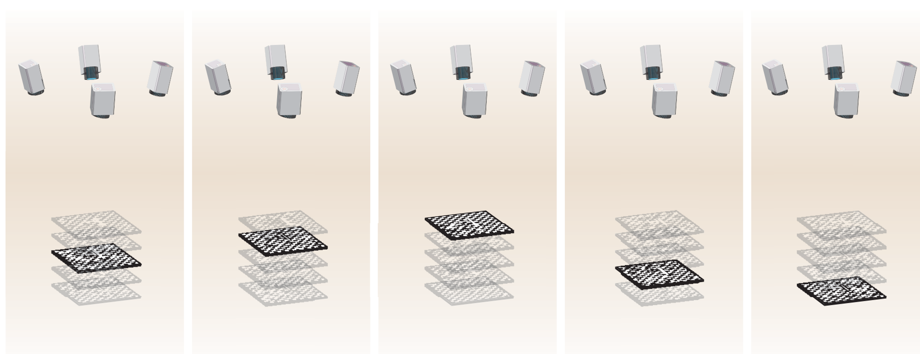


Figure 32. Elevated viewset poses

World Origin Viewset (Required)

To successfully perform 3D camera calibration, one (and only one) of the viewsets provided to the calibration function must define Phys3D space *(Cog3DCalibrationPlatePoseTypeConstants.PoseDefineWorldCoord)*. You can use any one of the viewsets described above as the origin viewset, but in most cases you will create a dedicated world origin viewset by precisely fixing the calibration plate so that its origin corresponds to a known location.

Note If you supply *any* elevated pose type viewsets, the origin viewset must be one of the elevated viewsets, since the elevated pose offset spacing is relative to the Phys3D origin viewset.

Computing Correspondence Pairs

The standard Cognex checkerboard calibration plate establishes a 3D physical coordinate system. Each plate vertex has a known position within that coordinate system, based on the plate vertex spacing and the location of the origin fiducial marks.

Each plate vertex also has a corresponding position within the raw 2D image coordinate system of an image acquired of the plate. The function **Cog3DCheckerboardFeatureExtractor. Execute()** automatically extract calibration plate vertex locations and create a vector of correspondences between 3D plate locations and 2D image locations. These correspondences are stored as a **Cog3DCrspFeaturesCollection** object.

You must call this function for each image in each viewset.

Calibrating

After collecting all of the required correspondence pairs from all of the viewsets, you calibrate by calling a single function: **Cog3DCameraCalibrator.Execute()**.

VisionPro 3D-Locate Calibration

In VisionPro 3D-Locate, **Cog3DCameraCalibrator.Execute()** generates a **Cog3DCalibrationResult** that you use to map points in Raw2D Space into Phys3D Space.

The **Execute()** method accepts:

- The viewsets you acquired or the set of **Cog3DCrspFeaturesCollection** feature correspondences,
- The **Cog3DScalarCollection** object of elevated/tilted plate poses or the object of specified plate poses

- A **Cog3DCalibrationPlatePoseTypeConstants** describing the plate poses.

Intrinsic and Extrinsic Calibration Data

The result of calling **Cog3DCameraCalibrator.Execute()** is a **Cog3DCalibrationResult** object that contains a list of **Cog3DCameraCalibration** objects that contain the 3D camera calibration for each camera being calibrated, as well as information about the measured poses of the cameras relative to the calibration plates and residual error information.

Each **Cog3DCameraCalibration** object contains individual transformation objects for each of the three transformations described in the section *Some Useful Definitions* on page 56:

- **Cog3DCameraCalibration.Raw2DFromCamera2D** describe the camera intrinsics.
- **Cog3DCameraCalibration.Camera3DFromPhys3D** gives the camera extrinsics (the 3D pose of the camera in Phys3D space).
- **Cog3DCameraCalibration.MapPointFromPhys3DToRaw2D** and **Cog3DCameraCalibration.ComputeRayPhys3DFromPointRaw2D()** transform between Phys3D space and Raw2D space.

Specifying a New 3D Physical Space

You must specify one viewset that defines the origin of your Phys3D space when you call **Cog3DCameraCalibrator.Execute()** . The physical pose of the calibration plate fiducial marks (shown in the section *Calibration Plate Requirements* on page 16) determines the origin of Phys3D space. All 3D measurements and poses are reported in this space.

The 3D camera calibration tool supports two techniques that you can use to alter Phys3D space origin after calibration.

- If you wish to redefine Phys3D space based on a new calibration plate pose, simply acquire a viewset containing the plate at the desired pose, then call the overload of **Cog3DCameraCalibrator.Execute()** that allows you to provide precomputed camera intrinsics, supplying the camera intrinsics from the original calibration.
- You can create a new **Cog3DCameraCalibration** object with a Phys3D space that you define by supplying a 3D rigid transformation that specifies the pose of the new space in the current space (you can also define the new space based on the composition of the current space with a specified 3D rigid transformation) by calling the function **Cog3DCameraCalibration.CloneWithNewCamera3DFromPhys3D**, or **Cog3DCameraCalibration.CloseComposeWithPhys3DFromAny3D**.

Assessing the Calibration Quality

There are two methods that you can use to measure the accuracy of a 3D camera calibration.

- The calibration function returns residual error data that indicates how well the computed calibration maps between the observed physical and image points.
- A separate calibration validation tool allows you to validate an existing calibration at any time by simply acquiring a single viewset.

Note Cognex strongly recommends that you use the calibration validation tool. Using the calibration validation tool allows you to establish an accuracy baseline at calibration time, then easily track the calibration accuracy over time.

Interpreting Residual Error Data at Calibration Time

3D camera calibration takes as input collections of correspondences between 3D physical locations and 2D image locations. The computed calibration minimizes the least squares error between the two sets of points. The result of mapping a given 2D or 3D point through the computed calibration is never exactly the same as the corresponding point. This difference is the *residual error*.

The **Cog3DResiduals** class is a container that holds residual error statistics for a given set of point pairs. During 3D camera calibration, the generates two types of residual statistics:

- 3D residual statistics are based on the 3D distance between the physical plate vertex positions and the expected vertex positions in 3D physical units.
- 2D residual statistics are based on the 2D distance between image vertex positions and the expected vertex positions in 2D image units (pixels).

For both 2D and 3D residual error statistics, the tool generates the following objects:

- Statistics for all plate poses and all cameras

  Cog3DCameraCalibrationResult.OverallResidualsRaw2D and **Cog3DCameraCalibrationResult.OverallResidualsPhys3D**

- Statistics for any combination of a single plate pose (*i*) and a single camera (*j*):

  Cog3DCameraCalibrationResult.ResidualsRaw2D[i, j] and **Cog3DCameraCalibrationResult.ResidualsPhys3D[i,j]**

The global residual error measures provide an overall measure of the accuracy of the calibration, while the plate- and camera-level statistics let you identify potential optical or mechanical issues with a particular camera or plate pose.

Using the Calibration Validation Tool

3D-Locate includes a stand-alone tool that you can use to compute a measure of the overall accuracy of an existing 3D camera calibration. The camera calibration validation tool takes the following data as input:

- 3D camera calibration objects for all the calibrated cameras.
- Correspondence pairs from one or more viewsets of a calibration plate. For best results, you should use the same calibration plate for validation that was used to compute the original calibration, but there is no requirement that the plate poses or viewsets used for calibration validation match those used for the original calibration.

Note If you intend to compare the results of validations performed over time, you should use the same plate poses for each validation.

The camera calibration validation tool computes a global RMS measure of the accuracy of the calibration. It computes this results in the following two ways:

- It validates the calibration using the extrinsic parameters computed during the original calibration.
- It recomputes the camera extrinsic parameters using the validation plate poses, then validates the calibration using the newly computed extrinsics.

By comparing these two results, you can determine if increasing error is due to extrinsic changes (movement of cameras relative to each other) or intrinsic changes (focus or aperture changes).

Because you can validate an existing calibration at any time, and because there are no constraints on the plate poses that you use for validation, you can easily include calibration validation as part of the regular maintenance of a 3D vision system.

Hand-Eye Calibration

Hand-eye calibration provides a method that lets you accurately map 3D locations from a 3D calibrated camera's Camera3D space to a physical 3D space known to a robot. This enables the vision system to report 3D poses of objects in a 3D physical coordinate system known to the robot. Hand-eye calibration can be done for systems using a single camera or multiple cameras. If multiple cameras are used, hand-eye calibration can be done simultaneously for all cameras if the images for all cameras can be acquired at the same time at each station.

Figure 33 shows these two 3D spaces and the transformation that are produced by hand-eye calibration.

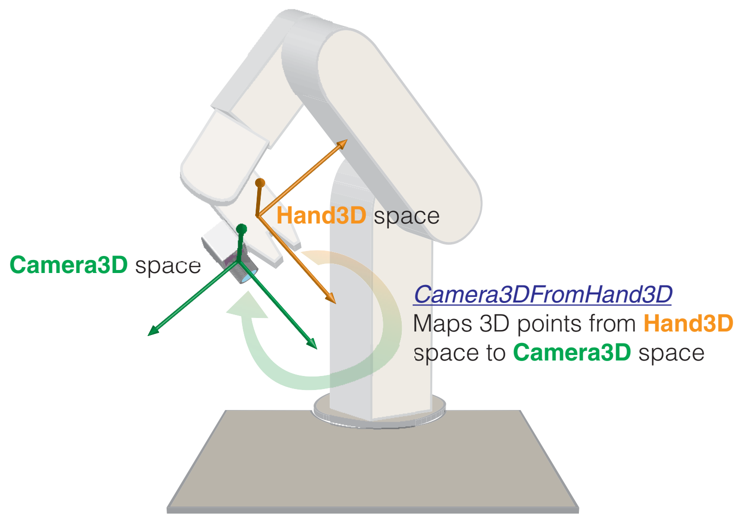
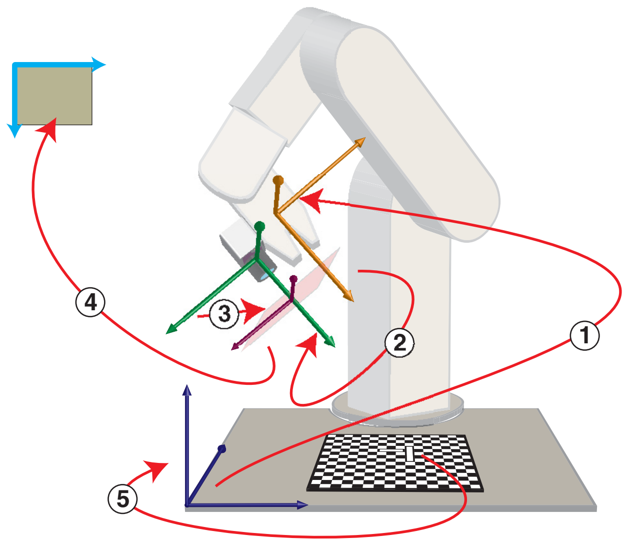


Figure 33. Transformation produced by hand-eye calibration

The Camera3D space shown in Figure 33 represents one end of the linked series standard 3D camera calibration coordinate spaces and transformations. Camera3D space can be mapped to Camera2D space which can be mapped to Raw2D space.

Note The hand-eye calibration tool allows you to use a separately generated 3D camera calibration to provide the intrinsic part of the calibration. In general, using a separate 3D camera calibration procedure will produce a more accurate hand-eye calibration.

