Cognex 3D-Locate

Developer's Guide

CVL 8.0

June 2016

Contents

Contents

Contents

Preface

This manual describes how Cognex vision software tools work and how you use them to solve vision applications. For specific programming information, refer to the programming reference documentation for your framework.

Style Conventions Used in This Manual

This manual uses the following style conventions for text:

boldface	Used for C/C++ 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 C/C++ 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 CVL software runs on 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 *Getting Started* manual for your CVL release for details on the operating systems, hardware, and software supported by that release.

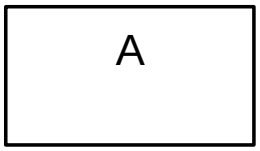
Software Diagramming Conventions

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{};
  ```

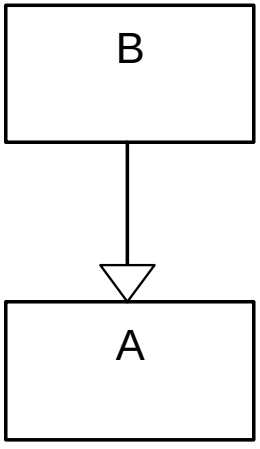
  is shown graphically as follows:



- **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 {};
  ```

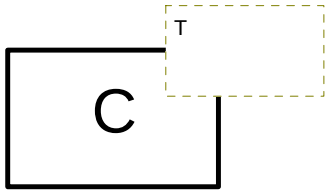
  is shown graphically as follows:



- **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 Cognex 3D Vision tools and capabilities are provided in the following chapters:

- *3D Vision Framework* 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 CVL, including specific recipes and procedures for calibrating a 3D machine vision system.
- *3D Calibration Wizard* describes the application you can use to generate a 3D calibration object.
- *Locating Objects in 3D* describes the triangulation and fitting tools that enable the generation of 3D poses for objects.
- *3D Application Development Guide* provides information that you can use to evaluate potential 3D vision applications, and specific development approaches that you can use.

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

CVL support three-dimensional machine vision tools that take the information from two-dimensional images and generate information about objects in three-dimensional space.

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 25 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 30 is a specialized calibration capability intended for use with robots and robot-mounted machine vision cameras.

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 **cc3DCameraCalib.**

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.

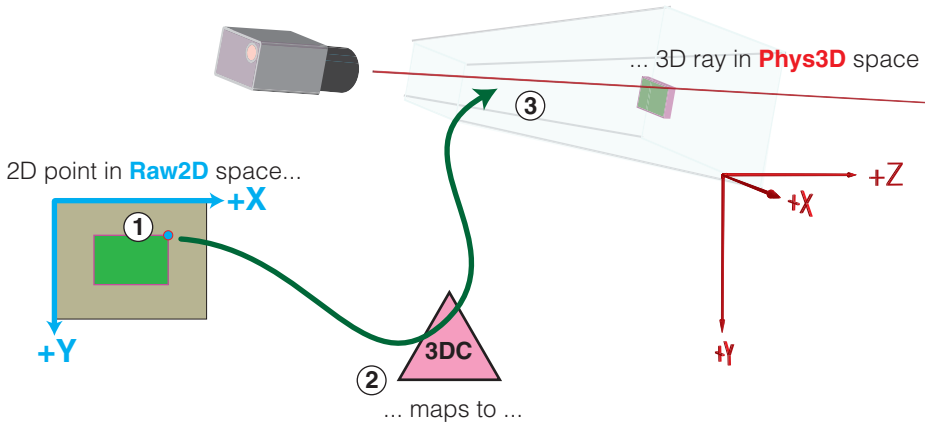


Figure 1. Transforming a 2D point from Raw2D space into a 3D ray in Phys3D space

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.

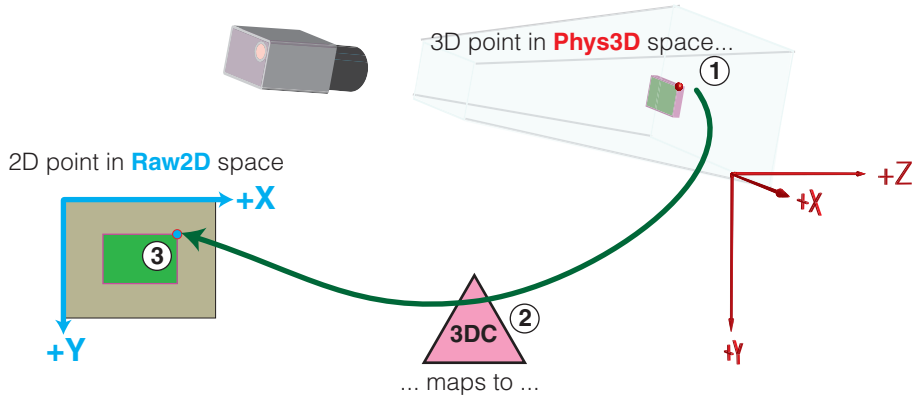


Figure 2. 3D calibration maps 3D points in physical space to 2D points in an image

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 22.

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.

Note Using the standard CVL calibration plate feature extractor, described in the *Calibration* chapter of the *CVL User's Guide*, you construct a list of correspondence pairs giving the locations of the plate vertices in both image coordinates and physical coordinates. Alternatively, you can generate the correspondence pairs manually.

Calibration Plate Requirements

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

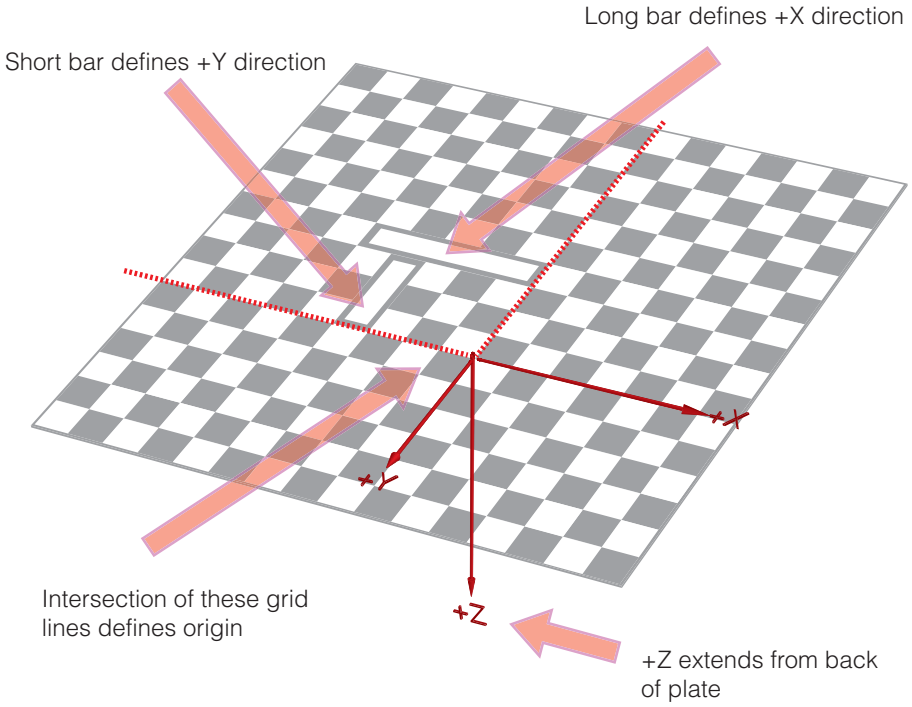


Figure 3. Calibration plate fiducial mark

Plate Pose Requirements

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.

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 4).

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

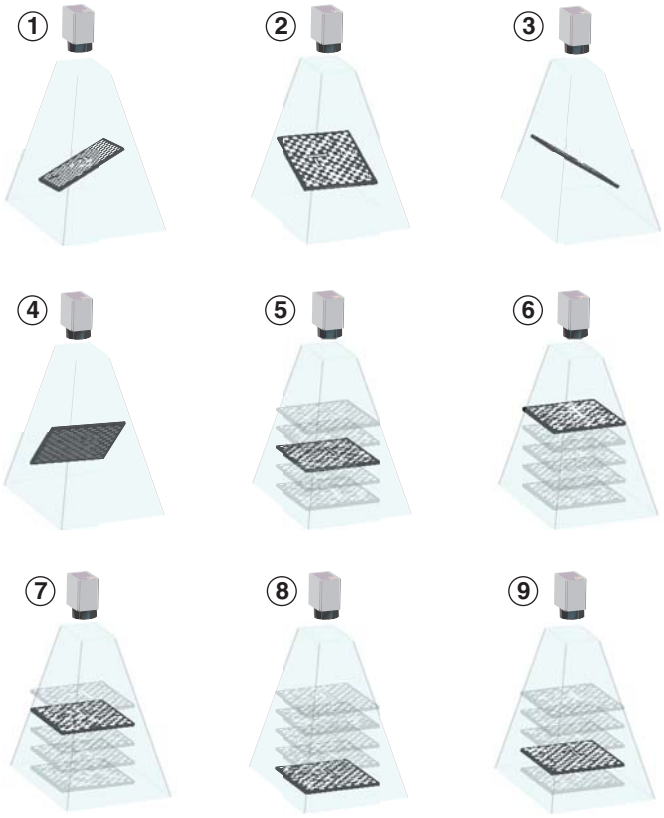


Figure 4. Required (1-4) and recommended (5-9) plate views for 3D calibration

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 3D calibration.

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 **cc3DCameraCalib** 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 5 shows Raw2D space, Phys3D space, and the Raw2DFromPhys3D transformation that links the spaces.

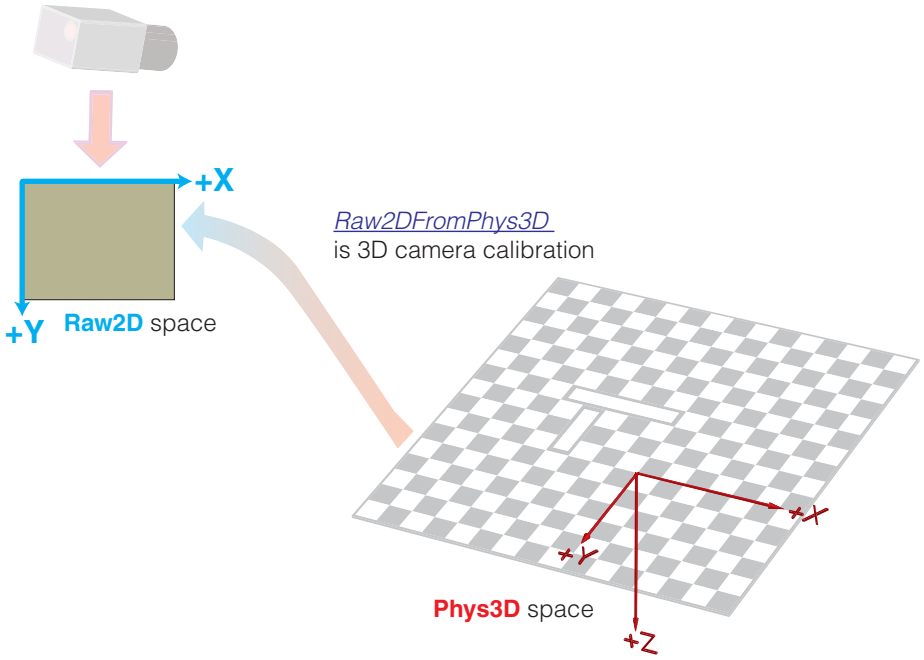


Figure 5. Raw 2D and Physical 3D space

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 18.

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. 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:

- **cc3DCameraCalib::raw2DFromCamera2D**

  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.

- **cc3DCameraCalib::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 6 shows all four of the spaces, and their associated transformations, generated during 3D calibration.

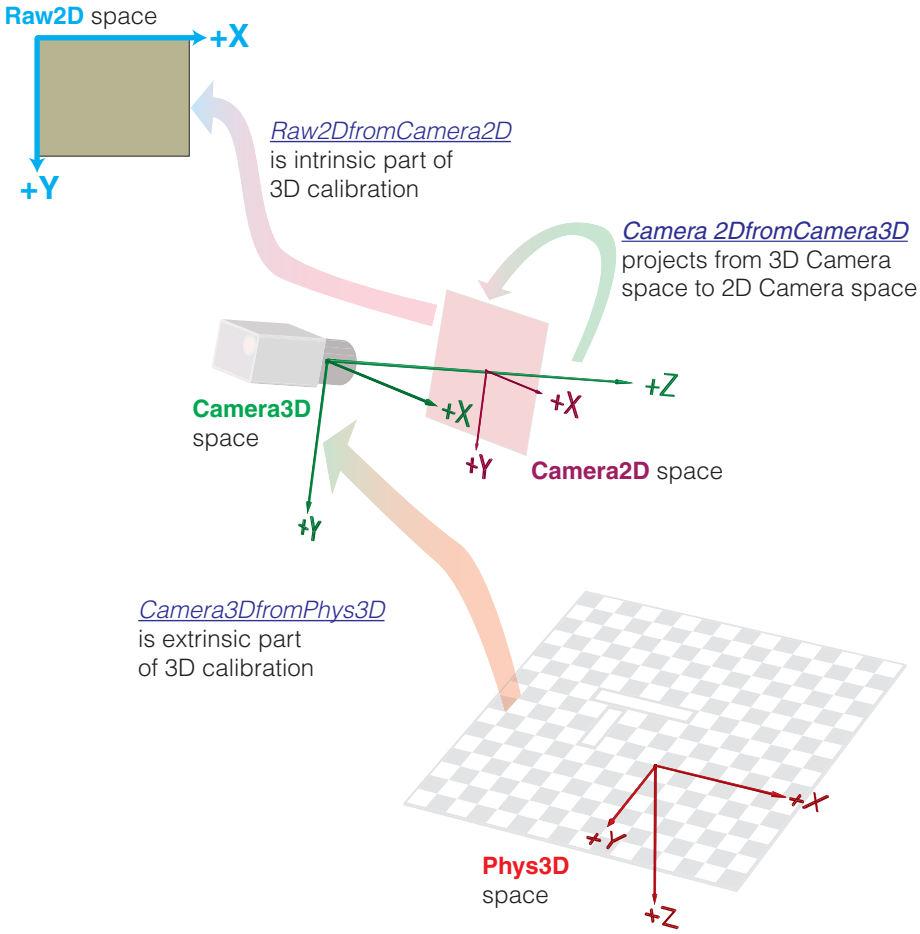


Figure 6. Spaces and transformations computed by 3D calibration

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 7, and a separate calibration object is computed for each camera.

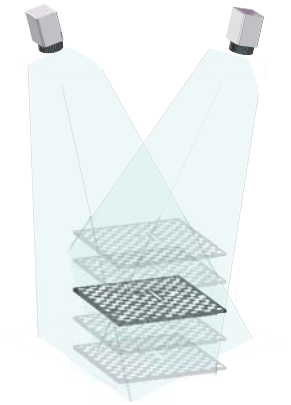


Figure 7. Multi-camera calibration

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

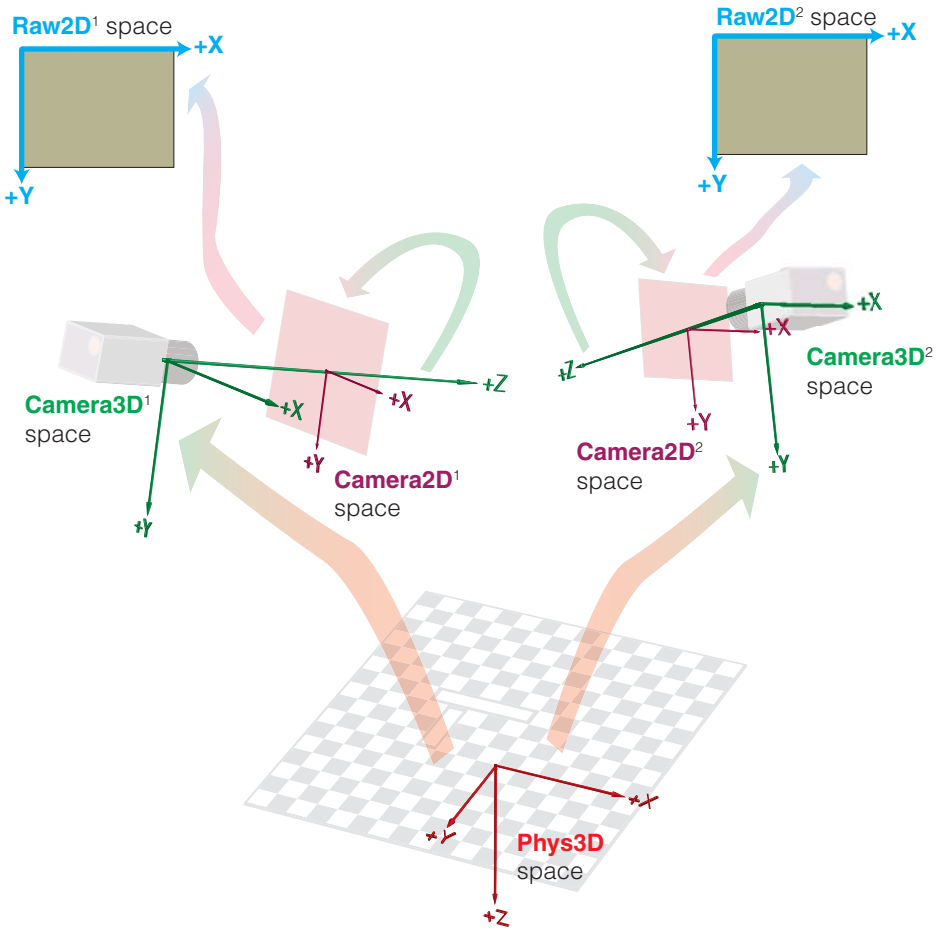


Figure 8. 3D camera calibrations from multiple cameras share common 3D physical space

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

Figure 9. 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 9, the application could use a corner on the lower edge of the end of the part, as shown in Figure 10.

Figure 10. 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 **cf3DTriangulatePointPhys3DUsingPointsRaw2D()** function providing the following inputs:

- The two feature locations in Raw2D space
- The two **cc3DCameraCalib** objects produced during 3D calibration

The function returns the 3D location of the feature in Phys3D space. Figure 11 shows the triangulation process.

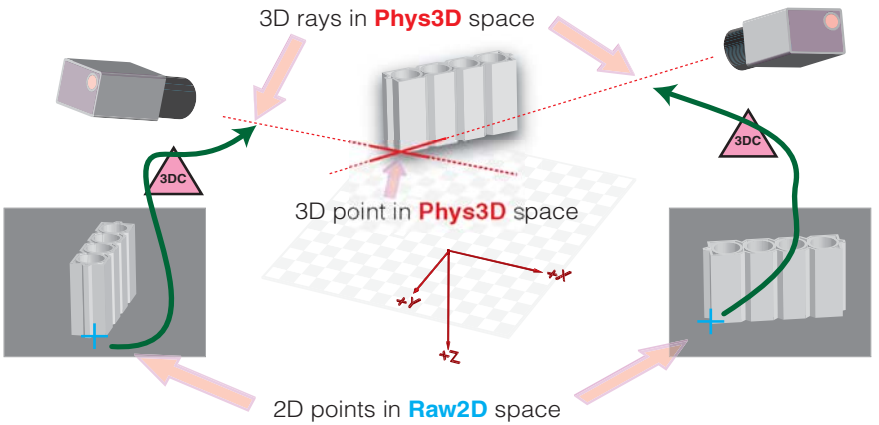


Figure 11. Triangulating a 3D point from two 2D points from two 3D-calibrated cameras

Estimating 3D Object Poses

While the triangulation tool can provide 3D locations of points, many 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 triangulation tool to determine the 3D pose of an object:

- You have numerical data or measurements that describe the 3D locations of features of your object (such as a CAD description).
- The CAD model points must be expressed in a single 3D coordinate space, the origin of which is determined by the points themselves. This space is referred to as *Model3D space*.
- The numerical data provides the positions of features which can be located by 2D vision tools in an acquired image.
- There exist groups of at least three non-collinear features which can be located in multiple images from separated, 3D-calibrated cameras.

For example, if you had an accurate CAD model of the part shown in Figure 9, it might include the positions of the eight outermost corners of the part, as shown in Figure 12.

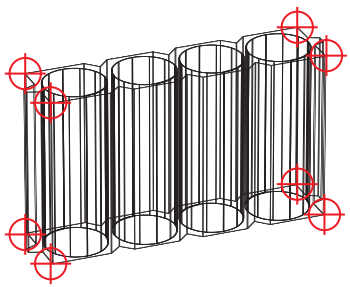


Figure 12. CAD model points

The first step is to acquire images of the part from multiple 3D-calibrated cameras and locate as many of the CAD features as possible. As shown in Figure 12, the same 6 (out of a possible 8) CAD features are visible in both of the camera's views.

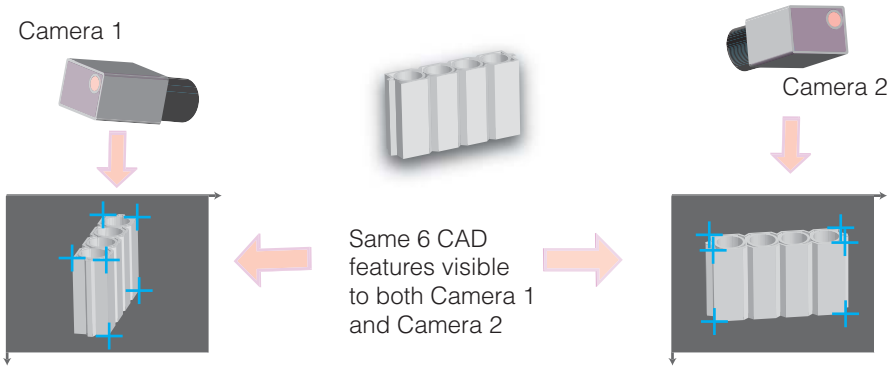


Figure 13. Locating CAD features

The second step is to determine the correspondence between the features in the two images' Raw2D spaces. Figure 14 labels the six 2D points from the two images so that point 1 in both images refers to the same CAD feature location.

Once you have established the correspondence among the multiple sets of image points, the application calls the triangulation tool's **cf3DComputePhys3DFromModel3DUsingPointsRaw2D()** function providing the following inputs:

- The two vectors of 2D feature locations in Raw2D space, ordered so that the locations in the two vectors refer to the same 3D feature
- The 3D feature locations in Model3D space, in the same order as the two sets of 2D features
- The two **cc3DCameraCalib** objects associated with the 3D-calibrated cameras

As shown in Figure 14, the triangulation tool computes the 3D pose of the object's Model3D coordinate space (as defined by the 3D points that you supply) in Phys3D space.

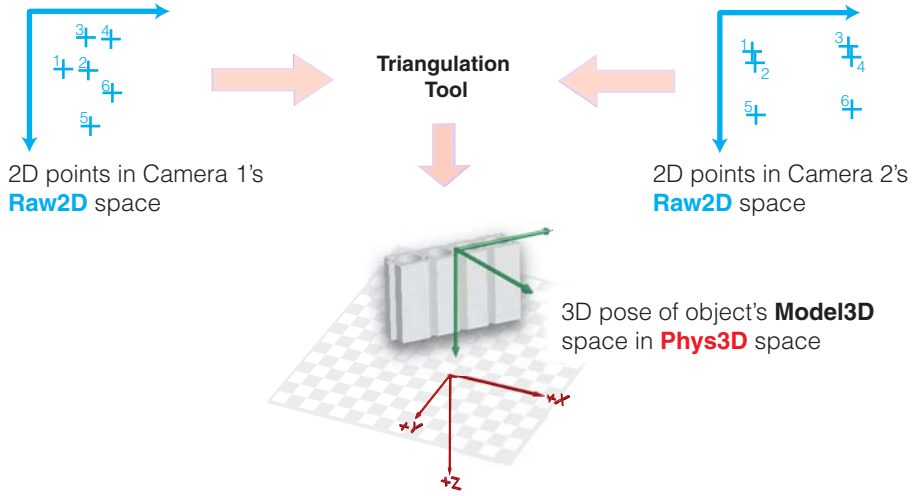


Figure 14. Direct pose estimation from 2D points using triangulation

Note The three point sets that the tool uses (the two sets of 2D points in Raw2D space from the acquired images and the single set of 3D feature locations that define the object in Model3D space) *must* be supplied in the same order, so that a given index refers to the same physical point in each point set.

Single-Camera 3D Pose Estimation

The triangulation tool also lets you directly estimate a 3D pose from a single camera and 2D point set using the same technique as described in the previous section. Single-camera pose estimation is less reliable and less accurate, and it requires the following:

- At least three of the points in Model3D space must be non-collinear
- The 2D points must be widely spaced in the image

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 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).

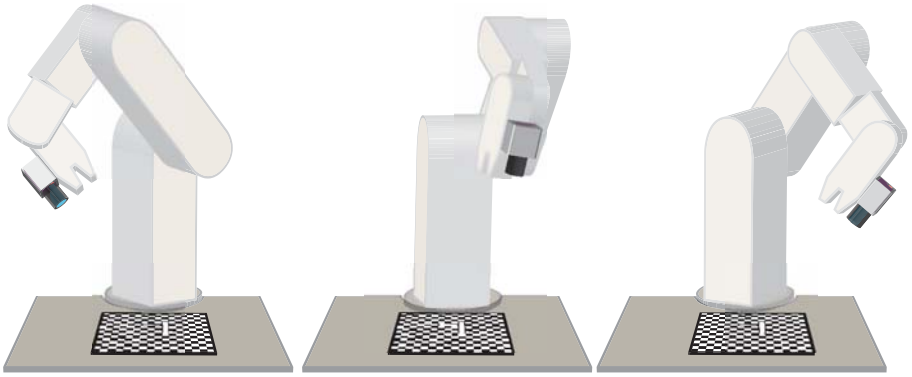


Figure 15. Hand-eye calibration

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 16 shows the inputs from a single station.

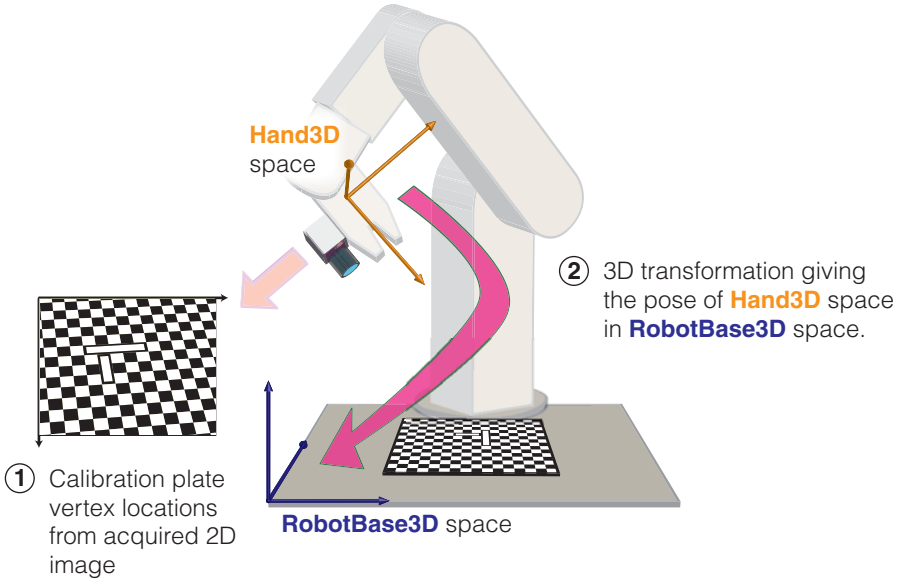


Figure 16. Per-station inputs for hand-eye calibration

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 17.