Figure 34 shows *all* of the coordinate spaces and transformations associated with hand-eye calibration. Transformation ① is provided by the robot; transformation ② is the hand-eye calibration transformation, transformations ③ and ④ are standard 3D camera calibration transformations.



① **Hand3D** space from **RobotBase3D** space (supplied by robot)

② **Camera3D** space from **Hand3D** space (result of hand-eye calibration)

③ **Camera2D** space from **Camera3D** space (projection)

④ **Raw2D** space from **Camera2D** space (intrinsic part of original camera calibration)

⑤ **RobotBase3D** space from **CalPlate3D** space (result of hand-eye calibration)

Figure 34. Spaces and transformations associated with hand-eye calibration

Stationary Camera/Moving Plate Calibration

The diagrams and discussion in the preceding section describe hand-eye calibration in the case where the camera is mounted on the robot's moving hand and the calibration plate is stationary. Hand-eye calibration is also supported for systems in which the camera is stationary and the calibration plate is attached to the robot hand. In this second case, the computed hand-eye calibration transformation maps points from the Camera3D to the RobotBase3D space, as shown in Figure 35.

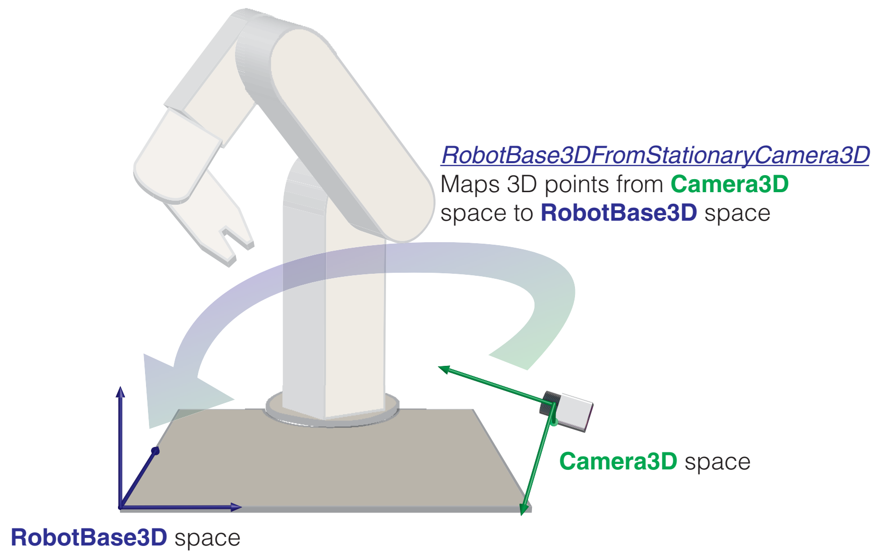
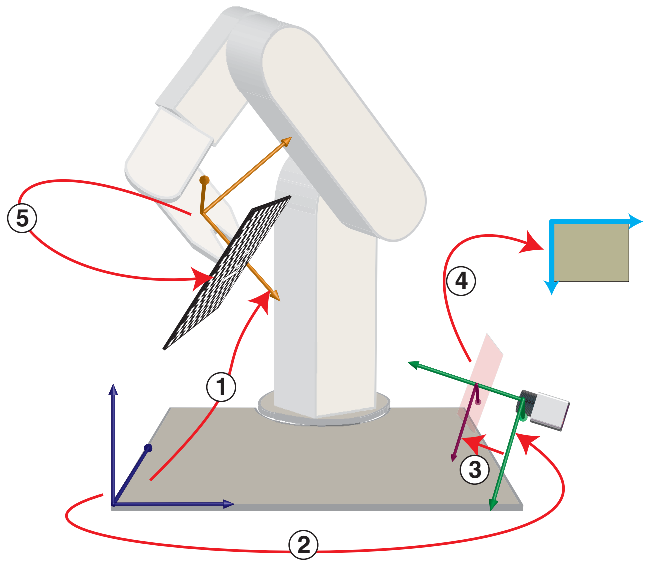


Figure 35. Hand-eye calibration result for stationary camera/moving plate case.

The full outputs for stationary camera/moving plate hand-eye calibration are shown in Figure 36.



1. **Hand3D** space from **RobotBase3D** space (supplied by robot)
2. **RobotBase3D** space from **Camera3D** space (result of hand-eye calibration)
3. **Camera2D** space from **Camera3D** space (projection)
4. **Raw2D** space from **Camera2D** space (intrinsic part of camera calibration)
5. **CalPlate3D** space from **Hand3D** space (result of hand-eye calibration))

Figure 36. Coordinate spaces and transformations associated with stationary camera/moving plate hand-eye calibration.

Hand-Eye Calibration Procedures

The procedure for performing hand-eye calibration differs slightly for the moving camera/stationary plate and stationary camera/moving plate cases. For both types of calibration, however, you must decide how to compute the camera intrinsic parameters:

- You can use the same data set acquired for hand-eye calibration to calibrate the camera(s) first and obtain the camera intrinsics.
- You can perform a separate 3D camera calibration before performing hand-eye calibration. This method is more complex, but it will produce a more accurate hand-eye calibration.

Note One additional type of hand-eye calibration is also supported: The hand-eye calibration tool allows you to provide the camera extrinsics for each calibration input station. In this case, you must have performed a 3D camera calibration and you must use this calibration to compute the extrinsics (the poses of the cameras with respect to the calibration plate) yourself.

In both cases, you perform the calibration by supplying specific input data generated at a number of different robot stations. The inputs for both moving camera/stationary plate and stationary camera/moving plate are shown in Figure 37.

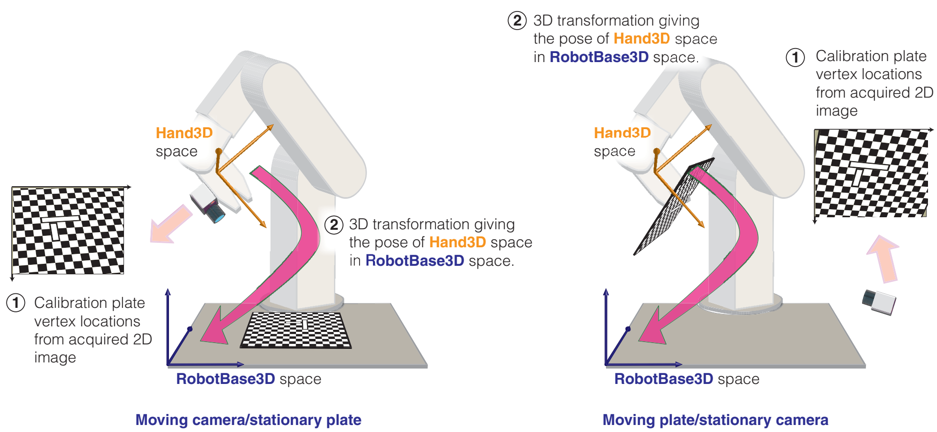


Figure 37. Per-station inputs

Accurate hand-eye calibration depends on the motion between the stations. The motion between stations should include the largest possible rotation that keeps the calibration plate in the FOVs of the cameras. The axis of rotation is also critically imporant in providing an accurate hand-eye calibration. At a minimum, hand-eye calibration should include stations where the axes of rotation are about 3 orthogonal axes. One way to do this is to use three sets of stations, where the first station's axis of rotation is about the robot's X-axis, the second about the robot's Y-axis, and the third about the robot's Z-axis.

The specific steps required to perform hand-eye calibration for both moving camera/stationary plate and stationary camera/moving plate hand-eye calibration are listed below.

Moving Camera/Stationary Plate Calibration

1. If you are performing a separate 3D camera calibration for your camera, perform that now. Note that whatever lens, focal length, focus setting, and aperture that you use for 3D camera calibration must not be changed before or during hand-eye calibration; make sure that the optical configuration that you use for 3D camera calibration is suitable for your hand-eye configuration. You may use the same data set acquired for hand-eye calibration to calibrate the camera and obtain the camera intrinsics.
2. Rigidly mount the camera to the robot end-effector arm. There is no requirement that the camera be mounted to the outermost robot arm segment, as long as the robot can provide the rigid 3D transformation between the robot's gripper or end-effector and the link to which the camera is mounted.
3. Fix the cameras lens, focus, and aperture. In most cases, a positive mechanical lock should be used to prevent any change in the optical configuration associated with the camera.

Note Changing the cameras focus or aperture during or after calibration will invalidate the calibration.

4. Mount the calibration plate. There is no requirement for the plate pose, other than that it cannot move during hand-eye calibration.
5. Move the robot arm so that the camera's field of view is filled by the calibration plate. Ensure that all plate vertices can be viewed in focus by the camera.
6. Acquire an image of the calibration plate. Record the 3D pose of the robot hand (specifically the link to which the camera is attached) in 3DRobotBase space (this information is available for the robot).
7. Repeat steps 5 and 6 at least two times, making sure that the motion of the robot hand meets the requirements listed in the section *Motion Requirements* on page 77. For best results, you should repeat steps 5 and 6 at least 9 times, acquiring plate images from at least ten stations.
8. Extract the feature correspondence pairs from all of the images acquired in step 6 using the function **Cog3DCheckerboardFeatureExtractor.Execute()**.
9. Call the **Cog3DHandEyeCalibrator.Execute()** method with the feature correspondence pairs, the corresponding 3D hand poses, and the 3D camera calibration intrinsics computed in step 1.

Stationary Camera/Moving Plate Calibration

1. If you are performing a separate 3D camera calibration for your camera, perform that now. Note that whatever lens, focal length, focus setting, and aperture that you use for 3D camera calibration must not be changed before or during hand-eye calibration; make sure that the optical configuration that you use for 3D camera calibration is suitable for your hand-eye configuration.
2. Rigidly mount the camera so that it can view the working area of your robot.
3. Fix the cameras lens, focus, and aperture. In most cases, a positive mechanical lock should be used to prevent any change in the optical configuration associated with the camera.

Note Changing the cameras focus or aperture during or after calibration will invalidate the calibration.

4. Mount the calibration plate to the robot arm. In most cases, you will place the calibration plate in the robot gripper or end-effector. You do not need to specify the rigid transform between the robot end-effector (Hand3D space) and the calibration plate (CalPlate3D); this transformation is computed during calibration.
5. Move the robot arm so that the camera's field of view is filled by the calibration plate. Ensure that all plate vertices can be viewed in focus by the camera.
6. Acquire an image of the calibration plate. Record the 3D pose of the robot hand (specifically the link to which the plate is attached) in 3DRobotBase space (this information is available for the robot).
7. Repeat steps 5 and 6 at least two times, making sure that the apparent motion of the calibration plate meets the requirements listed in the section *Motion Requirements* on page 77. For best results, you should repeat steps 5 and 6 at least 9 times, acquiring plate images from at least ten stations.
8. Extract the feature correspondence pairs from all of the images acquired in step 6 using **Cog3DCheckerboardFeatureExtractor.Execute()**
9. Call the **Cog3DHandEyeCalibrator.Execute()** method with the feature correspondence pairs, the corresponding 3D hand poses, and the 3D camera calibration intrinsics computed in step 1.

Motion Requirements

For all types of hand-eye calibration, the following requirements must be met:

- At least three separate sets of input data (stations) must be provided.
- The plate should be positioned so that the plate fills the camera's field of view and all or substantially all of the plate vertices are in focus.