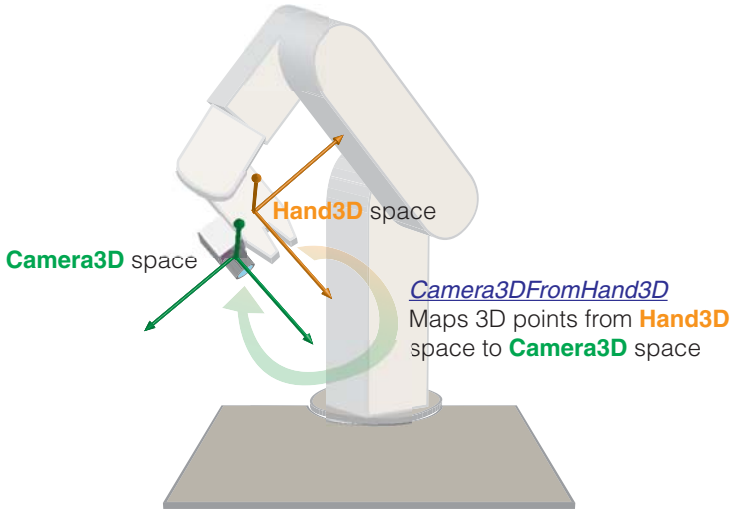


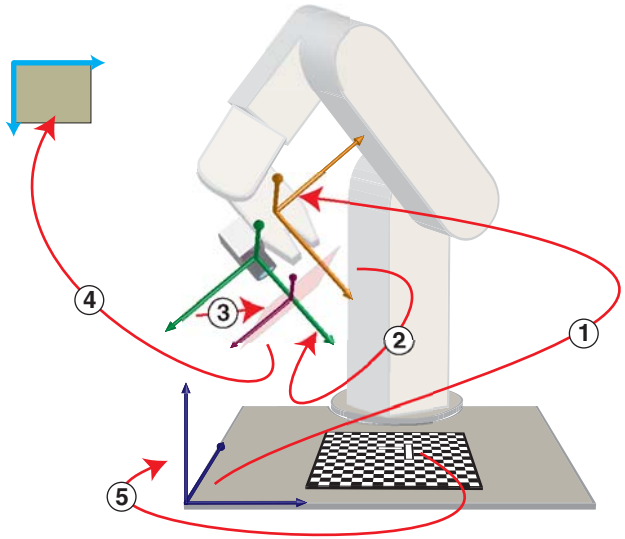
Figure 17. Hand-eye calibration outputs

Hand-eye calibration also computes a standard 3D camera calibration during hand-eye calibration. A separate 3D calibration object is produced for each hand-eye calibration station.

The intrinsic part of the 3D calibration (the transformations between Camera3D, Camera2D, and Raw2D spaces) are the same for every station -- they are computed simultaneously using all of the separate station views of the same calibration plate, just as a standard 3D camera calibration is computed using multiple views of a calibration plate.

The extrinsic part of the 3D calibration (the transformation between Phys3D and Camera3D) reflects the position of the camera with respect to the calibration plate when that station's calibration image was acquired. This may be useful if you fixed the calibration plate at a location that is useful in your application.

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 18.



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 18. Spaces and transformations associated with hand-eye calibration

3D Vision Framework 2

This chapter describes the basic 3D-Locate building blocks that provide the mathematical foundation for your 3D applications. The 3D-Locate 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 36, provide an overview of the chapter and define some terms that you will encounter as you read.
- *3D Shapes* on page 37 describes the 3D shape classes and interfaces, and also describes the shape projection functions.
- *3D Transformations* on page 41 provides information on the classes that implement 3D rigid transformations.

For additional information on the 3D Vision Framework, examine the header files and sample code files listed in the sections *CVL Header Files and Sample Code* on page 44.

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.

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.

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

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 common 3D shapes (line, plane, circle, rectangle, box, and sphere).
- 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 pointer. 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 **cc3DShape** base class, and most also inherit from one or more shape type classes for different shape types:

- **cc3DVertex** provides functionality for shapes that can be described as a set of vertices.
- **cc3DCurve** provides functionality for shapes that can be described by 3D line segments, arcs, and other 1D shapes.
- **cc3DSurface** provides functionality for shapes that can be described by surfaces.
- **cc3DVolume** provides functionality for shapes that can be described as a volume or collection of volumes.

For example, Figure 19 shows the inheritance diagram for a concrete class, **cc3DRect**.

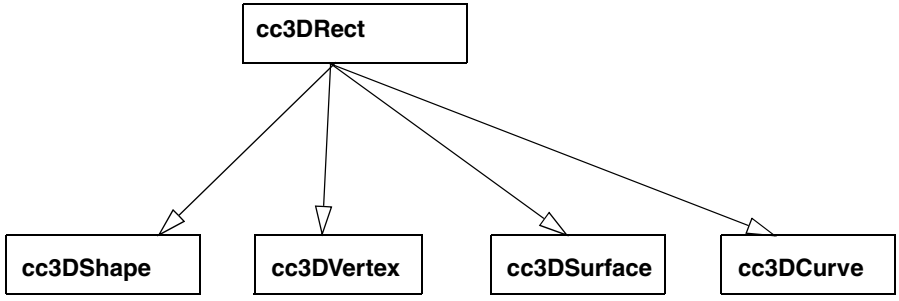


Figure 19. cc3DRect inheritance

cc3DRect represents a 3D rectangle, which has vertices (the four corners), curves (the four sides), and surfaces (the top and bottom surface). Contrast this with **cc3DSphere**, which has a surface but no vertices or curves.

Figure 20 shows the full class hierarchy for CVL 3D shapes.

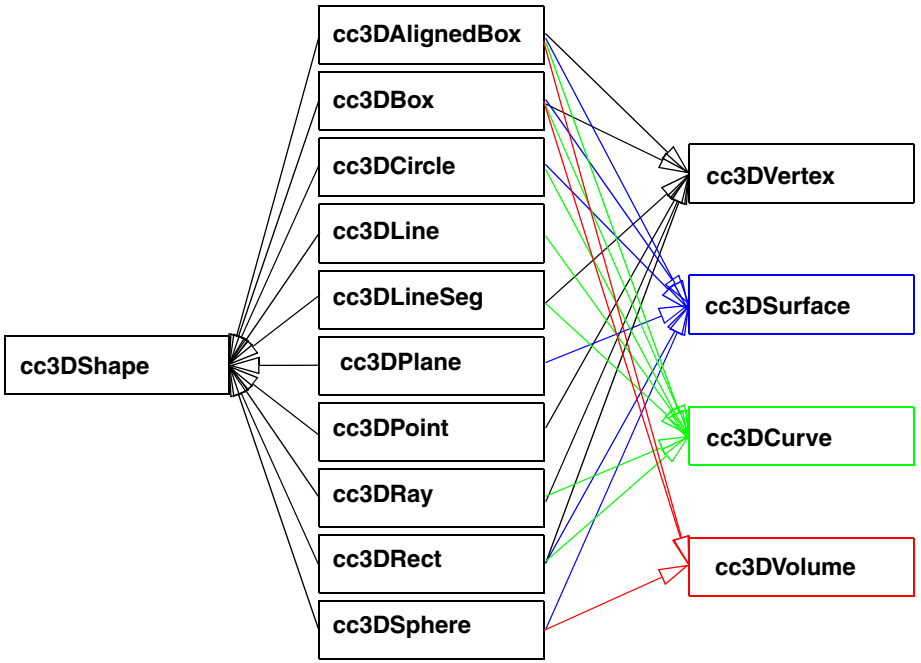


Figure 20. CVL 3D shapes class hierarchy

3D Shape State Type

All concrete shape types that are derived from **cc3DShape** must implement the **stateType()** member function, which allows you to set and get the state type for the particular shape. The 3D shape framework defines four state types that correspond to the four shape type base classes:

Shape type class	State type enumeration
cc3DVertex	*cc3DShapeDefs::eVertex*
cc3DSurface	*cc3DShapeDefs::eSurface*
cc3DCurve	*cc3DShapeDefs::eCurve*
cc3DVolume	*cc3DShapeDefs::eVolume*

Table 3. State types

A shape's shape type controls how the shape's geometry is interpreted by various members of **cc3DShape**. Figure 21 shows how the state type effects how the distance to shape measurement is made between a point and a **cc3DBox**.

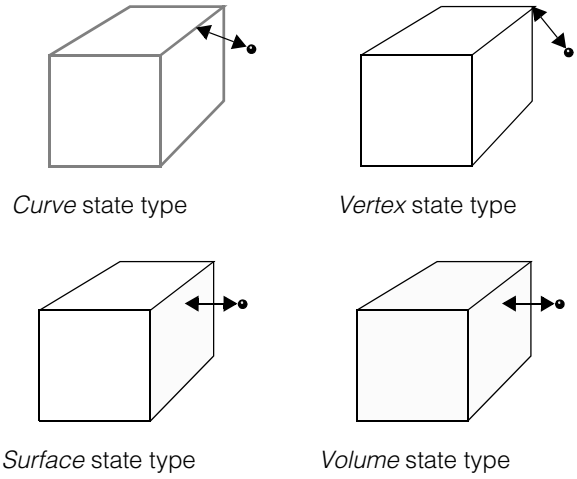


Figure 21. The effect of state type on distance measurements from a point to a cc3DBox

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 47.

The 3D shape projection function, **cf3DProject3DShapeTo2DGraphicList()** generates standard 2D graphics that correspond to the projected 3D shape and appends them to a standard CVL **ccGraphicList** that you can display. The function provides access to the standard **ccGraphicsProps** interface for controlling the graphical appearance of the shapes.

You can also specify two parameters that are specific to projection, the projection accuracy and the projection shape representation.

Projection Accuracy

The projected shape is composed of a collection of individual line segments. Increasing the number of line segments in the projected graphic improves the accuracy of the projection, but requires additional memory and processing time.

You can control the projection accuracy by specifying a *maximum error*, which is the greatest distance, in 2D image space, between the projected graphic and the true path of the projected shape, as shown in Figure 22.

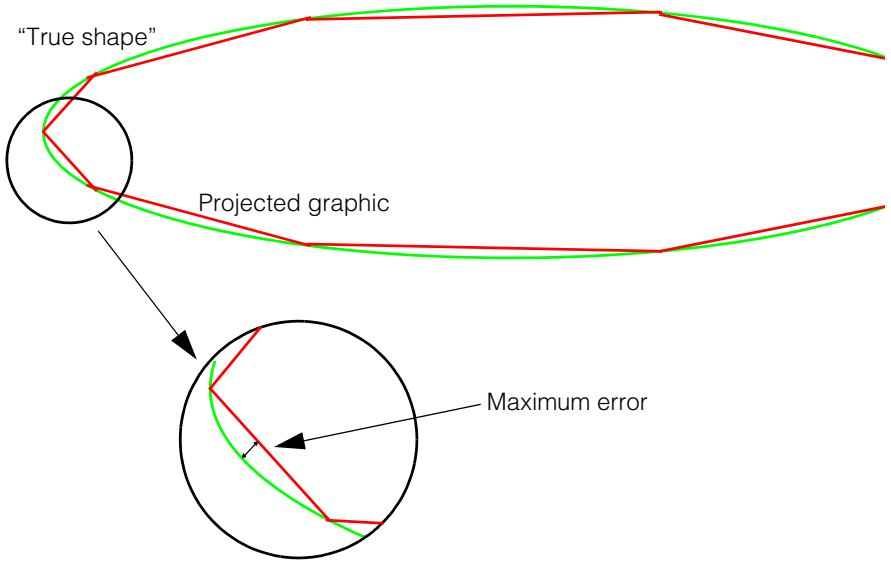


Figure 22. Projection error

Projection Shape Representation

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

3D Transformations

The 3D-Locate 3D vision framework provides a set of classes 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 classes are 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 Transformation Classes

Three classes implement 3D transformation in 3D-Locate.

- **cc3DXformBase** is an abstract base class for 3D transformations.
- **cc3DXformRigid** is a concrete implementation for rigid 3D transformations.
- **cc3DRotation** is a concrete implementations for 3D rotation.

Figure 23 shows the class hierarchy for CVL 3D transformation classes.

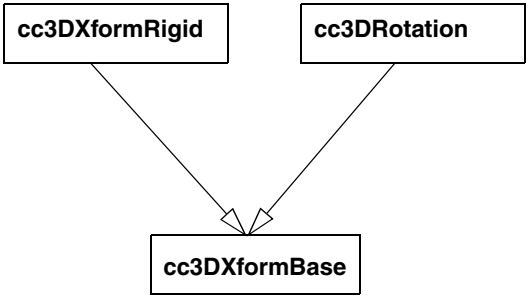


Figure 23. CVL 3D transform inheritance

3D Rigid Transformation

The **cc3DXformRigid** class lets you directly access the translation and rotation components of the transformation through the **cc3DXformRigid::trans()** and **cc3DXformRigid::rotation()** members ,.

trans() returns a **cc3DVect** giving the translation and **rotation()** returns a **cc3DRotation** object. .

A 3D pose is the 3D orientation (rotation) and 3D position (translation) of a 3D coordinate space within another 3D coordinate space. Since this is effectively mapping one coordinate space to another, a **cc3DXformRigid** is also used to represent a 3D pose.

Transformation Names

All 3D-Locate classes, members, arguments, and global functions use the following convention for names that refer to 3D transformations:

```
<dest_space>From<source_space>
```

where *dest_space* is the coordinate space to which the point is mapped and *source_space* is the space from which the point is mapped. For example, a function parameter called *World3DFromModel3D* provides a mapping from the Model3D space to the World3D space.

The same naming convention used for transformations is also used for poses. A pose named *World3DFromModel3D* describes the orientation and position of the Model3D coordinate space (shown in green) with respect to the World3D coordinate space (shown in red), as shown in Figure 24.

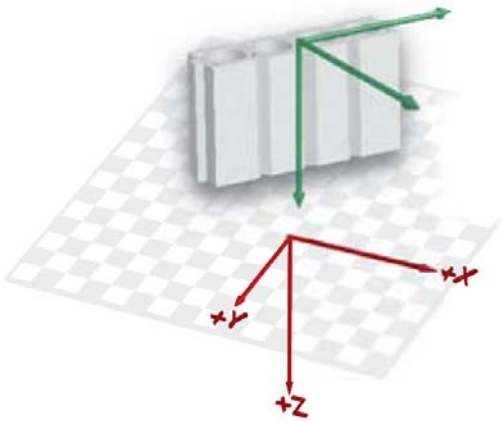


Figure 24. 3D pose

Transformation Order

Whenever you use a **cc3DXformRigid** to map a point from one 3D coordinate space to another, the transformation is performed in the following order:

1. The rotation component (**rotation()**) is applied to the point.
2. The translation component (**trans()**) is applied to the rotated point.

3D Rotation

3D rotations are encapsulated in the **cc3DRotation** class, which allow 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)

The details of the 3D-Locate implementations and interfaces for 3D rotation representations in CVL can be found in the *3D-Locate Class Reference* and in the header files and sample code listed in the section *CVL Header Files and Sample Code* on page 44.

CVL Header Files and Sample Code

This section lists the classes, header files, and sample code that implement the CVL 3D vision framework discussed in this chapter.

3D Shapes

The following table summarizes the classes and header files that support 3D shapes:

C++ Class	Header file
cc3DShape **cc3DLine** **cc3DAlignedBox** **cc3DBox** **cc3DVertex** **cc3DCurve** **cc3DSurface** **cc3DVolume** **cc3DPoint** **cc3DLineSeg** **cc3DRay** **cc3DPlane** **cc3DCircle** **cc3DRect** **cc3DSphere**	*ch_c3d/shapes3d.h*

3D Graphics Projection

The following table summarizes the classes and header files that support 3D graphics projection:

C++ Class	Header file
cc3DShapeProjectParams **cf3DProject3DShapeTo2DGraphicList()** **cf3DProject3DCoordinateAxesTo2DGraphicList()**	ch_c3d/shapproj.h

The sample code file *c3d_runtime.cpp* located in *%VISION_ROOT%\sample\cvl\c3d\basic* shows how to display 3D graphics on an image. See the function **cc3D_Part::draw()**.

3D Transformations

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

Functionality	C++ Class	Header file
3D Rotation	**cc3DRotation**	*ch_c3d/xform3d.h*
3D Rotation – Euler angles	**cc3DEulerZYX**	*ch_c3d/eulerzyx.h*
3D Rotation – quaternions	**cc3DQuaternion**	*ch_c3d/quatern.h*
3D Rotation – axis angle	**cc3DAxisAngle**	*ch_c3d/axisang.h*
3D Rotation – matrix	**cc3DMatrix** (typedef of **cc3Matrix**)	*ch_cvl/matrix.h*
3D Translation	**cc3DVect** (typedef of **cc3Vect**)	*ch_cvl/vector.h*
3D Point	**cc3DVect** (typedef of **cc3Vect**)	*ch_cvl/vector.h*
3D Transform	**cc3DXformRigid**	*ch_c3d/xform3d.h*
3D Pose	**cc3DXformRigid**	*ch_c3d/xform3d.h*

The sample code files *c3d_setup.cpp* and *c3d_runtime.cpp* located in *%VISION_ROOT%\sample\cvl\c3d\basic* make use of 3D transforms. *c3d_runtime.cpp* has several examples computing the pose of parts.

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.

An introduction to 3D calibration can be found in the section *3D Calibration* on page 14. In addition, CVL supports the 3D Calibration Wizard, a graphical interface for configuring 3D cameras, capturing viewsets, and generating the 3D calibration object of type **cc3DCameraCalib**. See chapter *4, 3D Calibration Wizard*, for more information.

This chapter contains the following sections:

- *Some Useful Definitions* on page 48 defines some terms that you will encounter as you read this chapter.
- *3D Calibration Basics* on page 50 provides a review of the coordinate spaces and transformations associated with 3D camera calibration.
- *3D Camera Calibration* on page 53 provides a detailed discussion of the procedure, including Cognex's recommended best practices, for performing 3D camera calibration.
- *Hand-Eye Calibration* on page 64 contains an overview of the coordinate spaces used in hand-eye calibration, and describes the basic techniques for performing hand-eye calibration.

For more information on the 3D calibration tools, examine the header files and sample code files listed in the section *CVL Header Files and Sample Code* on page 72.

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 — CVL 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 18.

Figure 25 shows all four of the spaces, and their associated transformations, generated during 3D camera calibration.

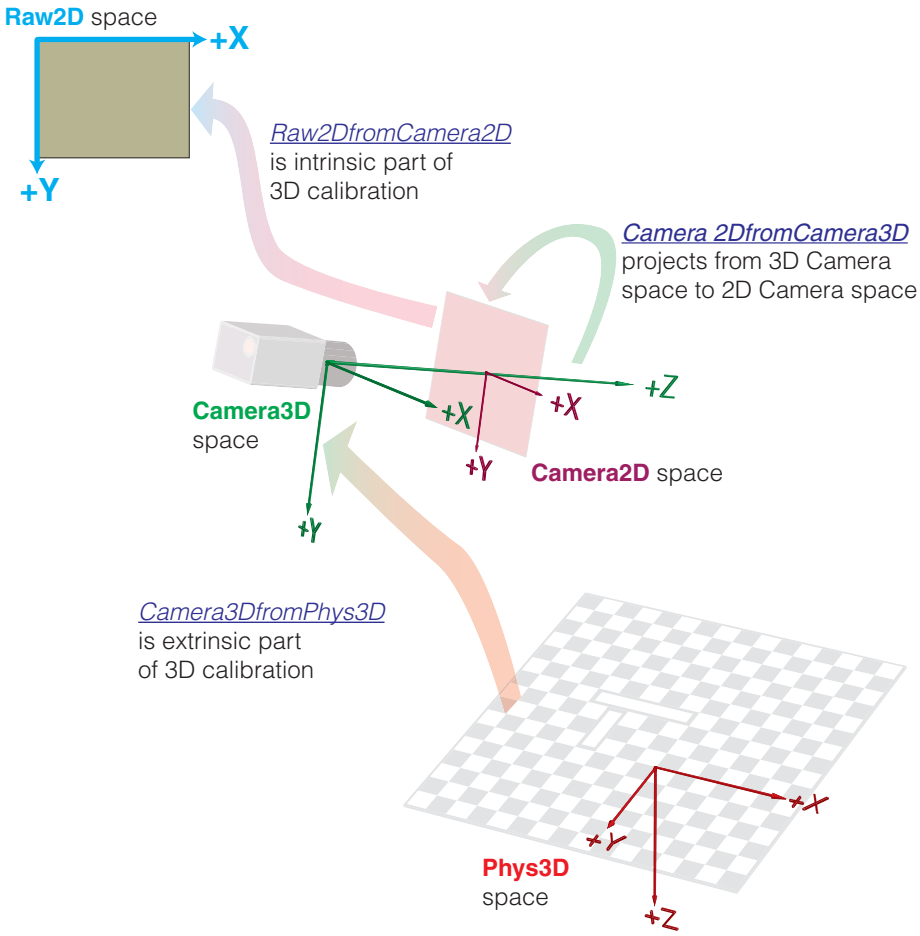


Figure 25. Spaces and transformations computed by 3D camera calibration

Table 4 defines and describes the spaces shown in Figure 25.

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. 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 4. 3D calibration coordinate spaces

It is the first and last spaces shown in Table 4 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

3D camera calibration involves five steps:

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

   See the topic *Camera Positioning* on page 53 for more information.

2. Acquire a series of *viewsets* of the calibration plate, one per camera, acquired at the same time. A viewset contains images from the multiple cameras of the same calibration plate in a given pose. You must acquire a minimum of four viewsets.

   See the topic *Acquiring the Viewsets* on page 54 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 61 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 *cc3DCameraCalibDefs::ePoseTilted* .
- Z-offset elevated viewsets. These viewsets are optional, but acquiring this type of viewset improves calibration accuracy. This pose type is called *cc3DCameraCalibDefs::ePoseElevated*
- 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 *cc3DCameraCalibDefs::ePoseDefineWorldCoord* .

Each typoe 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.

Using a Region of Interest

During 3D camera calibration, you can specify a 2D region of interest for each camera that you are calibrating. The 3D camera calibration tool will only generate calibration for the specified regions. The computed 3D camera calibration retains the region of interest information, and tools such as shape projection that use the calibration information can make use of this information to limit the extent of the shape projection.

Note You must specify a rectangle for each camera; the default is the entire image.

Acquiring the Tilted Viewsets (Required)

To acquire the *cc3DCameraCalibDefs::ePoseTilted* 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 26.

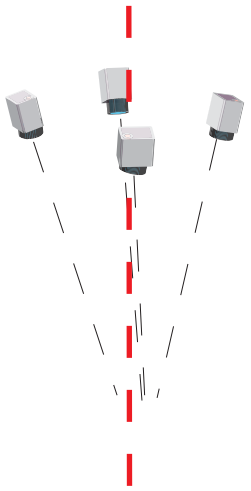


Figure 26. Rotation axis for tilted viewsets

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 27, and acquire the first viewset.

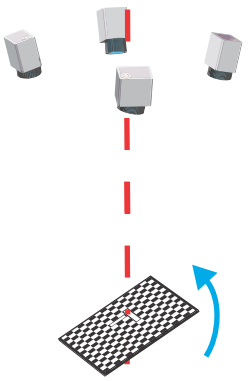


Figure 27. Tilting the calibration plate

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 28 shows the acquisition of all four of the required tilted viewsets:

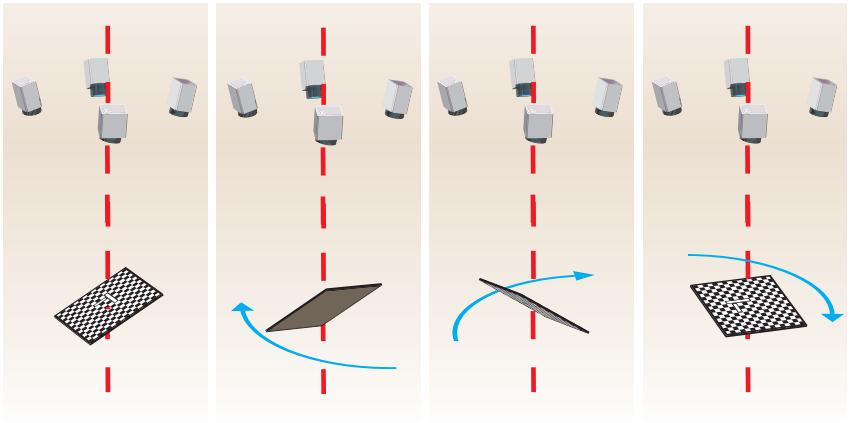


Figure 28. Rotating the calibration plate

Elevated Viewsets (Optional)

Acquiring *cc3DCameraCalibDefs::ePoseElevated* 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 29.

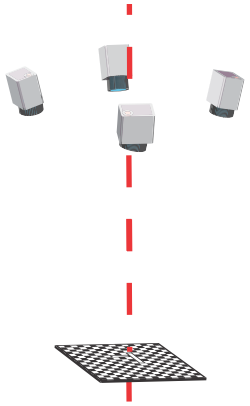


Figure 29. Anchor plate for elevated viewsets.

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 30. You must precisely record the actual spacing between each plate, and the plate must be kept parallel to the first elevated viewset pose.

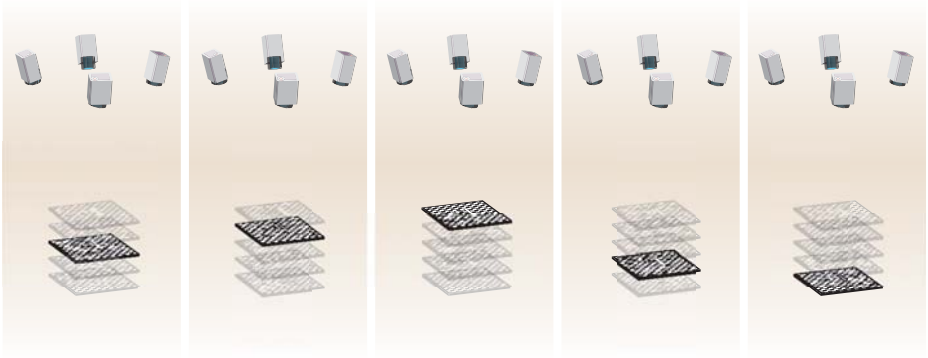


Figure 30. Elevated viewset poses

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 *(cc3DCameraCalibDefs::ePoseDefineWorldCoord)* . 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 **cfCalib2VertexFeatureExtract()** 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 **ccCrspPairWeightedVector** 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: **cf3DCalibrateCameras()** .

Organizing Viewset Data

cf3DCalibrateCameras() depends on the proper correspondence being maintained for the data sets.**cf3DCalibrateCameras()** takes a vector of **cc3DCameraCalibFeatures** objects, one object for each viewset. **cc3DCameraCalibFeatures** has a member, **cc3DCameraCalibFeatures::features()**, which is a vector of **ccCrspPairWeightedVector** objects, one for each image in the viewset.

The order in which you supply the viewsets (the order of **cc3DCameraCalibFeatures**) is known as *pose order*. Within each **cc3DCameraCalibFeatures::features** vector, the order **ccCrspPairWeightedVector** objects, known as *camera order*, must be the same. Figure 31 shows how the data are organized.