- Between any two adjacent stations, there must be significant rotation of the calibration plate with respect to the camera. In general, larger rotations provide better results than smaller rotations.
- The rotation between any two adjacent stations may not be exactly 180°
- At least two of the axes of rotation may not be parallel. For best results, none of the axes should be parallel.

Residual Error

Like the 3D camera calibration tool, the hand-eye calibration tool can generate residual error statistics that you can use to assess the accuracy of the calibration. Unlike the 3D camera calibration tool, the hand-eye calibration tool does not use the calibration plate vertex positions to compute residual error. Instead, it takes a set of evenly spaced points from the 3D physical space defined by the calibration plate at a given input station and maps those points to the 3D physical space defined by the calibration plate at the first input station.

In the absence of any error, the points should be unchanged by the mapping process. In most cases, however, the following sources of error are typically present:

- Uncorrected optical distortion
- Calibration plate defects (inconsistent vertex spacing, non-coplanarity)
- Variance between reported and actual robot hand pose.

The hand-eye calibration tool allows you to specify the number of sampling points that are used to compute the residual error statistics, and computes the residual statistics for all sampling points measured across all stations.

.NET Classes and Sample Code

This section lists the functionality and .NET classes for calibration contained within the Cognex.VisionPro3D namespace.

3D Camera Calibration

The following classes support camera calibration:

Cog3DCalibrationFeatureExtractorBase
Cog3DCameraCalibration
Cog3DCameraCalibrationDistortionModelConstants
Cog3DCameraCalibration
Cog3DCameraCalibrationResult
Cog3DCameraCalibrationValidationResult
Cog3DCameraCalibrationValidator
Cog3DCameraCalibrationIntrinsics
Cog3DCameraCalibrator
Cog3DCheckerboardFeatureExtractor

Sample Code

The sample code located in
%VPRO_ROOT%\samples3d\Programming\Setup\Calibration\CameraCalibration
shows how to calibrate cameras. Copy the contents to a folder where you have write permission before you execute it.

Hand-Eye Calibration

The following classes support hand-eye calibration:

Cog3DHandEyeCalibrator
Cog3DHandEyeCalibrationResult
Cog3DHandEyeCalibrationValidator
Cog3DHandEyeCalibrationValidationResult

Sample Code

The sample code file located in
%VPRO_ROOT%\samples3d\Programming\Setup\Calibration\HandEyeCalibration
shows how to perform hand-eye calibration. Copy the contents to a folder where you have write permission before you execute it.

Locating Objects in 3D

4

This chapter describes how to perform a 3D pose estimation for one or multiple objects in an image set acquired from 3D calibrated cameras.

The sample code installed in %VPRO_ROOT%*samples3D\Programming* includes several Visual Studio 2010 solutions that demonstrate how to perform a 3D pose estimation. Copy the contents to a folder where you have write permission before you execute them.

This chapter contains the following sections:

- *Some Useful Definitions* on page 82 defines some terms that you will encounter as you read this chapter.
- *3D Vision Applications* on page 83 contains a block diagram to summarize the 3D pose estimation process.
- *2D Part Location and 2D Feature Location* on page 86 summarizes the process of locating individual part instances in a scene and 2D feature location.
- *2D Image Feature to 3D Model Feature Correspondence* on page 89 describes how to define the correspondence objects that map 2D features to 3D features.
- *3D Models* on page 96 describes how to create the 3D model of your object used in 3D pose estimations.
- *Part Correspondence* on page 103 describes how to ensure your application can perform a 3D pose estimation for multiple parts in a single scene.
- *3D Pose Estimation* on page 110 describes how to perform a 3D pose estimation using all available data.

Some Useful Definitions

This section defines some terms and concepts used in this chapter.

3D Pose The position and orientation of a 3D coordinate system within another 3D coordinate system. A pose comprises 6 degrees of freedom: X-translation, Y-translation, Z-translation, X-rotation, Y-rotation, and Z-rotation.

3D Position The location of a 3D point within a 3D coordinate system. A 3D position is represented by an x-value, y-value, and z-value.

3D Pose Uncertainty The variation in a part's pose from its nominal pose, for your application.

3D-Calibrated Camera A camera for which a 3D calibration (both extrinsic and intrinsic) has been computed.

Raw2D Space Left-handed 2D coordinate space based on the pixels in an acquired image.

Camera3D Space A right-handed 3D coordinate space with its origin at the camera's optical convergence point, X- and Y-axes that are approximately parallel to and oriented in the same direction as the Raw2D coordinate system X- and Y-axes, and a Z-axis that extends along the optical axis away from the camera.

Camera2D Space The plane at Z=1 of Camera3D Space. When Camera2D space is viewed from the camera (Z < 1 and in the direction of the Camera3D positive Z axis) then Camera2D space appears as a left-handed 2D coordinate system. When Camera2D space is viewed in the direction of the Camera3D negative Z axis from a point in front of the camera (where Z > 1), then Camera2D Space appears as a right-handed 2D coordinate system.

Phys3D Space A right-handed 3D coordinate space initially defined by the fiducial mark on the calibration plate specified as having an origin-defining pose-type used to perform 3D calibration. This space can be defined by any coordinate frame in physical space.

Hand3D Space A right-handed 3D coordinate space defined by the end-effector on a robot. The position of the robot hand is reported by the robot controller as the pose of Hand3D space in RobotBase3D space. (Some robot manufacturers and integrators refer to this as *tool space*.)

RobotBase3D Space A right-handed 3D coordinate space defined by the robot manufacturer or integrator. It is typically associated with the robot base (the part of the robot that is rigidly fixed to the physical world).

.

3D Vision Applications

The following diagram summarizes the setup and runtime tasks your 3D application must accomplish in order to perform a successful 3D pose estimation.

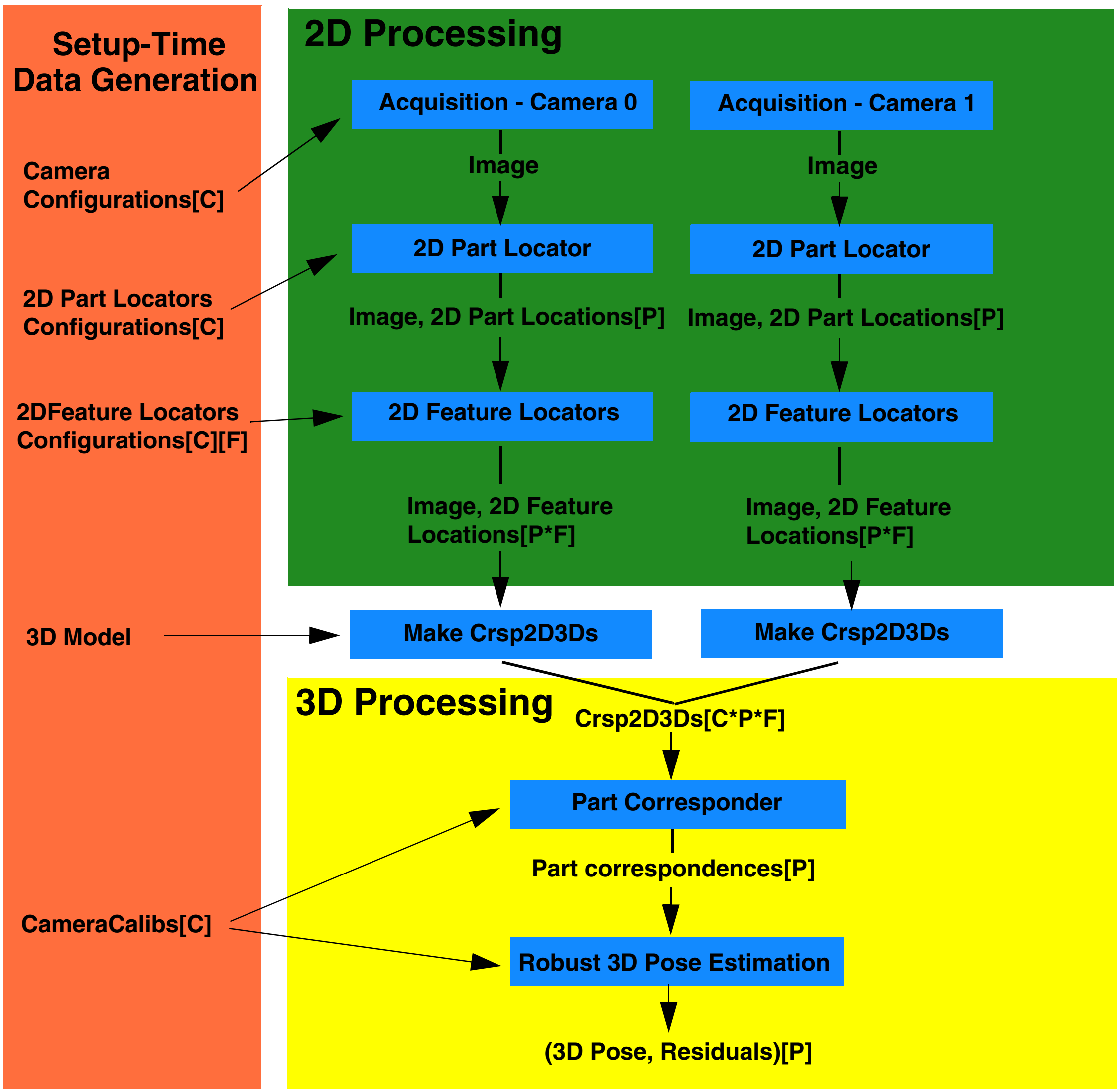


Figure 38. Application architecture (C = Number of cameras, P = Number of parts, F= Number of 2D Features)

3D Application Architecture

VisionPro 3D-Locate API provides all the pieces you need to build a 3D application that can determine the 3D pose of your part. Cognex recommends a 3D application architecture similar to the architecture implemented by the VisionPro 3D-Locate Starter Kit. This section describes that architecture and the remainder of the chapter describes the VisionPro 3D-Locate functionality used to implement the architecture.

Setup-Time Data Generation

Your 3D application needs to support various setup-time processes or steps. The setup-time steps produce much of the data that will be used at runtime. Cognex suggests you provide support in your 3D application for the following setup-time steps.

- Camera Configuration (1 per camera)

  Allows the setting of camera parameters like exposure time, contrast, brightness, trigger type.

- Camera Calibration (1 per camera)

  Produces a camera calibration object for each camera. Most of the tools used in the Runtime 3D component require camera calibrations.

- Camera Calibration Validation (1)

  Allows fast verification (at some time in the future) of how well your cameras are matching your camera calibration.

- Configure 2D Part Locators (1 per camera)

  2D part locators are VisionPro 2D vision tool(s) used to coarsely locate your part in acquired images.

- Configure 2D Feature Locators (1 per feature per camera)

  2D feature locators are VisionPro 2D vision tool(s) used to accurately locate features on your part in acquired images.

- 3D Model Creation (1)

  Create a 3D model (list of simple 3D shapes) that is a 3D geometric representation of your part. Do this by manually specifying all the parameters of each 3D model feature or by using the ModelFeatureGenerator.

2D Processing

Your 3D application needs to support the 2D location of parts and their features in acquired images. 2D processing begins with the simultaneous acquisition of an image from each camera. Each image is independently processed by first locating all the parts in the image and then locating all the features on each part.

- Acquisition

  Acquire an image from each camera.

- 2D Part Location

  For each image, locate all the parts in it.