Figure 31. Organizing viewset data

In addition to the vector of **ccCrspPairWeightedVector** objects, each **cc3DCameraCalibFeatures** also contains an enumeration that lets you specify the pose type of the viewset (tilted, elevated, or world origin).

Note If you are supplying *any* elevated viewsets, the world origin viewset must be an elevated viewset.

Once you have organized the input data, call the global function **cf3DCalibrateCameras()** to perform the calibration.

Intrinsic and Extrinsic Calibration Data

The result of calling **cf3DCalibrateCameras()** is a **cc3DCameraCalibResult** object that contains a vector of **cc3DCameraCalib** 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 **cc3DCameraCalib** object contains individual transformation objects for each of the three transformations described in the section *Some Useful Definitions* on page 48:

- **cc3DCameraCalib::raw2DFromCamera2D()** describe the camera intrinsics.
- **cc3DCameraCalib::camera3DFromPhys3D()** gives the camera extrinsics (the 3D pose of the camera in Phys3D space).

- **cc3DCameraCalib::pointRaw2DFromPointPhys3D()** and **cc3DCameraCalib::rayPhys3DFromPointRaw2D()** 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 **cf3DCalibrateCameras()** . 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 **cf3DCalibrateCameras()** that allows you to provide precomputed camera intrinsics, supplying the camera intrinsics from the original calibration.
- You can create a new **cc3DCameraCalib** 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 **cc3DCameraCalib::cloneWithNewCamera3DFromPhys3D()** , or **cc3DCameraCalib::cloneComposeWithPhys3DFromAny3D()** .

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 accurately 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 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 **cc3DResiduals** class is a container that holds residual error statistics for a given set of point pairs. During 3D camera calibration, the **cf3DCalibrateCameras()** function 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 **cc3DResidual** objects:

- Statistics for all plate poses and all cameras

  cc3DCameraCalibResult::residualsRaw2D
  and
  cc3DCameraCalibResult::residualsPhys3D

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

  cc3DCameraCalibResult::cameraResults()[i].
  cameraPlateResults[j].residualsRaw2D()
  and
  cc3DCameraCalibResult::cameraResults()[i].
  cameraPlateResults[j].residualsPhys3D()

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).

In addition to a global RMS error measurement for both 2D and 3D error, the camera calibration validation tool also returns a **cc3DCameraCalibResult** that provides individual residual error data for all of the calibrated cameras.

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.

Figure 32 shows these two 3D spaces and the transformation that are produced by hand-eye calibration.

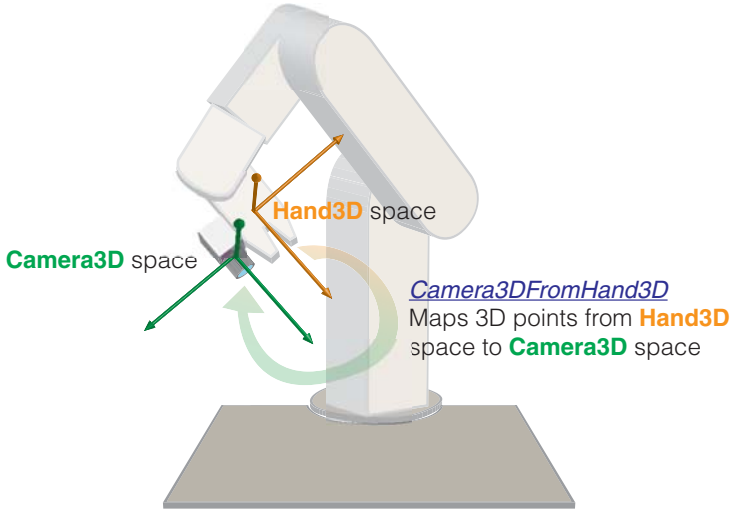
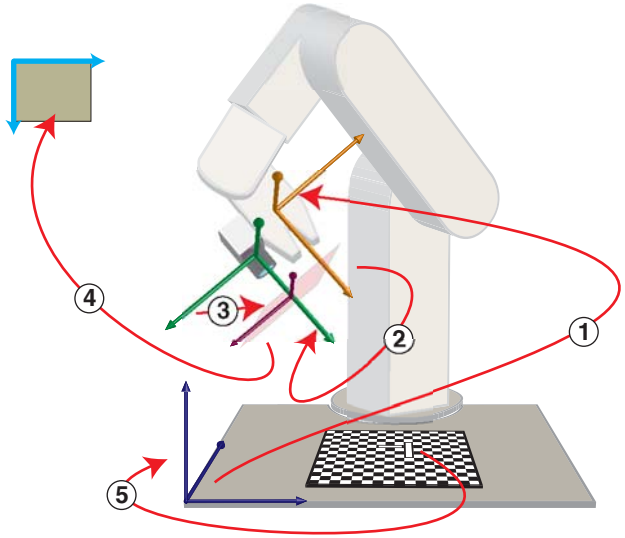


Figure 32. Transformation produced by hand-eye calibration

The Camera3D space shown in Figure 32 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. In fact, hand-eye calibration automatically performs a standard 3D camera calibration as part of the hand-eye calibration process.

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 33 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 33. 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 34.

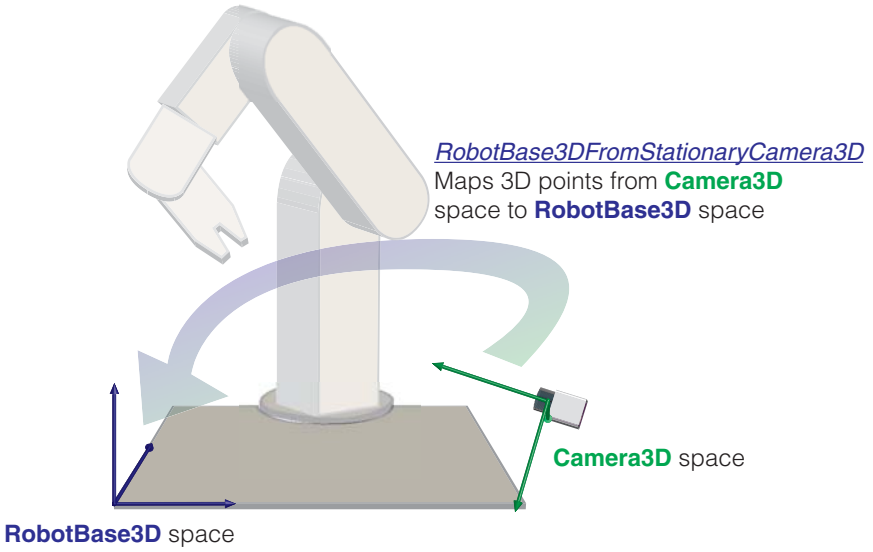
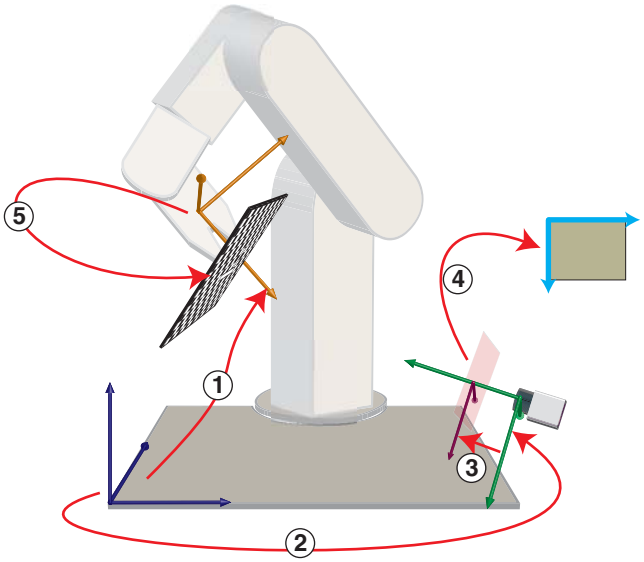


Figure 34. Hand-eye calibration result for stationary camera/moving plate case.

The full outputs for stationary camera/moving plate hand-eye calibration are shown in Figure 34.



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 35. 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:

- Hand-eye calibration can automatically compute the camera intrinsics during hand-eye calibration. This method is simplest, since all calibration is performed at once.
- 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: For applications that use a multi-camera head, 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 on the multi-camera head and you must use this calibration to compute the extrinsics (the pose of the head 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 36.

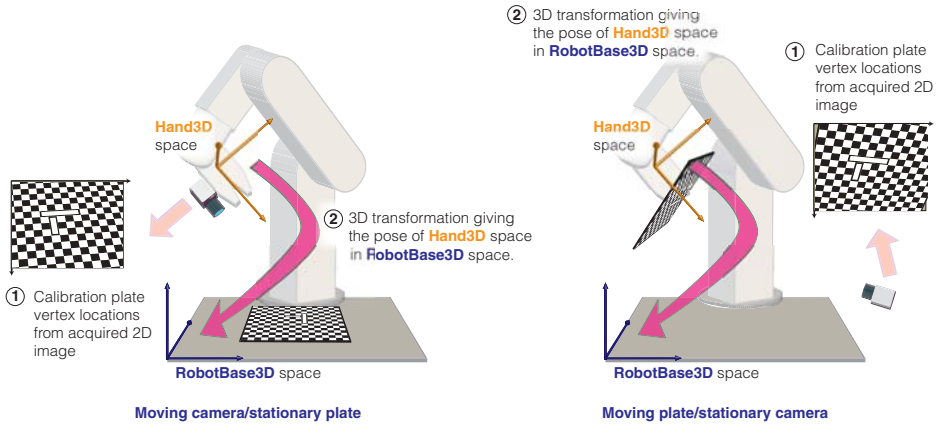


Figure 36. Per-station inputs

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.
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 70. 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 **cfCalib2VertexFeatureExtract()** .
9. Call the **cf3DHandEyeCalibration()** global function with the feature correspondence pairs, the corresponding 3D hand poses, and (if you are supplying the camera intrinsics) 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 70. 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 CVL function **cfCalib2VertexFeatureExtract()**
9. Call the **cf3DHandEyeCalibration()** global function with the feature correspondence pairs, the corresponding 3D hand poses, and (if you are supplying the camera intrinsics) 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. It also allows you to obtain several different residual error measures:

- The residual error for a specific sampling point measured across all stations.
- The residual error for all sampling points measured at a particular station.
- The residual error for all sampling points measured across all stations.

CVL Header Files and Sample Code

This section lists the header files and sample code for the 3D calibration tools.

3D Camera Calibration

The following table summarizes the classes and header files that support camera calibration:

Functionality	C++ Class	Header file
Camera calibration	**cc3DCameraCalib** **cc3DCameraCalibFeatures** **cc3DCameraCalibParams** **cc3DCameraCalibCameraPlateResult** **cc3DCameraCalibCameraResult** **cc3DCameraCalibResult** **cf3DCalibrateCameras()**	ch_c3d/ccalib3d.h

Sample Code

The sample code file *c3d_setup.cpp* located in *%VISION_ROOT%\sample\cvl\c3d\basic* shows how to calibrate cameras. See the function **cf3D_CalibrateCameras()**. This sample code also demonstrates the usage of multi-thread worker manager in the **cf3D_MultiCore()** function.

The file c3d_runtime.cpp shows how to verify that a camera is still calibrated. See the **cf3D_CheckCameraCalibration()** function.

The sample code file *c3d_setup.cpp* located in *%VISION_ROOT%\sample\cvl\c3d\basic* shows how to perform world calibration. See the function **cf3D_ComputeWorldCalibration()**.

Hand-Eye Calibration

The following table summarizes the classes and header files that support hand-eye calibration:

Functionality	C++ Class	Header file
Hand-eye calibration	**cc3DHandEyeCalibrationInputData** **cc3DHandEyeCalibrationInputDataVector** **cc3DHandEyeCalibrationInputDataXO** **cc3DHandEyeCalibrationInputDataVectorXO** **cc3DHandEyeCalibrationRunParams** **cc3DHandEyeCalibrationResidualStatistics** **cc3DHandEyeCalibrationResult** **cc3DHandEyeCalibrationResultXO** **cf3DHandEyeCalibration()**	*ch_c3d/ handeye.h*

Sample Code

The sample code file *c3d_setup.cpp* located in *%VISION_ROOT%\sample\cvl\c3d\basic* shows how to perform hand-eye calibration. See the function **cf3D_ComputeHandEyeCalibration()**.

3D Calibration Wizard 4

Three-dimensional vision applications require that the cameras you use be calibrated, a process that establishes a mathematical relationship between the 2D coordinate system associated with the pixels in each acquired image and a 3D coordinate system associated with the physical world. See Chapter 3, *3D Calibration Tools*, for a description of the methods and API for performing 3D calibration in your Visual Studio project.

In addition to the API for calibration, CVL also supports the 3D Calibration Wizard, a graphical interface for configuring 3D cameras, capturing viewsets, and generating the 3D calibration object of type **cc3DCameraCalib**.

Using the wizard, you specify the number of cameras you are using and provide details about the viewsets you will capture, such as the height of any elevated view sets or the number of tilted viewsets. Once configured, you use the wizard to capture the viewsets you need and perform the calibration itself, generating a *camera calibration archive* (CCA) file that can be read into your Visual Studio application.

In addition, the 3D Calibration Wizard can be used to validate an existing 3D vision environment, which is necessary to verify that the cameras you are currently using all correctly translate the Raw2D space in their acquired images to the identical Phys3D space of the physical coordinate system.

The 3D Calibration Wizard is not appropriate for all vision environments. For example, the wizard cannot be used in robot calibration, where a single machine vision camera is mounted on the end-effector of a robot.

This chapter contains the following sections:

- *Some Useful Definitions* on page 76 defines some of the terms that you will encounter as you read this chapter.
- *Camera Setup* on page 78 describes how to use the wizard to configure the cameras you want to use.
- *Calibration Setup* on page 81 describes how to configure the wizard with the necessary viewsets you need to capture.
- *Viewsets* on page 83 describes how to capture the viewsets you need to generate as well as validate a calibration object.
- *Validation* on page 89 describes how to validate an existing calibration object.

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.

Camera Setup

Launch the 3D Calibration Wizard by choosing **Start->Programs->Cognex->CVL->Utilities->Calibration Wizard**. The wizard launches and displays its **Camera Setup** tab, where you can configure the number, type, and format for the 3D cameras your vision application will use.

By default the wizard is configured for two cameras, and supports a display window for each camera, as shown in Figure 37:

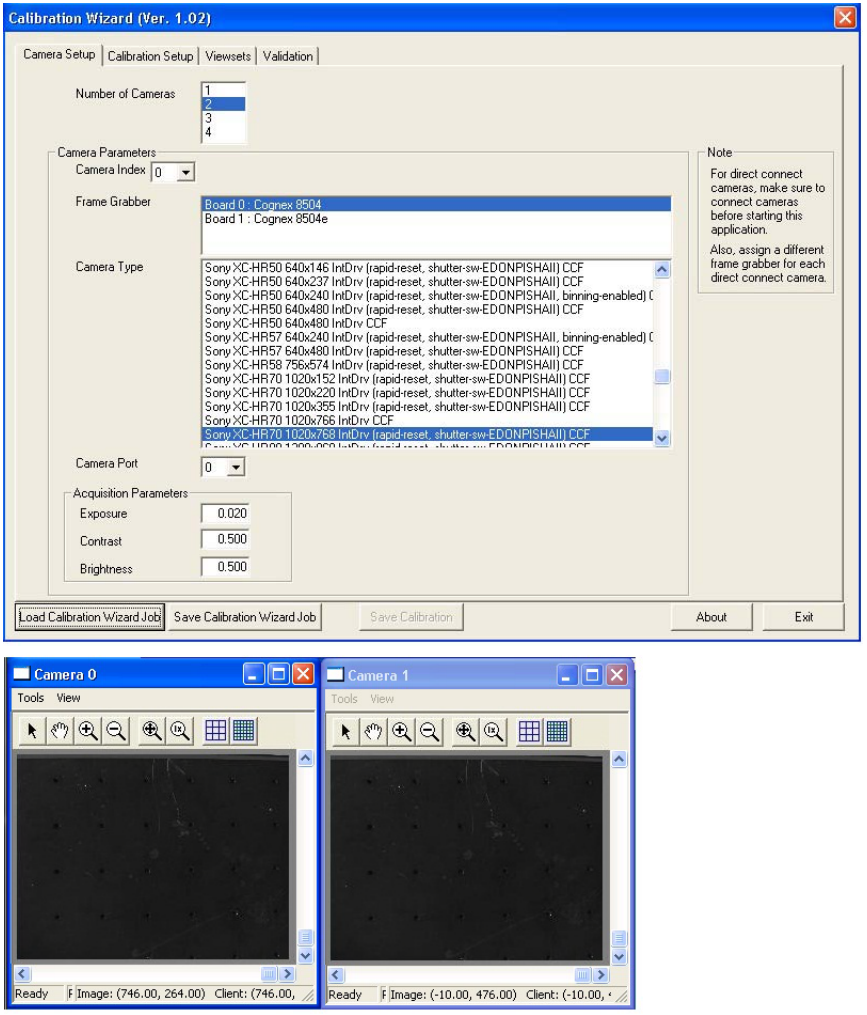
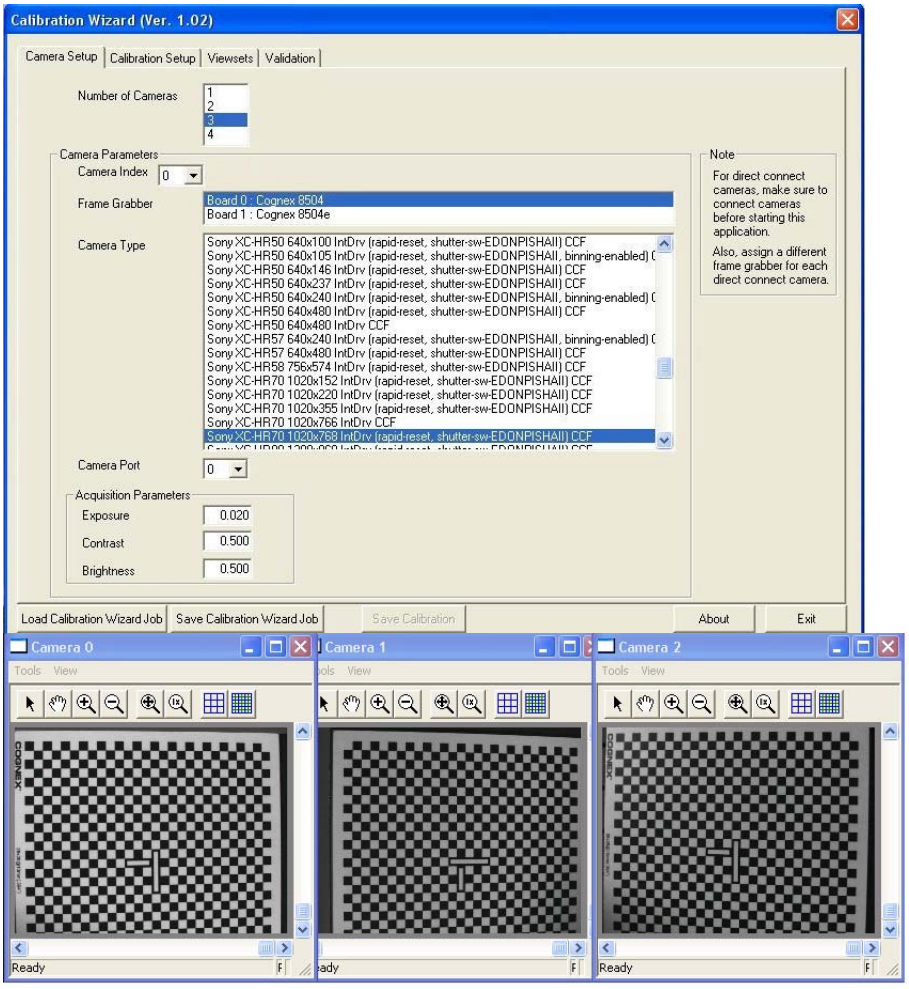


Figure 37. New instance of 3D Calibration Wizard

Choose the **Number of Cameras** your 3D vision environment will use and then configure the following **Camera Parameters** for each camera:

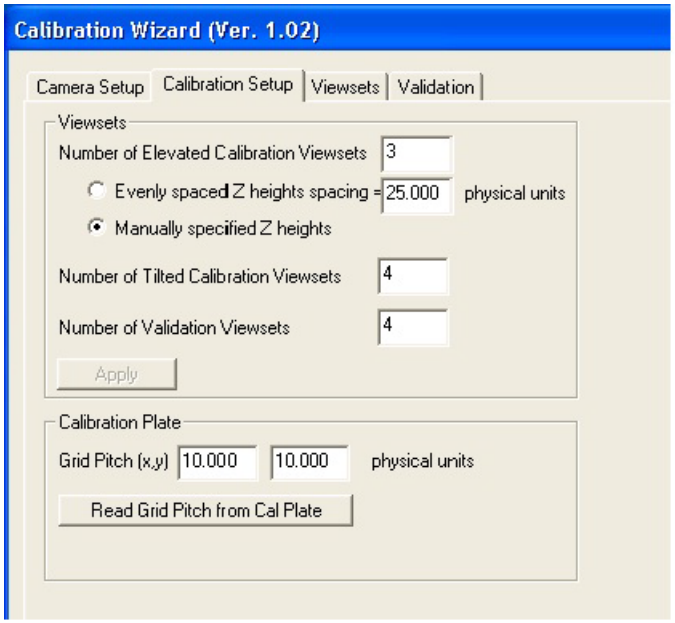
Camera Index	Select the particular camera you want to configure. All of the camera parameters you configure stay with the particular **Camera Index** value. For example, if you link **Camera Index** 0 to the camera attached to port 0 of your frame grabber and later change **Camera Index** 0 to port 1, all of the current acquisition parameters are applied to the camera attached to port 1. If two cameras refer to the same camera/camera port, one of the display images will be blank since each physical camera can only correspond to a unique camera index.
Frame Grabber	Select any camera available to the PC: • A camera connected to a frame grabber installed in the PC • An IEEE-1394 (FireWire) camera • A GigE Vision camera Cognex recommends you use the same camera model for all the cameras of your 3D vision application.
Camera Type	Select a video format for this camera. The 3D Calibration Wizard automatically applies any video format you select. Cognex recommends you use the same camera type for all the cameras of your 3D vision application.
Camera Port	Select the necessary camera port to correspond to this **Camera Index**.
Acquisition Parameters	Adjust the parameters as necessary to acquire suitable images of your physical volume.

Once the cameras are properly configured, the display windows should reflect clear images of the physical volume, as shown in the following example:



Calibration Setup

Use the **Calibration Setup** tab to provide the 3D Calibration Wizard with initial details of the viewsets you will use to generate the 3D calibration object. The following figure shows an example:



The number of viewsets (elevated, tilted, or validation) as well as the spacing between elevated viewsets you specify here act as a starting point for the calibration process. You can change details such as the number and type of viewsets later with the **Viewsets** tab.

Refer to the following sections for more information regarding elevated viewsets, tilted viewsets, and the calibration plate:

- *Elevated Viewsets (Optional)* on page 57 provides details on how to acquire elevated viewsets.
- *Acquiring the Tilted Viewsets (Required)* on page 56 provides details on how to acquire tilted viewsets.
- *World Origin Viewset (Required)* on page 59 provides details on the one (and only one) of the viewsets that must be provided to define the Phys3D space of the working volume.
- *Calibration Plate Requirements* on page 16 describes the calibration plate.

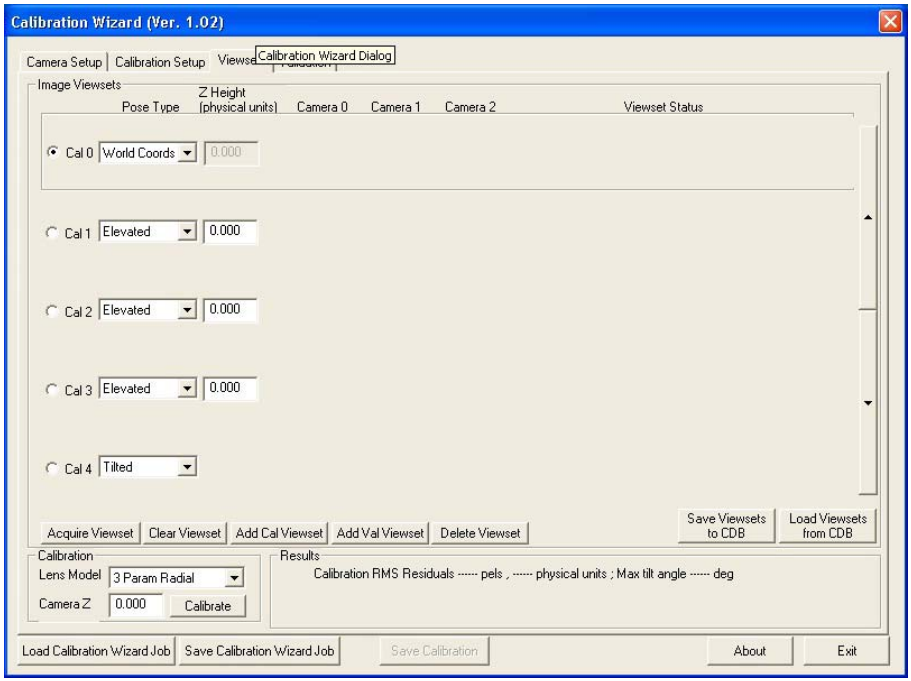
Provide the following information: :

Number of Elevated Calibration Viewsets	Acquiring elevated viewsets is not required to perform 3D calibration, but providing them will result in a more accurate calibration object. You have the option of providing viewsets that are evenly spaced along the Z axis based upon multiples of a known physical unit, or manually spaced viewsets based upon distances along the Z axis that you specify. The actual physical unit used during calibration is irrelevant as long as the interval between elevated viewsets matches the same physical unit for the grid pitch of the calibration plate.
Number of Tilted Calibration Viewsets	Cognex strongly recommends you acquire at least four viewsets that show a standard Cognex checkerboard calibration plate at four specific poses, with a minimum angle of 15 degrees.
Number of Validation Viewsets	Cognex recommends you capture at least four viewsets to provide a baseline later if you need to validate an existing calibration. You cannot perform a validation with calibration viewsets. The easiest approach is to capture four viewsets of the calibration plate rotated 90 degrees around the axis of the working volume.
Calibration Plate	Enter the dimensions for the grid pitch of the calibration plate, or click **Read Grid Pitch** to read the grid pitch from the 2D code printed on a Cognex Calibration plate.

Once the 3D Calibration Wizard has been configured with the number and type of viewsets you want to capture, acquire them with the **Viewsets** tab.

Viewsets

Use the **Viewsets** tab to capture the viewsets you need to perform or validate a 3D calibration. The following figure shows an example **Viewsets** tab before any viewsets have been acquired:



The **Viewsets** tab contains an entry for each viewset you configured the 3D Calibration Wizard to expect using the **Calibration Setup** tab. For example, if you configured the wizard with five elevated viewsets, four tilted viewsets, and four validation viewsets, the **Viewsets** tab contains 14 entries:

- One world origin viewset

  By default the wizard assigns the first viewset to the world origin viewset, as indicated by the **World Coords** option for **Pose Type**.

- Five elevated viewsets (optional)

  The wizard allocates a position for each elevated viewset and might assign values for the height along the Z axis based on your settings in the **Calibration Setup** tab.