- 2D Feature Location

  For each part in each image, locate all the features on the part.

- Crsp2D3Ds Creation

  Setup-time

    If you are going to use the ModelFeatureGenerator to produce the complete 3D model, then Crsp2D3Ds need to be created.

  Runtime

    Crsp2D3Ds are created at runtime and then passed to the Part Corresponder.

3D Processing

Your 3D application needs to support the final 2 steps in computing 3D poses for the parts in the acquired images.

- Part Correspondence

  Determines the correspondence of parts across cameras. When you Make Crsp2D3Ds and fill them in with 2D features, those 2D features have affinity to a single part in one image. The part corresponder determines the correct affinity for 2D features across all the parts in all the images.

- 3D Pose Estimation

  The pose estimator uses the results of part correspondence to compute the 3D pose of a part. It uses all the Crsp2D3Ds that apply to that part plus the 3D model plus optional robust pose estimation parameters to compute the pose that best fits the 3D model to all the 2D features located in all the images for the part.

2D Part Location and 2D Feature Location

The purpose of the 2D part location phase of your 3D application it to coarsely locate all the parts in the camera's field of view. The parts only require coarse location because the next phase is 2D feature location. In the 2D feature location phase, specific features of the object will be located in each image with high accuracy. It is these 2D features that are eventually used for 3D pose estimation.

2D Part Location

A 3D vision environment, once calibrated, can estimate the 3D pose of a part from 2D images acquired from multiple cameras.

Many production environments cannot guarantee the precise placement of the object for each inspection, so the first goal of the 3D vision application is to perform 2D part location, locating the object in the camera's field of view and determining its position, rotation and angle.

VisionPro provides a variety of 2D vision tools determining the location of a part. For example, a PMAlign tool can be configured to locate a specific pattern of known features, as shown in Figure 39:

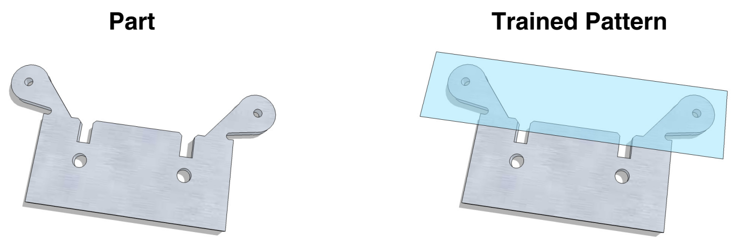


Figure 39. Trained Pattern

Once trained, a PMAlign tool can be configured to locate single or multiple instances of the part in the field of view, as shown in Figure 40:

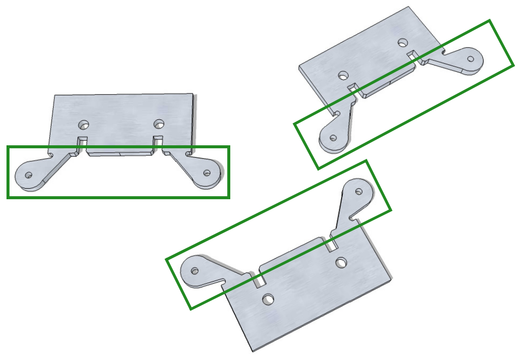


Figure 40. Multiple Parts Located

The primary challenge in the 2D part location phase of your 3D application is correctly configuring the 2D part location tool(s) to handle the perspective distortion in the image of your part(s) as the part(s) moves around and tilts in the camera's field of view. For example, configuring a PMAlign tool to accommodate some amount of perspective distortion often requires:

- Enabling the ZoneScaleX and ZoneScaleY degrees of freedom
- Increasing the Elasticity to 5-10

The 3D application does not directly use the results of 2D part location for 3D pose estimation. Instead, other 2D vision tools are positioned relative to the results of the 2D part location and the results of those 2D vision tools are used in the calculation of the 3D pose estimation. This step is known as 2D feature location.

The sample code installed in %VPRO_ROOT%*samples3D\Programming* includes several Visual Studio 2010 solutions that all perform part location. Copy the contents to a folder where you have write permission before you execute them.

2D Feature Location

Once the coarse 2D location of a part has been determined, you use subsequent 2D vision tools to perform 2D feature location, analyzing the part for discrete 2D features that represent known 2D points, 2D circles and 2D line segments. The 2D features you locate become part of the data structures that can be used to build a model of the part or perform 3D pose estimation.

Using the same part as the previous section as an example, a 3D vision application might use two PMAlign tools to locate two 2D point features and two Find Line tools to locate two 2D line segments over the areas highlighted in Figure 41:

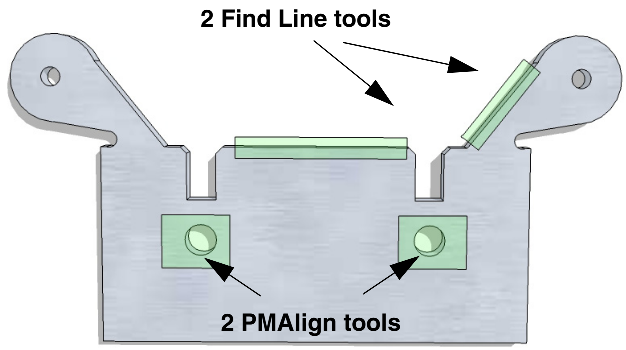


Figure 41. Four 2D Features Found

The 2D feature location vision tools execute and generate line segments and point results, as shown in Figure 42:

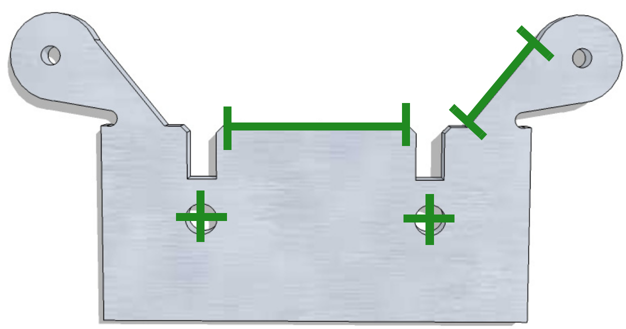


Figure 42. Line Segments and Point Results

In images containing multiple parts, the 2D vision tools you use for 2D part is configured to locate multiple instances of the part. Then the same set of 2D feature location vision tools analyze the acquired images relative to each found 2D instance.

2D Image Feature to 3D Model Feature Correspondence

A key component to any VisionPro 3D application is the *feature correspondence* between a single 2D feature found in an image and the 3D feature it corresponds to on the part itself. To define this relationship, the VisionPro 3D API supports the **Cog3DCrsp2D3D** class. **Cog3DCrsp2D3D** objects are used to create a mathematical 3D model of your part as well as determine 3D pose estimations.

An object of type **Cog3DCrsp2D3D** associates a 3D geometric feature with a 2D feature found in one image of an image set. A single **Cog3DCrsp2D3D** object includes the following information:

- The index of the camera from which the image was acquired
- An index of the found part instance in the field of view
- An index into the list of 3D geometric features that form the 3D model of the part
- The type of the 3D model feature
- One or more 2D points (a **Cog3DVect2** object or a **Cog3DVect2Collection** object) that defines the found 2D feature

A **Cog3DCrsp2D3D** object can be represented as shown in Figure 43:

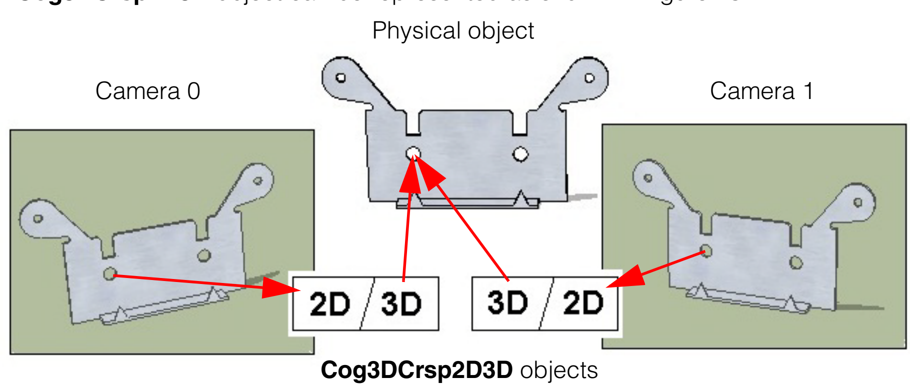


Figure 43. A Cog3DCrsp2D3D feature correspondence

Ultimately, your 3D vision application must create a list of **Cog3DCrsp2D3D** objects for all the 2D features found on all the parts on all the images of an image set.

At setup time, you use a list of **Cog3DCrsp2D3D** objects to create a 3D model, as described in the section *3D Models* on page 96. At runtime, the list of **Cog3DCrsp2D3D** objects is used as an input parameter for part correspondence and 3D pose estimation methods, as described in the sections *Part Correspondence* on page 103 and *3D Pose Estimation* on page 110.

The number of **Cog3DCrsp2D3D** objects your application generates is a product of the following multipliers:

- The number of cameras you use
- The number of parts in the field of view, assuming all cameras can see all the parts
- The number of 2D features, per part, your application uses

For example, an application that uses two cameras and four 2D features per part on a field of view with three parts will need to generate a total of (2 x 4 x 3) = 24 **Cog3DCrsp2D3D** objects.

Creation Algorithm

Your application must assign the correct values to the properties of each **Cog3DCrsp2D3D** object. As your 3D application executes, a list of **Cog3DCrsp2D3D** objects is typically generated as the application locates 2D features in each image, and the process can be described with the following algorithm:

```
For CameraIndex=0 to numCameras-1
  For PartInstanceIndex=0 to numParts-1 in Image[CameraIndex]
    For 2DFeatureIndex=0 to num2DFeatures per part
      {
      Locate 2DFeature[2DFeatureIndex] on Part[PartInstanceIndex]
                                                 in Image[Camera]
      Create and fully initialize a Crsp
      Add the Crsp to the List<Crsp>
      }
```

The sample code installed in %VPRO_ROOT%*samples3D\Programming\Setup\ModelGeneration* includes Visual Studio 2010 solutions for creating **Cog3DCrsp2D3D** objects. Copy the contents to a folder where you have write permission before you execute them.

Properties

A **Cog3DCrsp2D3D** object has six properties, where three relate to the 3D model feature and three relate to the 2D feature. Figure 44 lists the properties:

3D Model Feature Data

FeatureModel3DIndex	FeatureModel3DType	Subfeature
int	Type	enum

2D Image Feature Data

CameraIndex	PartInstanceIndex	FeatureRaw2D
int	int	Object

Figure 44. Properties of a Cog3DCrsp2D3D object

FeatureModel3DIndex

As you develop your 3D application, you define the order of 3D model features as they are added to the model. For example, Figure 45 highlights the 3D model features over an image in the order which they are created (the order is zero-based):

0: Point feature

1: Point feature

2: Edge feature

3: Edge feature

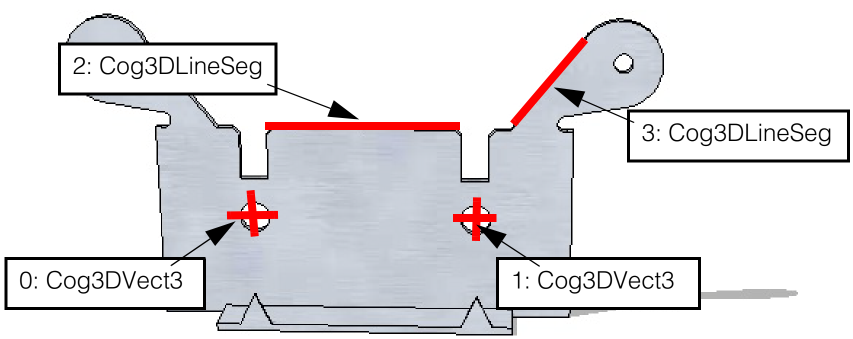


Figure 45. 3D features of a model

Using this image, if you are creating a **Cog3DCrsp2D3D** object for the hole on the right side of the part, you would assign it a **FeatureModel3DIndex** value of “1”, since it is the second model feature the application locates.

FeatureModel3DType

The **FeatureModel3DType** property identifies the .NET Type of the 3D model feature. For example, the following programming statement sets **FeatureModel3DType** property for a **Cog3DCrsp2D3D** object named *crsp* that represents a line segment in your 3D model:

```
crsp.FeatureModel3DType = typeof(Cog3DLineSeg);
```

The property must be set for one of the following supported 3D model features:

- **Cog3DVect3**
- **Cog3DCircle**
- **Cog3DLineSeg**
- **Cog3DLine**
- **Cog3DCylinder**

See the section *3D Models* on page 96 for more information on creating a 3D model.

Subfeature

The **Subfeature** property defines an enumeration value of type **Cog3DSubfeatureConstants** that corresponds to the found 2D feature.

- If your found 2D feature is a point, specify a **Subfeature** property value of **Cog3DSubfeatureConstants.Point0**.
- If your found 2D feature is a line or line segment, specify a **Subfeature** property value of **Cog3DSubfeatureConstants.StraightEdge0**.
- If your found 2D feature is a circle, specify a **Subfeature** property value of **Cog3DSubfeatureConstants.CircularEdge0**.

  Optionally, you can define a second **Cog3DCrsp2D3D** object for any circle to represent the point at the circle’s center. Use a **Subfeature** property value of **Cog3DSubfeatureConstants.Point0** for this second **Cog3DCrsp2D3D** object.

If the 3D feature you want to locate is a cylinder, your application may locate up to four 2D features: 2 circular features for the start circle and end circle of the cylider and 2 edge features for the occluding edges of the cylinder, as shown in Figure 46:

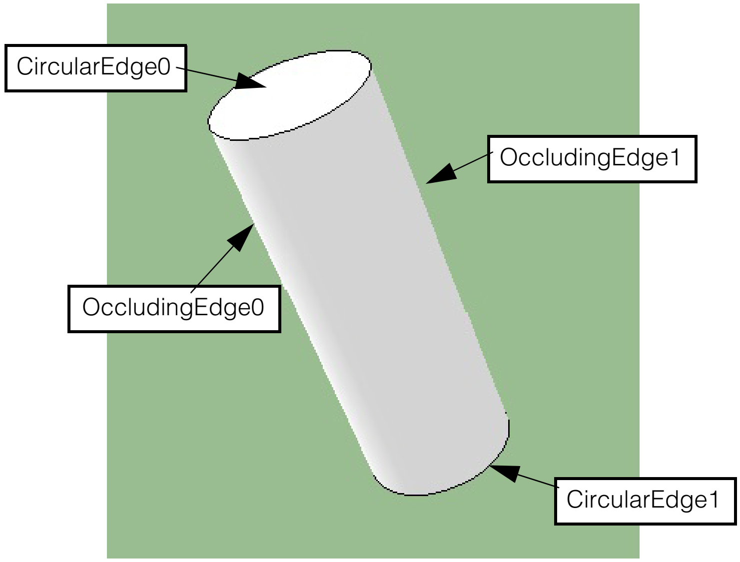


Figure 46. Features of a cylinder

For the four **Cog3DCrsp2D3D** objects needed to represent these features, you must specify a different **Subfeature** property:

- The two straight features of the cylinder must be assigned a **Subfeature** property of **Cog3DSubfeatureConstants.OccludingEdge0** and **Cog3DSubfeatureConstants.OccludingEdge1**, respectively.
- The two circular features of the cylinder must be assigned a **Subfeature** property of **Cog3DSubfeatureConstants.CircularEdge0** and **Cog3DSubfeatureConstants.CircularEdge1**, respectively.

CameraIndex

The **CameraIndex** property indicates which camera was used to capture the image in which these 2D features were located. If you are defining **Cog3DCrsp2D3D** objects using the algorithm described in the section *Creation Algorithm* on page 90, set the **CameraIndex** property to the current value of `CameraIndex`.

PartInstanceIndex

The **PartInstanceIndex** property indicates an index for each part in one image of the image set. For scenes with only one part, the value is 0. If you are defining **Cog3DCrsp2D3D** objects using the algorithm described in the section *Creation Algorithm* on page 90, set the **PartInstanceIndex** to the current value of `PartInstanceIndex`.

In applications that capture image sets of more than one part, the **PartInstanceIndex** value for **Cog3DCrsp2D3D** objects you create will range from 0 to *numParts-1* in the current image. Your application must assign a unique value of **PartInstanceIndex** for each part in each image. For example, the following figure shows an image set and lists the values for **PartInstanceIndex** across both images:

Camera 0 **Camera 1**

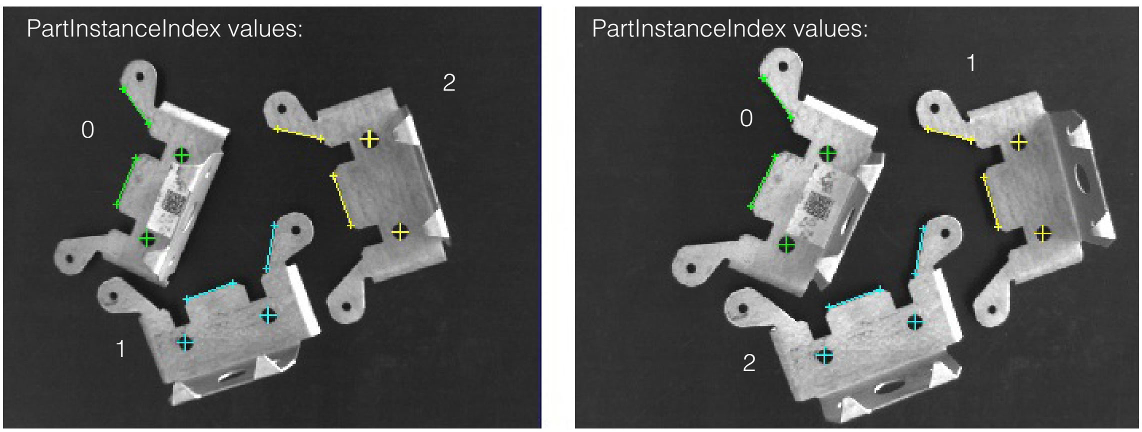


Figure 47. PartInstanceIndex Values for Three Parts

Cog3DCrsp2D3D objects where the **PartInstanceIndex** values do not match across the image set are *non-unified* and cannot be used for 3D pose estimation.

To solve the correspondence problem when your image sets contain multiple parts, use the part corresponder class **Cog3DPartCorresponderUsing2DPoses** or **Cog3DPartCorresponderUsingCrsps2D3Ds**. See the section *Part Correspondence* on page 103 for more information. The part correspondence class accepts non-unified **Cog3DCrsp2D3D** objects and the generated 3D model and returns *unified* **Cog3DCrsp2D3D** objects that can now be used for 3D pose estimation.

FeatureRaw2D

The **FeatureRaw2D** property is either a **Cog3DVect2** object or a **Cog3DVect2Collection** object and is used to store the coordinates of 2D points that comprise the found 2D feature in the image.

For 2D features represented by a single point, create and initialize a **Cog3DVect2** object. For all other types of results (line segments, circles, cylinders), create and initialize a **Cog3DVect2Collection** object.

2D features must be in Raw2D coordinate space. Most applications run the 2D vision tools based on a fixturing scheme that positions the 2D feature location tools based on an initial found feature, using a tool such as the CogPMAlign tool. This allows the 2D feature location tools to run in fixtured space, but the 3D vision tools for 3D model creation and 3D pose estimation require the **FeatureRaw2D** property to be specified in root space. Therefore, your application must map the result from fixtured coordinate space to the root coordinate space.

Refer to the sample code in %VPRO_ROOT%*samples3D\Programming\Setup\ModelGeneration\CreateCrsp* for sample code that transforms the results back to root coordinate space. Copy the conents to a folder where you have write permission before you execute it. In addition, see your VisionPro online documentation for details on coordinate spaces used before and after using a fixturing mechanism.

Your application should set this property to an initial value of null in the event that the 2D feature was not found, and then change the value based on the 2D vision tool result. A list of **Cog3DCrsp2D3D** objects where some of the objects have a **FeatureRaw2D** value of null can still be used for robust pose estimation, as described in the section *3D Pose Estimation* on page 110.

3D Models

Your 3D vision application must have a 3D model of the part for which you want to generate a 3D pose estimate. The 3D model is a list of 3D features within *Model3D* coordinate space. A 3D pose estimation method computes the 3D pose of the part based on the location of 2D features in an image set and the part's 3D model. Creating a 3D model is a setup task you perform as you develop your application.

During runtime, your vision application uses the 3D model as a parameter for Part Correspondence (see page 103) and 3D pose estimation, along with the saved camera calibration data and the **Cog3DCrsp2D3D** objects generated from the runtime image set of one or more parts.

3D Model Features

A 3D model feature can be one of the following 3D classes:

- **Cog3DVect3**
- **Cog3DCircle**
- **Cog3DLineSeg**
- **Cog3DLine**
- **Cog3DCylinder**

For example, Figure 48 shows an image of a part and the 3D model features that can be used to define the 3D model:

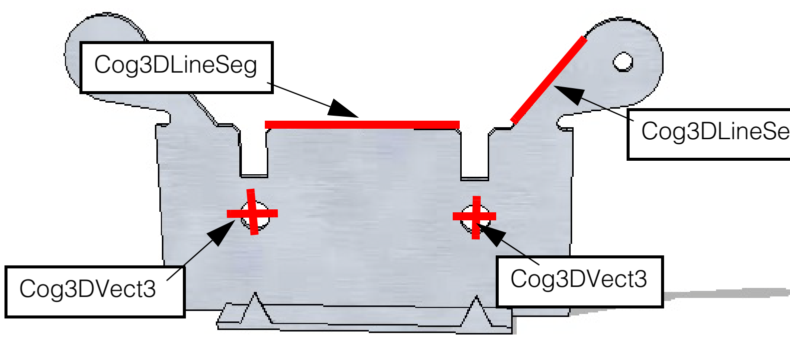


Figure 48. A 3D Model

You typically choose 3D features for your 3D model based on the 2D features you can consistently locate in your image sets. You should choose stable 2D features that can tolerate the 3D pose uncertainty of your 3D application. The following are typically good 2D features:

- Sharp, straight edges
- Circles
- Unique features that can be found regardless of rotation, scale or translation changes

Similarly, avoid the following types of 2D features:

- Round corners
- Round edges
- Features that extrude significantly towards the camera from the rest of the part

3D Model Creation

You can create a 3D model using a list of **Cog3DCrsp2D3D** objects that associate the 3D features (type and index) of your part with the 2D features visible in the image set of a single part. See the section *2D Image Feature to 3D Model Feature Correspondence* on page 89 for details on how to create **Cog3DCrsp2D3D** objects.