For example, if you choose to capture five viewsets that are evenly spaced by an amount of 25 physical units, the 3D Calibration Wizard will allocate five viewsets with Z axis positions of 0, -50, -25, 25, and 50. Cognex recommends you keep these default assignments and acquire the viewsets in this order.

Conversely, if you choose to capture five viewsets that are manually spaced apart, the 3D Calibration Wizard assigns a default Z axis position of 0 to each allocated viewset and allows you to assign the distance of the viewset from the calibration cameras.

- Four tilted viewsets

  Tilted viewsets have no specified height along the Z axis.

- Four validation viewsets

  See the section *Validation* on page 89 for more information.

Choosing the Right Lens Model

Near the bottom of the **Viewsets** tab is a pulldown list for **Lens Model**. Choose the right option based on the camera type you are using:

- **3 Param Radial**

  The default option is appropriate for most camera types.

- **Sine Tangent**

  Typically used for wide-angle or fish-eye lens in which the signature curving effects are severe, usually occurring for lenses with focal lengths such as 4mm or 6mm. This option can require more time for the 3D Calibration Wizard to generate a calibration object.

- **Telecentric**

  Use this option if you are using telecentric cameras, such as cameras with high-magnification lenses (such as microscope lenses) where lens distortion/perspective is usually miniscule.

  Telecentric cameras require a working environment where the calibration plate cannot undergo a change in translation or tilt. For calibration purposes, you must use appropriate hardware that allows the plate to undergo a change only within the Z axis.

In addition, use the **Camera Z** parameter to specify the distance, in physical units between the top of the calibration plate in the world origin viewset and the lens of the telecentric camera. The **Camera Z** parameter has no effect on other choices for **Lens Model**.

Capturing a Viewset

For each viewset you need to acquire, perform the following steps:

1. Select the particular viewset name or click its radio button.

   Use the arrow keys along the right side of the wizard to move up and down the collection of viewsets.

2. Adjust the position/height of the calibration plate to match the desginated **Pose Type**:

   - **World Coords** for the one world origin viewset
   - **Elevated** or **Tilted**
   - **Specified** if you are using the **Telecentric** option for a **Lens Model**

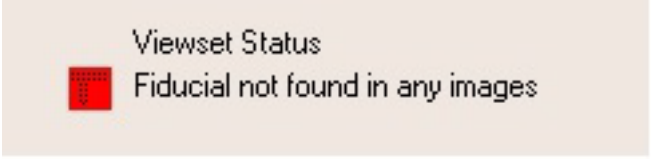
   Validation viewsets require no elevation or tilting.

3. Click **Acquire Viewset** to acquire an image from all 3D calibration cameras simultaneously.

The 3D Calibration Wizard displays the images it acquired and reports on its ability to locate the fiducial in the calibration plate, as shown in the following example:

Click on a thumbnail image to view a full-screen diagnostic display of the found checkers and fiducial mark.

The fiducial must be found in one or more images for the wizard to generate an accurate 3D calibration object. If it cannot locate the fidicual in any image, the wizard displays the images it acquired along with an error message:



Managing Viewsets

Use the controls along the bottom of the **Viewsets** tab to manage, add or remove both calibration and validation viewsets from the current set:

The tab also contains buttons to allow you to save and load viewsets for later use.

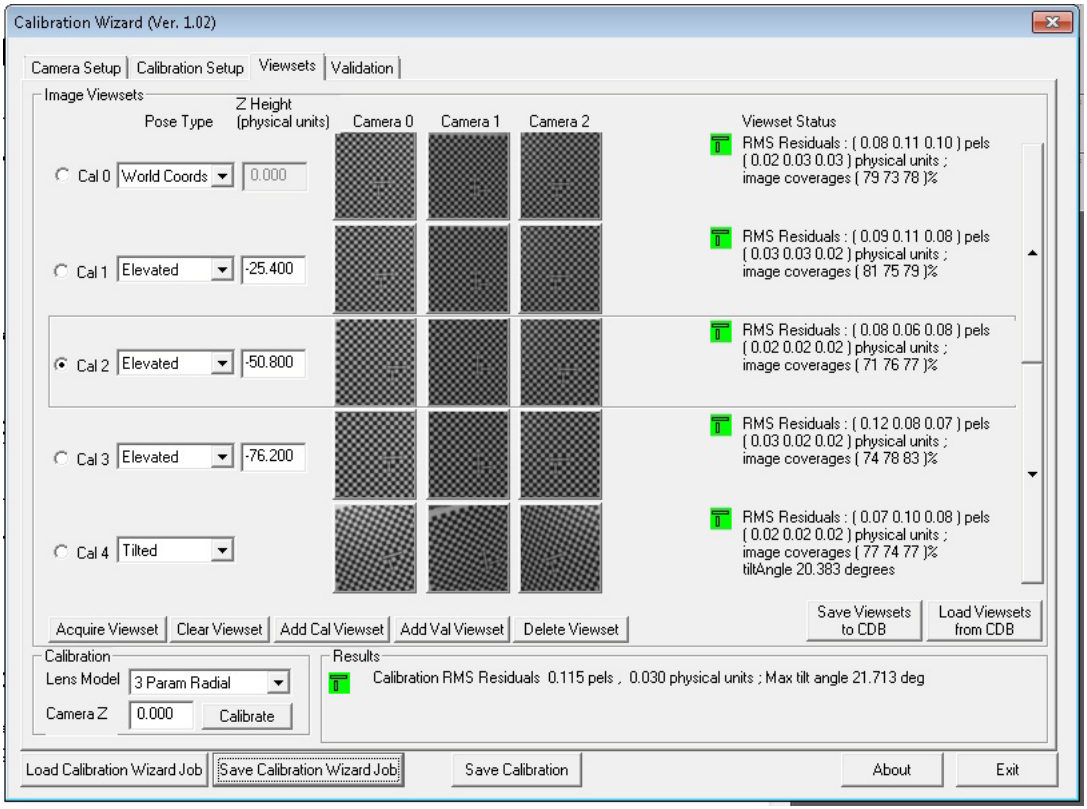
Generating a Calibration Object

Click **Calibrate** along the bottom of the **Viewsets** tab to have the 3D Calibration Wizard attempt to generate a calibration object based on the current viewsets.

Be aware that the calibration process can require a few seconds or several minutes based on the lens model you are using. The wizard displays a progress bar to indicate that calculations are continuing. You can click **Cancel** at any time.

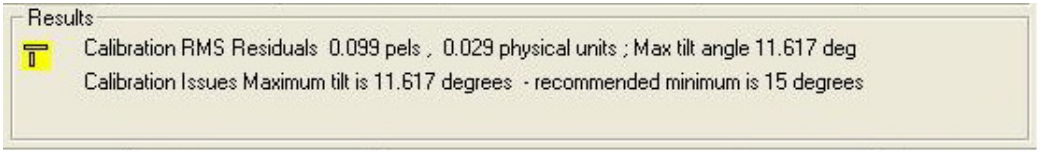
The 3D Calibration Wizard runs the checkerboard feature extractor in the background, asynchronously changing each viewset status to either found or not found.

The following figure shows the wizard after a successful calibration:



The smaller the **RMS Residuals** reported by the wizard, the more accurate the calibration object.

Green fiducial icons represent an accurate calibration, while red icons indicate the calibration procedure was not successful. Yellow icons indicate the generated calibration is less than optimal. In addition, the **Viewsets** tab displays any errors or warnings, as shown in the following figure:



Saving and Loading a Calibration Object

Once a 3D calibration object has been created, save it by clicking **Save Calibration** along the bottom of the **Viewsets** tab. The wizard saves it as a Camera Calibration Archive (.cca) file.

To load it into your CVL application, use the **ccFileArchive** class as shown in the following sample code:

```
cc3DCameraCalibResult cameraCalib;

ccFileArchive resArchiveLoad(_T("./calib3d.cca"),
  ccArchive::eLoading, ccArchive::eLittleEndian, true, true);
resArchiveLoad >> cameraCalib;
```

Refer to the CVL Class Reference for more information on the **ccFileArchive** class.

Saving and Loading Calibration Jobs

Use the **Save Calibration Wizard Job** feature to save the current set of parameters to a *Camera Calibration Wizard Job* (.cwz) file, or use the **Load Calibration Wizard Job** feature to load an existing Job file into the 3D Calibration Wizard.

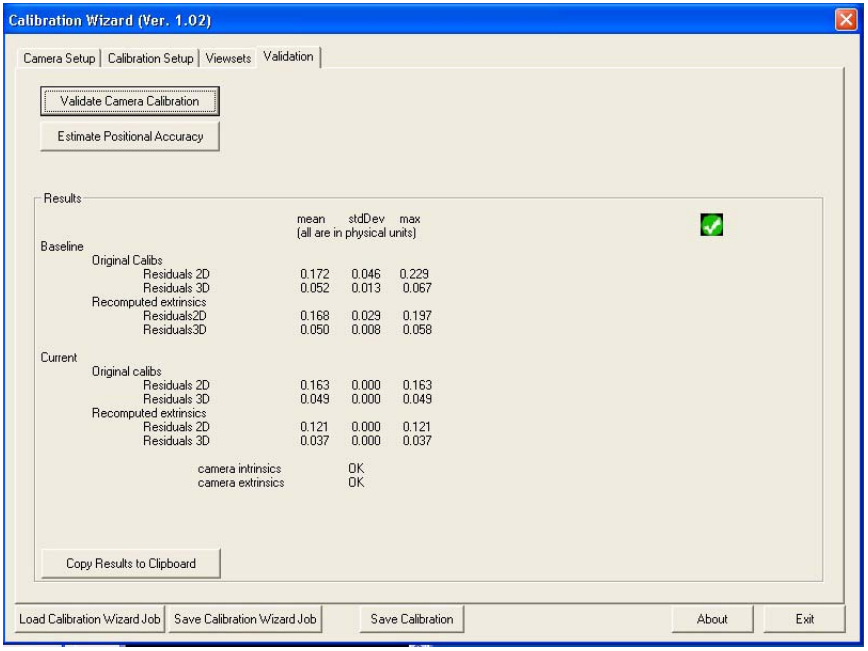
Validation

Over time the precise location, position and physical characteristics of the cameras in your 3D working environment can change due to intentional or unintentional causes, such as vibrations in the production environment or accidental contact with the vision equipment.

Validating an existing calibration can be necessary when you need to verify the accuracy of a calibration object to the current physical setup of the cameras in your working volume.

The Validation function compares an aquired image of the calibration plate in the Phys3D space to a set of baseline validation viewsets you captured during the initial calibration process. You cannot perform a validation of an existing calibration object without a set of validation viewsets as a baseline.

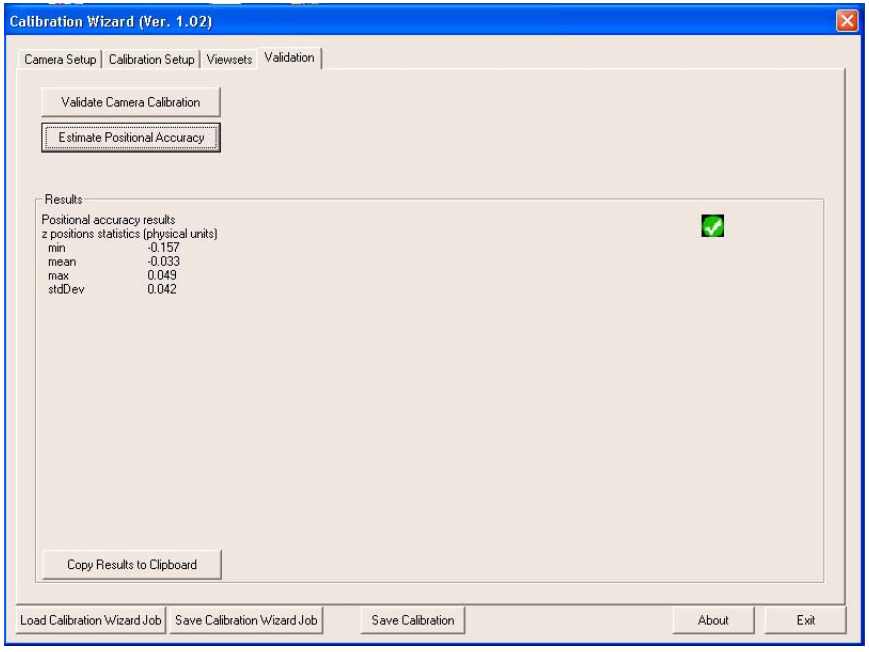
Click **Validate Camera Calibration** on the **Validation** tab to perform a validation. The 3D Calibration Wizard prompts you to position the calibration plate in the working volume. Hold the calibration plate similar to the tilt used for the baseline images and click **OK**. The wizard acquires a viewset of the plate and performs the validation, and reports results as shown in the following example:



The wizard provides an overall indication of camera intrinsics (settings internal to the camera) and camera extrinsics (position of the camera), and gives specific values for measurements performed on the baseline validation viewsets and the new validation viewset.

In addition to camera validation, the **Validation** tab can also provide an estimation of how well the 3D calibrated cameras are able to measure the position of features along the Z axis in terms of accuracy and consistency.

Click Estimate Positional Accuracy and provide a new viewset of the calibration plate when prompted. The 3D Calibration Wizard will generate this data and display it as shown in the following example:



Smaller values indicate more accuracy in the ability to detect the distance between features in the objects you want to locate in the Phys3D Space environment.