To create a 3D model of a part with the **Cog3DModelFeatureGeneratorUsingCrsp2D3Ds** class, your application must perform the following steps:

1. Capture multiple image sets of a single part.

   Cognex recommends five image sets stored in an image-database file, as described in the section *Image Sets* on page 98.

2. Create a List<List<**Cog3DCrsp2D3D**>> based on the 2D features in all the image sets. The double-indexed list is indexed by [*imageSetIndex*][*crspIndex*].

3. Pass the **Cog3DCrsp2D3D** objects, along with the camera calibration data, to the **GenerateFeaturesModel3D( )** method of the **Cog3DModelFeatureGeneratorUsingCrsp2D3Ds** class.

   The **GenerateFeaturesModel3D( )** method returns a **Cog3DModelFeatureGeneratorUsingCrsp2D3DsResults** object, which is a collection of 3D model features stored in individual **Cog3DModelFeatureGeneratorUsingCrsp2D3DsResult** objects, and a collection of the 3D poses of your part in each image set.

4. Initialize a variable of **List<Object>** to store your 3D model.

5. Examine the **IsFound** property of each **Cog3DModelFeatureGeneratorUsingCrsp2D3DsResult** result to determine if this 3D model feature was found.

   If the **IsFound** property is **False**, the **Cog3DCrsp2D3D** objects did not contain enough data to generate a model feature candidate. You can re-examine your image sets to reconfigure or modify the set of 2D vision tools you use to locate 2D features.

   If the **IsFound** property is **True**, use the **GetResidualsValid( )** method and the **GetResidualsPhys3D( )** and **GetResidualsRaw2D( )** methods to examine the **Cog3DResiduals** residual error information.

   You should examine the residual data and determine if the 3D model feature is accurate enough for your production environment. If necessary, you can modify the 2D vision tools used to extract 2D feature information and re-run them on your image-database of image sets in order to generate a new set 3D model features.

The sample code installed in %VPRO_ROOT%*samples3D\Programming\Setup\ModelGeneration* includes Visual Studio 2010 solutions for 3D model feature generation. Copy the contents to a folder where you have write permission before you execute them.

Image Sets

To create a 3D model using the 3D model feature generator, you must capture at least one image set of a single part. If possible, however, Cognex recommends you capture five image sets of your part, creating a List<List<**Cog3DCrsp2D3D**>> objects generated from these image sets and passed to the 3D model feature generation method. This set of **Cog3DCrsp2D3D** objects can be indexed as:

[image set index] [crsp index]

The five image sets you create should place the part at five different poses:

0. Place the part as close to possible to the origin of Phys3D space.

   This will be treated as the identify pose. The origin of the Model3D coordinate space (the coordinate space your 3D model features are defined in) is defined by the Phys3D origin in this pose.

1. Place the part in the upper left corner of the field of view visible by both cameras, and use a z-axis rotation of ~20 degrees.
2. Place the part in the upper right corner of the field of view visible by both cameras, and use a z-axis rotation of ~45 degrees.

3. Place the part in the lower left corner of the field of view visible by both cameras, and use a z-axis rotation of ~67 degrees.
4. Place the part in the lower right corner of the field of view visible by both cameras, and use a z -axis rotation of ~90 degrees.

Figure 49 illustrates the five poses:

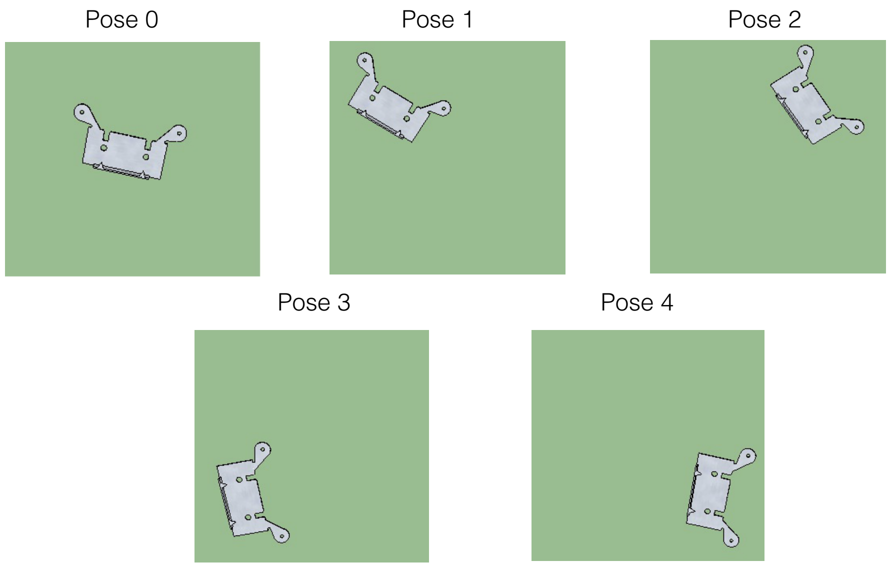


Figure 49. Five part poses for model generation

Cognex recommends you store the image sets and the camera calibration data, which allows you to examine the accuracy of the 2D feature extraction at any time later, or change the vision tools you use to extract 2D feature information without the need to acquire new image sets. Storing the image sets and the camera calibration data also creates a record of the image sets you use for 3D model generation in the event that you need to contact Cognex technical support.

See your VisionPro online documentation for details on saving acquired images in an image-database file.

Lines and Line Segments

As you create the **Cog3DCrsp2D3D** objects to be used during 3D model feature generation, be aware of the following issues regarding lines and line segments.

Line Segment Endpoints

In some vision applications a 2D vision tool configured to detect an edge might not return valid edge results for all potential edge points. If you create a **Cog3DCrsp2D3D** object based on these types of results, the **Cog3DLineSeg** will not accurately reflect the desired length of the line segment you want for your 3D model.

For example, Figure 50 shows the found edge points as well as the edge points that were not included in the best-fit line, and the length of line segment it would generate based only on the included 2D edge points:

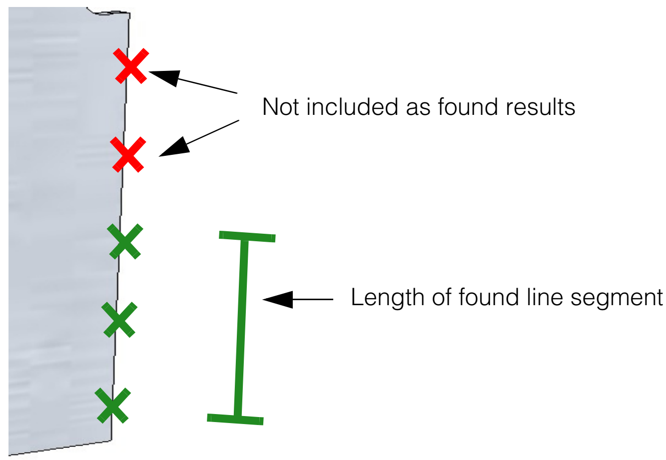


Figure 50. A Cog3DLineSeg

Thus, the 3D line segment in your 3D model would be much shorter than the actual line segment on your part.

If the detected edge in your runtime image sets included different edge points, they may correspond to a noticeably long line segment as shown Figure 51:

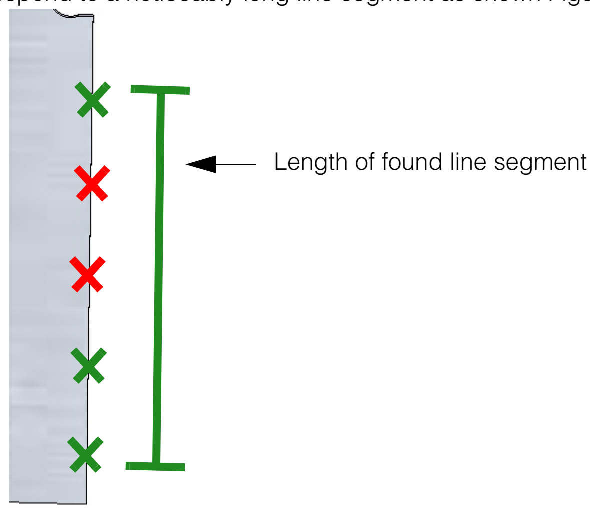


Figure 51. Comparing Cog3DLineSeg features

This difference between the short line segment in your 3D model and the long line segment found at runtime will result in higher residual errors than if the line segment added to the 3D model accurately reflected the actual size of the edge.

If you are using a **Cog3DLineSeg** for any of your 3D model features, Cognex recommends you explicitly add the endpoints of the found line segment to the **FeatureRaw2D** property of any **Cog3DCrsp2D3D** objects you generate for 3D model feature generation.

See the sample code installed in %VPRO_ROOT%*samples3D\Programming\Setup\ModelGeneration* for an example of adding the endpoints of a line segment to the **FeatureRaw2D** property of a **Cog3DCrsp2D3D** object. Copy the contents to a folder where you have write permission before you execute it.

Edges Parallel To Baseline

Cognex recommends you create an image-database of five image sets of your part, as described in the section *Image Sets* on page 98. If your production environment prevents that possibility, however, you can still create a 3D model using a single image set of your part.

If you are using a single image set for your 3D model creation, Cognex strongly recommends that you position the part in front of the cameras such that the part's line and line segment features are not parallel to the baseline between the cameras, as shown in Figure 52:

Avoid Use

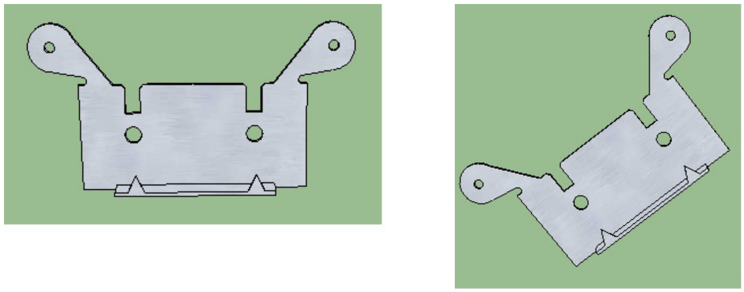


Figure 52. Edges Parallel to Baseline

Any 3D model features generated from edges that are parallel to the baseline will likely be inaccurate.

Part Correspondence

The section *2D Image Feature to 3D Model Feature Correspondence* on page 89 describes how to create the **Cog3DCrsp2D3D** objects necessary to define the feature correspondence between a single 2D feature found in an image and the 3D feature it corresponds to on the part itself.

Your application must assign the **PartInstanceIndex** property of each **Cog3DCrsp2D3D** object, which specifies an index for each part in one image of the image set. For scenes with only one part, the value is 0. In applications that capture image sets of more than one part, the **PartInstanceIndex** value for **Cog3DCrsp2D3D** objects you create will range from 0 to *numParts-1* in the image you are working on.

Depending on the method you use to locate respective parts in each image, the algorithm might not ensure that the same part receives the same value for **PartInstanceIndex** across the image set. **Cog3DCrsp2D3D** objects generated this way are known as *non-unified*, and **Cog3DCrsp2D3D** objects must be unified before they can be used to perform a 3D pose estimation.

For example, Figure 53 shows an image set where the **PartInstanceIndex** value does not match for the same part across both images:

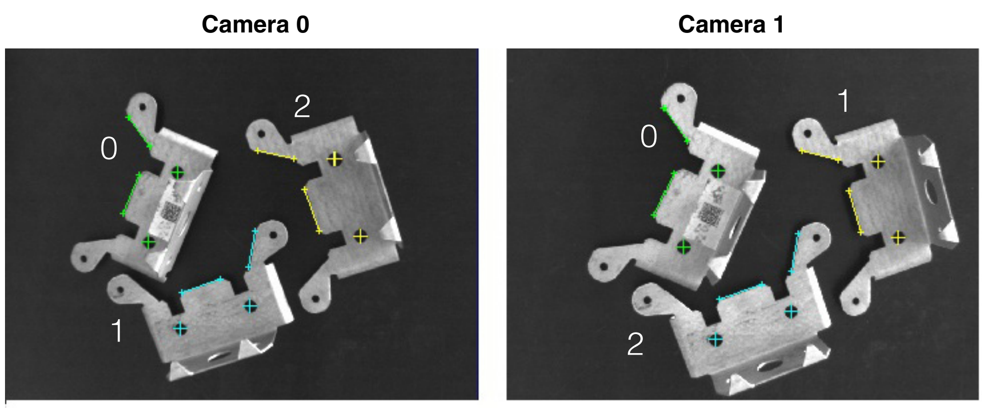
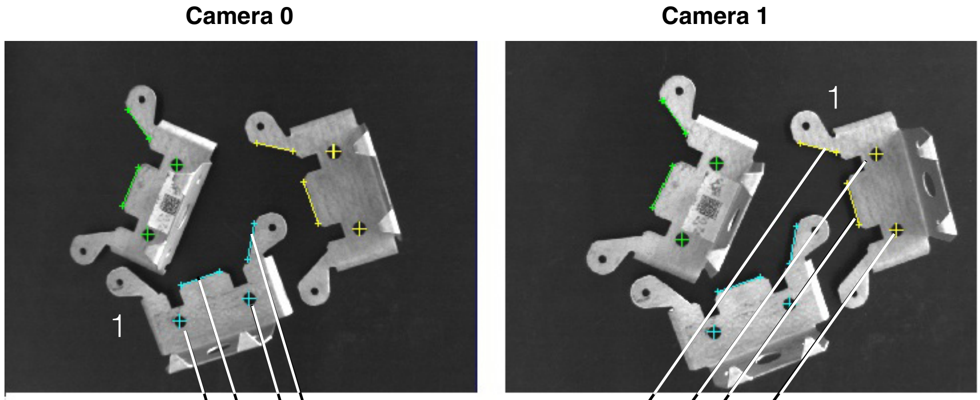


Figure 53. Mismatched PartInstanceIndex properties

For this image set, Figure 54 illustrates some of the **Cog3DCrsp2D3D** objects for **PartInstanceIndex** with a value of 1 and how the RawFeature2D properties refer to different part instances:



CameraIndex	PartInstanceIndex	FeatureRaw2D	FeatureModel3DIndex	FeatureModel3DType	Subfeature
0	1		0	Cog3DLineSeg	StraightEdge0
0	1		1	Cog3DVect3	Point0
0	1		2	Cog3DLineSeg	StraightEdge0
0	1		3	Cog3DVect3	Point0

CameraIndex	PartInstanceIndex	FeatureRaw2D	FeatureModel3DIndex	FeatureModel3DType	Subfeature
1	1		0	Cog3DLineSeg	StraightEdge0
1	1		1	Cog3DVect3	Point0
1	1		2	Cog3DLineSeg	StraightEdge0
1	1		3	Cog3DVect3	Point0

Figure 54. Cog3DCrsp2D3D objects across an image set

Non-Unified and Unified

Any list of **Cog3DCrsp2D3D** objects where the **PartInstanceIndex** values for the same part are not consistent across the images of the image set is *non-unified*, and cannot be used for 3D pose estimation.

To generate a set of *unified* **Cog3DCrsp2D3D** objects, the VisionPro 3D API supports the *part corresponder* **Cog3DPartCorresponderUsing2DPoses** class and **Cog3DPartCorresponderUsingCrsp2D3Ds** class, both of which generate a new list of unified **Cog3DCrsp2D3D** objects suitable for 3D pose estimation.

Using either the **Cog3DPartCorresponderUsing2DPoses** or **Cog3DPartCorresponderUsingCrsp2D3Ds** class requires that you use the saved calibration data generated earlier. See chapter 3, *3D Calibration Tools*, for details on 3D calibration. In addition, the part corresponder classes requires the defined 3D model you created at setup time. See the section *3D Models* on page 96 for details on creating a 3D model.

Use the **Execute** method of **Cog3DPartCorresponderUsing2DPoses** class to generate the part correspondences among different cameras. Then use the **UnifyCrsp2D3D** method to generate a list of unified **Cog3DCrsp2D3D** objects. You can also use the **ConvertToResults** method to generate a list of **Cog3DPartCorresponderUsingCrsp2D3DsResult** objects.

Alternatively, use the **CorrespondPartsUsingPointsAndIntersectionPoints()** method of the **Cog3DPartCorresponderUsingCrsp2D3Ds** class to generate a list of unified **Cog3DCrsp2D3D** objects and a list of **Cog3DPartCorresponderUsingCrsp2D3DsResult** objects, which contains more information about the correspondence result. See the VisionPro 3D Class Reference for more information.

Be aware that for applications where there is only one part in the scene or during 3D model feature generation where there is also one part in the scene, the PartInstanceIndex for all the **Cog3DCrsp2D3D** objects will be 0. Since these are unified **Cog3DCrsp2D3D** objects, there is no need to run the part corresponder.

The sample code installed in %VPRO_ROOT%*samples3D\Programming\Runtime\PoseEstimation* includes Visual Studio 2010 solutions for 3D pose estimation using part correspondence. Copy the contents to a folder where you have write permission before you execute them.

Cog3DPartCorresponderUsing2DPoses

The part corresponder **Cog3DPartCorresponderUsing2DPoses** class requires the 2D part locating results and 3D pose of the part at setup time (generating the part corresponder), and requires the 2D part locating results at run time (running **Execute** method). The 2D part locating results can be obtained by using the 2D part locating tools (such as a PMAlign tool). See the VisionPro online documentation for more information about the part corresponder **Cog3DPartCorresponderUsing2DPoses** class.

Part Correspondence Method of Cog3DPartCorresponderUsingCrsp2D3Ds

If there are 3D lines or 3D line segments in the 3D model, the **CorrespondPartsUsingPointsAndIntersectionPoints()** method internally computes the derived 3D model points from 3D line intersections, and the corresponding derived 2D points from 2D line intersections.

In order for a part instance to be corresponded, there must be at least two cameras seeing that part instance, and each camera sees at least three 2D points (including the derived 2D points corresponding to the derived 3D model points).

If this does not happen, then the part instance will not be corresponded.

Be aware that **FeaturesModel3D** and **Cog3DCrsp2D3Ds[i].FeatureModel3DIndex** for any *i* must be set correctly in order to use the **Cog3DPartCorresponderUsingCrsp2D3Ds** class. **Cog3DModelFeatureGeneratorUsing2D3Ds** might be used to generate 3D model features at setup time.

Outside the Field of View

if your application needs to generate the 3D pose of parts that may or may not be in the field of view for all cameras, you can also use the part corresponder class to generate unified **Cog3DCrsp2D3D** objects.

For example, Figure 55 shows a scene where six parts are presented to two cameras, Camera 0 and Camera 1, but because of the position of the parts as well as the location of the cameras, some of the parts are not in the field of view of both cameras.

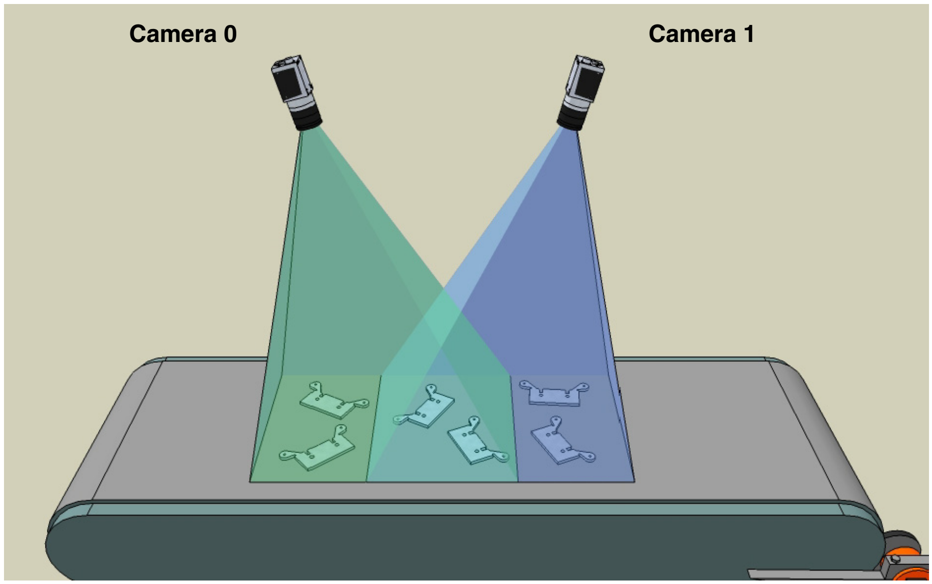


Figure 55. Parts Out of the Field of View

Without using part correspondence, the **PartInstanceIndex** values for the parts across the scene may not correlate, as shown in Figure 56:

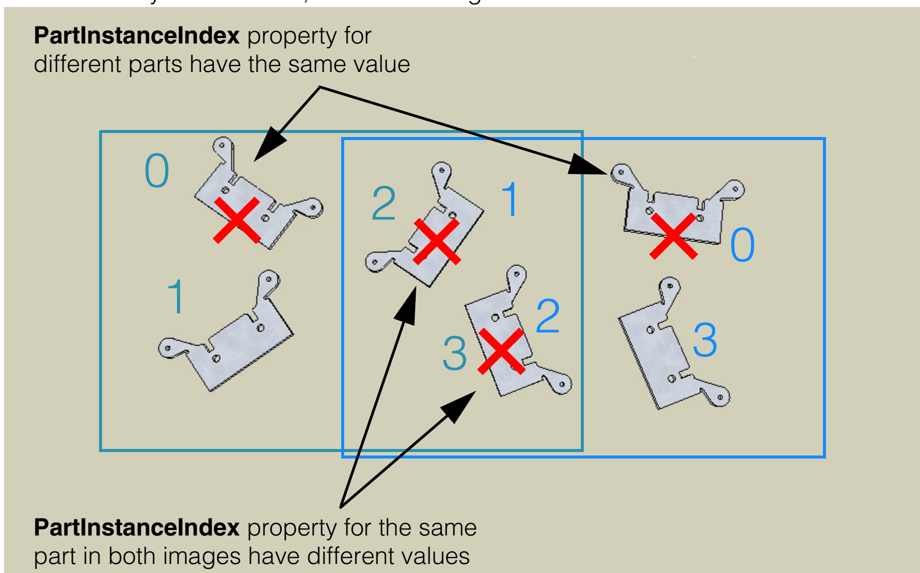


Figure 56. Non-unified PartInstanceIndex values

After generating **Cog3DCrsp2D3D** objects for all the features in all the part instances of both images, part correspondence can be used to assign a consistent **PartInstanceIndex** property for the part instances seen in both images as well as the parts seen by only one camera.

Figure 57 shows the unified **Cog3DCrsp2D3D** objects output by the part corresponder:

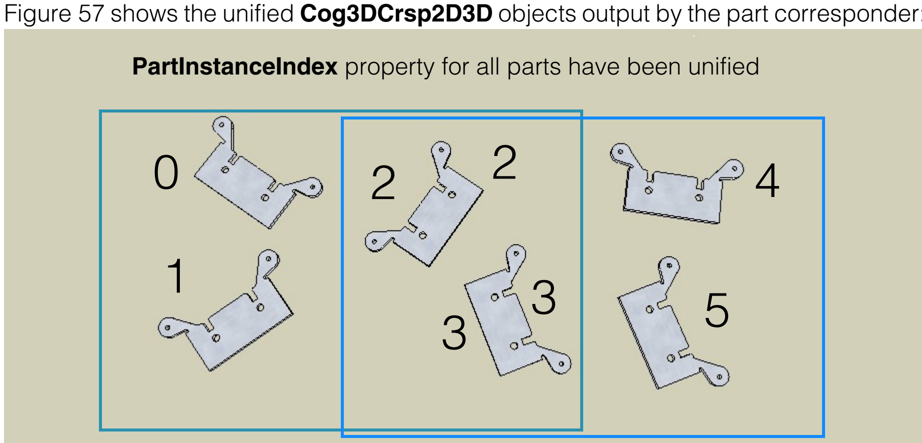


Figure 57. Unified Cog3DCrsp2D3D across an image set

The list of unified **Cog3DCrsp2D3D** objects can now be used for 3D pose estimation for all the part instances.

Coverage Property

The **Cog3DPartCorresponderUsingCrsp2D3DsResult** result object supports the **Coverage** property, which can be used to verify whether all the cameras can see every part instance.

A **Coverage** value of 1.0 indicates that all cameras can see all 2D point features of the part. See the VisionPro online documentation for more details.

3D Pose Estimation

VisionPro 3D-Locate provides a tool with several methods that estimate the 3D pose of parts given a List<**Cog3DCrsp2D3D**>, a set of camera calibrations and a 3D model of the part.

Use the methods and properties of the **Cog3DPoseEstimatorUsingCrsps2D3Ds** class to generate 3D pose estimates for a single part. VisionPro 3D-Locate performs 3D pose estimation by determining the best fit between a set of corresponded 2D image features and a 3D model. The pose in each pose result maps 3D model features from Model3D space to Phys3D space. A list of pose results is returned in order to handle the situation where there are multiple pose estimations for the available 2D features.

To use a 3D pose estimation method your application must have the following data available:

- A set of saved camera calibration data

  See chapter 3, *3D Calibration Tools*, for details on how to generate calibration data for the cameras you use.

- 3D model features that define the model in Model3D space

  3D pose estimation requires a minimum number of 3D model features. The minimum number depends on the types of the 3D model features. See the *VisionPro 3D Class Reference* for details.

  See the section *3D Models* on page 96 for details on how to generate a 3D model.

- A set of unified **Cog3DCrsp2D3D** objects associate 2D features found in an image set and the 3D model features

  See the section *2D Image Feature to 3D Model Feature Correspondence* on page 89 for details on how to construct **Cog3DCrsp2D3D** objects based on the image data from your 3D calibrated cameras.

  See the section *Part Correspondence* on page 103 for details on how to ensure unified **Cog3DCrsp2D3D** objects for scenes with multiple parts.

- A part instance index identifying the part to compute the pose estimation on.

The sample code installed in %VPRO_ROOT%*samples3D\Programming\Runtime\PoseEstimation* includes Visual Studio 2010 solutions for generating a 3D pose estimation for an image set with a single instance of a part and for an image set with multiple parts. Copy the contents to a folder where you have write permission before you execute them.

Pose Estimation Strategies

The **Cog3DPoseEstimatorUsingCrsps2D3Ds** class supports methods for three types of pose estimation strategies:

- Perform 3D pose estimation using all the available **Cog3DCrsp2D3D** feature correspondence objects.
- Perform 3D pose estimation using **Cog3DCrsp2D3D** feature correspondence objects and a set of *robust estimation parameters*, which can exclude outliers in the List<**Cog3DCrsp2D3D**> from being considered during the pose estimation.
- Perform 3D pose estimation using an intial pose estimate and all available **Cog3DCrsp2D3D** feature correspondence objects. This method is suitable for applications with a small range of pose uncertainty. See the section *Refining an Initial Pose Strategy* on page 114 for details.

Each method performs 3D pose estimation for a single part instance in your image set. If you are capturing image sets of multiple parts, you must create a set of unified **Cog3DCrsp2D3D** objects using the **Cog3DPartCorresponderUsing2DPoses** class or the **Cog3DPartCorresponderUsingCrsp2D3Ds** class.

To generate a 3D pose estimate for multiple part instances, you call the desired estimation method for each corresponded part by specifying a part instance index, as shown in the following algorithm:

```
For PartInstanceIndex=0 to CorrespondedResults.Count
  {
    If Using Robust Parameters
    {
      Set Robust Estimation Parameters
      Use Robust Pose Estimator for Part[PartInstanceIndex]
    } else
    {
      Use All Features Pose Estimator for Part[PartInstanceIndex]
    }
  }
```

Using All Features Strategy

Use the **EstimatePoseUsingAllCrsp2D3Ds( )** method to perform 3D pose estimation based on the entire set of **Cog3DCrsp2D3D** objects you pass as an input parameter. The method returns a **Cog3DPoseEstimatorUsingCrsp2D3DsResult** object which contains a list of 3D pose results as well as other data. See the section *Pose Estimation Results* on page 113 for details.

In general, the **EstimatePoseUsingAllCrsp2D3Ds( )** method offers the faster execution speed than the **EstimatePoseUsingInlierCrsp2D3Ds()** method. Use this method in applications where you are confident about the accuracy of your 2D features.

Using Robust Parameters Strategy

Use the **EstimatePoseUsingInlierCrsp2D3Ds( )** method to perform 3D pose estimation using the **Cog3DCrsp2D3D** objects you pass as an input parameter and an additional parameter of type **Cog3DRobustPoseEstimationParametersSimple**, which specifies the robust pose estimation parameters that can be used to exclude one or more **Cog3DCrsp2D3D** objects from being considered. **Cog3DCrsp2D3D** objects that are excluded are recorded as *outliers*, while **Cog3DCrsp2D3D** objects that are included in the 3D pose estimation are *inliers*.

You must explicitly set the properties of the **Cog3DRobustPoseEstimationParametersSimple** object before calling the **EstimatePoseUsingInlierCrsp2D3Ds( )** method. Specifically, Table 4 lists the four properties that determine the robust pose estimation parameters:

Property	Description
ResidualsPhys3DMaxThreshold	Sets a threshold that specifies the maximum allowed distance between the ray of any 2D feature and its corresponding mapped 3D model feature. If the threshold is exceeded, then the mapped **Cog3DCrsp2D3D** object is considered an outlier.
ResidualsPhys3DRmsThreshold	Sets a threshold that specifies the maximum allowed RMS value of the distances between the rays of 2D features and their corresponding mapped 3D model features. If the threshold is exceeded, then the **Cog3DCrsp2D3D** objects having the largest impact on the RMS residuals are considered outliers.
MinNumOfFeaturesModel3DFromAtLeast2Cameras	Sets the minimum number of 3D model features from at least 2 cameras to be used for the pose estimation. The accuracy of the pose estimation is improved when 2D features from multiple cameras are used.
MinNumOfFeaturesModel3D	Sets the minimum number of 3D model features to be used for the pose estimation.

Table 4. Robust Pose Estimation Parameters

The method returns a **Cog3DPoseEstimatorUsingCrsp2D3DsResult** object which contains a list of 3D pose results as well as other data. See the section *Pose Estimation Results* on page 113 for details.

Pose Estimation Results

The 3D pose estimation methods described in *Pose Estimation Strategies* on page 111 return an obect of type **Cog3DPoseEstimatorUsingCrsp2D3DsResult**, which describes the results of the 3D pose estimation.

Use the following methods and properties in your application to access the 3D pose estimation results:

- **GetPoseResults( )** method

  This method returns a list of **Cog3DPoseEstimatorUsing2DPointsResult** objects that encapsulate the result of a 3D pose estimation. The number of objects in the list is an indicator of the success of the 3D pose estimation method, and will have one of the following values:

  - 0: No 3D poses were returned that met the 3D pose estimation input parameters.

    The **Message** property of the result will contain diagnostic information about why no 3D poses were found.

  - 1: A single 3D pose was returned that met the 3D pose estimation input parameters.

  - Greater than 1: Multiple 3D poses were returned that met the 3D pose estimation input parameters.

    Adding more unified **Cog3DCrsp2D3D** objects can reduce the number of 3D poses that meet the estimation criteria.

  See the section *Pose Results* on page 114 for information on how to access the 3D pose estimates.

- **GetIndicesOfOutlierCrsp2D3Ds( )** method

  This method returns a list of integers indicating the indices of unified **Cog3DCrsp2D3D** objects that were considered outliers and excluded from the 3D pose estimation.

  This list is always empty if you use the **EstimatePoseUsingAllCrsp2D3Ds( )** method.

- **Message** property

  A string that contains diagnostic information about why the **GetPoseResults()** method returned no 3D pose estimates. This property is NULL when the **GetPoseResults()** method returns at least one 3D pose result.

- **PartInstanceIndex** property

  An integer value which is a copy of the **PartInstanceIndex** parameter

Pose Results

The **GetPoseResults( )** method of the **Cog3DPoseEstimatorUsingCrsp2D3DsResult** object returns a list of **Cog3DPoseEstimatorUsing2DPointsResult** objects. See the section *Pose Estimation Results* on page 113 for a description of the results returned by this method. Ideally the list contains a single item.

A **Cog3DPoseEstimatorUsing2DPointsResult** object supports the following properties to describe the 3D pose of the object in Phys3D space:

- **Phys3DFromModel3D**: A **Cog3DTransformRigid** object to describe the 3D pose of the part in Phys3D space.
- **ResidualsPhys3D**: A **Cog3DResiduals** to describe the 3D residuals in Phys3D space of this result.

  Residual error in Phys3D space is the difference between a ray of any 2D feature and its corresponding mapped 3D model feature.

- **ResidualsRaw2D**: A **Cog3DResiduals** to describe the 2D residuals in Raw2D space of this result.

  Residual error in Raw2D space is the difference between the found location of the 2D features in the image and the 2D locations that you would expect if you took the 3D model features and mapped them through the return pose from Phys3D to Raw2D space using the camera calibration data.

Refining an Initial Pose Strategy

The **Cog3DPoseEstimatorUsingCrsps2D3Ds** class supports an additional method that allows you to refine an existing 3D pose to perform a 3D pose estimation.

The **RefinePoseUsingAllCrsp2D3Ds()** method requires a **Cog3DTransformRigid** object that you supply as an initial pose. In addition, the method uses all the unified **Cog3DCrsp2D3D** objects and does not exclude any as outliers.

If the part pose uncertainty range is small for your application, you can use this method to get a more accurate 3D pose estimation by providing a coarse pose (for example, using an identity pose as the coarse pose if the part's nominal pose is identity).