Locating Objects in 3D

5

This chapter describes how to locate features and objects in three dimensions. The basic 3D location capabilities are

This chapter contains the following sections:

- *Some Useful Definitions* on page 92, provide an overview of the chapter and define some terms that you will encounter as you read.
- *Triangulating 3D Points from 2D Points* on page 93 describes how to compute the 3D location of a point by mapping multiple 2D points from different cameras to 3D rays, then computing the intersection of the rays.
- *3D Pose Estimation from 2D Point Sets* on page 95 describes how to estimate the 3D pose of a 3D shape based on 2D points from one or more calibrated cameras.
- *3D Fitting* on page 99 describes the tools that allow you to fit 3D shapes to sets of 3D points.
- *Working in 3D Using 2D Vision Tools* on page 100 provides guidance on the methods for using 2D vision tools to obtain accurate feature locations of 3D objects.

For more information on the 3D Vision Framework, examine the header files and sample code files listed in the section *CVL Header Files and Sample Code* on page 105.

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 used to perform 3D calibration. This space can be defined by any coordinate frame in physical space.

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.

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

Triangulating 3D Points from 2D Points

3D camera calibration allows you to map from a 2D image point in an image acquired from a 3D-calibrated camera to a 3D ray in 3D physical space. Given two cameras, each calibrated to the same 3D physical space, and a pair of 2D points, one from each camera, that correspond to the same 3D physical point, you can determine the 3D location of that physical point through change bars.

Note Triangulation *requires* that all of the cameras used to acquire the 2D images share the same 3D physical space. This requirement is automatically met when all of the cameras are 3D calibrated together with the same call to **cf3DCalibrateCameras()** .

Figure 38 shows how triangulation works.

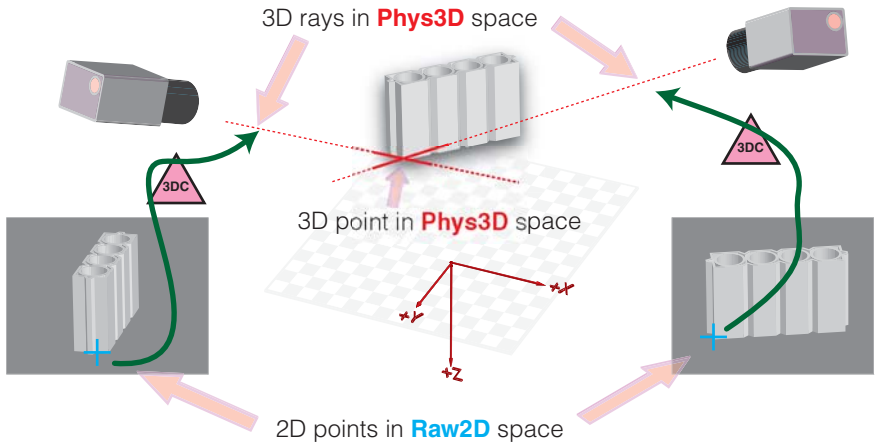


Figure 38. Triangulation

Triangulation can be performed using two or more cameras. Successfully triangulating the location of a 3D point requires that the 2D image points are accurate projections of the same 3D point to each of the cameras. In Figure 38, each of the 2D points corresponds to the same corner of the part, as shown in Figure 39.

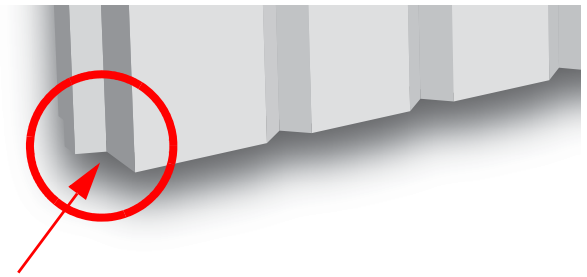


Figure 39. Part feature

This requirement, that the 2D points from each image accurately give the projected location of the same 3D feature is discussed in detail in the section *Working in 3D Using 2D Vision Tools* on page 100.

3D Pose Estimation from 2D Point Sets

3D-Locate provides two tools that estimate the poses of 3D objects from sets of 2D image points.

3D Shape Pose Estimation

Given one or more 3D calibrated cameras, 3D-Locate can determine the pose of a 3D shape based on 2D image points that correspond to features of the object.

Note In this release of 3D-Locate, this capability is supported for 3D circles only.

To estimate the 3D pose of a circle from a set of 2D points, simply supply the points and the camera calibration objects for the cameras used to acquire the 2D images. You can specify the nominal radius of the circle to fit, or you can have the tool determine the radius of the circle.

Note The tool can only compute the radius when you provide points and calibrations from multiple cameras.

Depending on the points that you supply, the tool may return an error (no circle exists that fits the supplied points), the pose of a single 3D circle, or the poses of two circles, each of which fits the supplied points equally well, as shown in Figure 40.

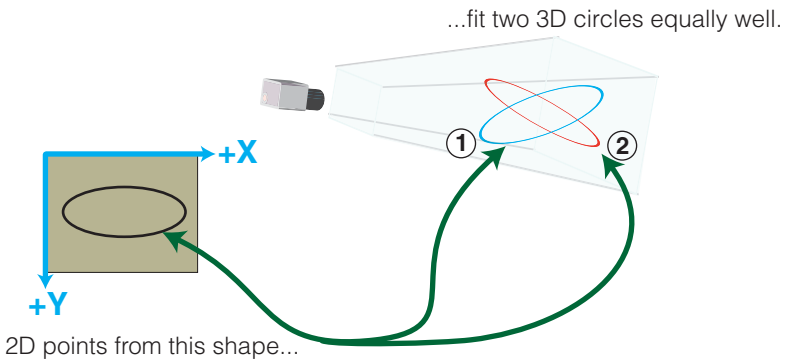


Figure 40. Two circles may fit the given points.

3D Model Pose Estimation

As described in the section *Estimating 3D Object Poses* on page 27, given a collection of 3D model points that describe an object and one or more corresponding sets of 2D image points acquired from 3D calibrated cameras, 3D-Locate can determine the pose of a 3D shape based on 2D image points that correspond to features of the object.

Unlike 3D point triangulation, which *requires* multiple cameras, 3D model pose estimation can be performed using a single camera.

Successful 3D model pose estimation depends on three factors (in addition to having an accurate 3D camera calibration):

- You must construct an accurate 3D model of the object that you are aligning.
- The points that you obtain from the calibrated images must be accurate projections of 3D points defined by the model.
- The 2D points that you obtain from the calibrated images must be correctly corresponded to the 3D model points.

Note If you are using a single camera for 3D model pose estimation, then the following additional requirements apply: At least three of the 3D model points must be non-collinear and the 2D image points must be widely spaced in the image.

Each of these requirements is discussed in the following sections.

Constructing a 3D Model

You specify a 3D model for pose estimation by supplying a collection of 3D points, all of which describe visible features on the surface of the model. For example, if you had an accurate CAD model of the part shown in Figure 38, it might include the positions of the eight outermost corners of the part, as shown in Figure 41.

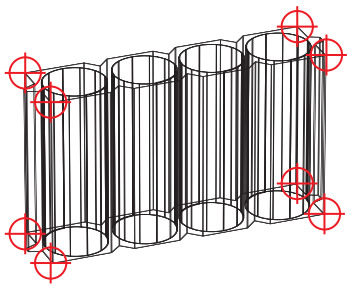


Figure 41. CAD model points

For the part shown in Figure 41, CAD data would be available for many other points. The points shown in Figure 41, however, have two important attributes:

- They give the locations of features that are visible in a 2D image of the part.
- They are feature points that can be accurately located in a 2D image of the part.

The 3D points that you supply to define a 3D model define their own 3D coordinate space. This space is called 3D model space. The 3D model pose estimation function returns the pose of 3D model space in 3D physical space.

Obtaining Accurate 2D Model Points

As discussed in the section *Working in 3D Using 2D Vision Tools* on page 100, performing accurate 3D alignment requires that you obtain highly accurate 2D locations that correspond to the projected location of a 3D feature.

One effective technique that you can use to locate corner points in the 2D image is to use a line finding tool (**cfCaliperFindLine()**) to locate edges that intersect at the model point, then use then use the **ccLine.intersect()** method to determine the 2D intersection point, as shown in Figure 42.

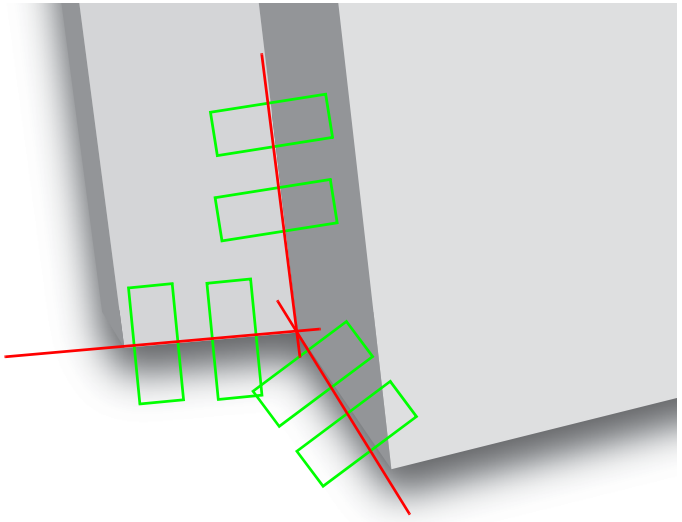


Figure 42. Obtaining an accurate 2D feature location

The section *Working in 3D Using 2D Vision Tools* on page 100 contains a general discussion of this topic.

Corresponding 2D Image Points to 3D Model Points

As described in the section *Estimating 3D Object Poses* on page 27, using the 3D model pose estimation tool requires that you supply 2*n* sets of points, where *n* is the number of cameras that are using. For a given camera, the first set of points are the 3D model points and second set is the 2D image points that correspond to the model points. It is an absolute requirement that for each pair of points, the order of these points be the same. If the points do not correspond, the pose estimator will fail to work.

3D Fitting

3D-Locate provides tools that let you fit 3D shapes to sets of 3D points and compute rigid transformations between sets of 3D points.

3D Shape Fitting

Given a set of 3D points, 3D-Locate can fit those points to a 3D line, 3D circle, or 3D plane. The result of the fitting operation minimizes the least squares error.

3D shape fitting does not require the use of a 3D camera calibration operation. Typically, you will have generated the 3D points using a separate triangulation operation.

3D Model Rigid Transformation Computation

Given two sets of 3D points, 3D-Locate can compute the 3D rigid transformation that maps between the two sets of points. The computed transformation minimizes the least squares error between the two point set poses.

You can use this functionality to determine the movement of 3D objects. Typically, you will generate the 3D locations of the points using a separate triangulation operation.

Working in 3D Using 2D Vision Tools

All of the 3D vision alignment tools described in this chapter depend upon your application's ability to provide accurate 2D image locations that correspond to a 3D physical point. The Cognex vision tools that you use to locate 2D features in images are not designed to provide direct information about 3D feature locations; instead, you must use these tools in specific ways to generate 2D locations that accurately reflect 3D features.

In general, two types of 3D object features can be visualized in two dimensions - part edges and 2D features on planar part surfaces. Each feature type is discussed below.

Part Edges

Generally, rounded part edges cannot be used, as the apparent edge in a 2D image does not correspond to a consistent 3D edge as the part rotates, as shown in Figure 43.

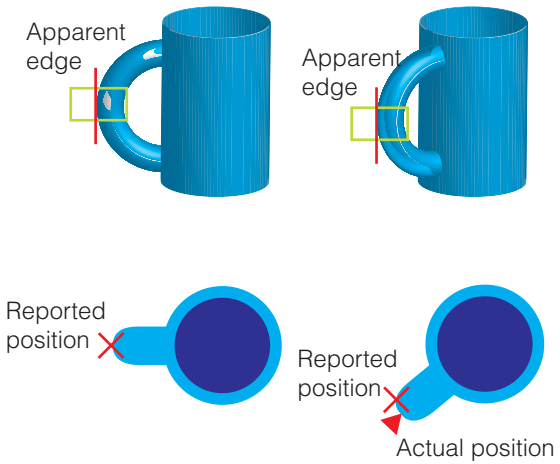


Figure 43. Movement of apparent edge with rotation.

Part corners and sharply creased edges, on the other hand, minimize the effect of rotation on the reported edge, as shown in Figure 44.

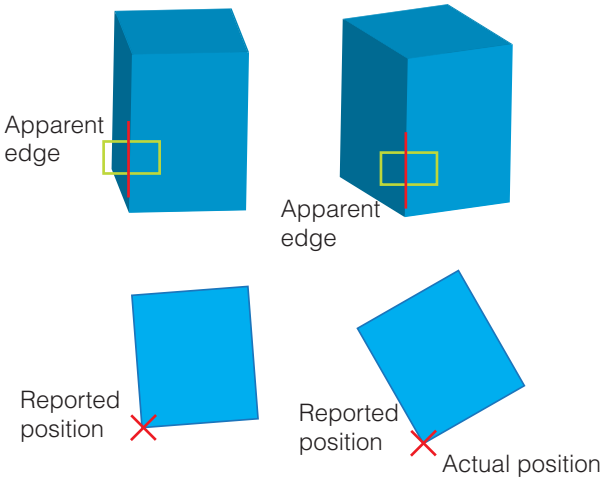


Figure 44. Using sharp edges or corner features.

As shown in Figure 42 on page 97, to obtain a 3D point, you will typically use CVL tools to find multiple intersecting edges as well as the intersection point. Note that the intersection point is a 2D point; you use triangulation to determine the corresponding 3D location.

Note Your application must use other Cognex vision tools to perform a coarse alignment in order to determine where to place the line finding tool.

Locating Features on Planar Surfaces

If your objects or parts include planar surfaces with 2D features that appear at a consistent location, you may be able to use those features to establish 3D locations on that planar surface.

Using Pattern Alignment Tools

In general, however, you will not be able to use 2D pattern alignment tools (such as PatMax, CNLSearch, and RSI Search) for 3D vision. These tools are unable to tolerate pattern deformation caused by perspective distortion. For relatively small amounts of rotation, the tools may still find the pattern, but the accuracy of the returned position deteriorates rapidly, as shown in Figure 45.

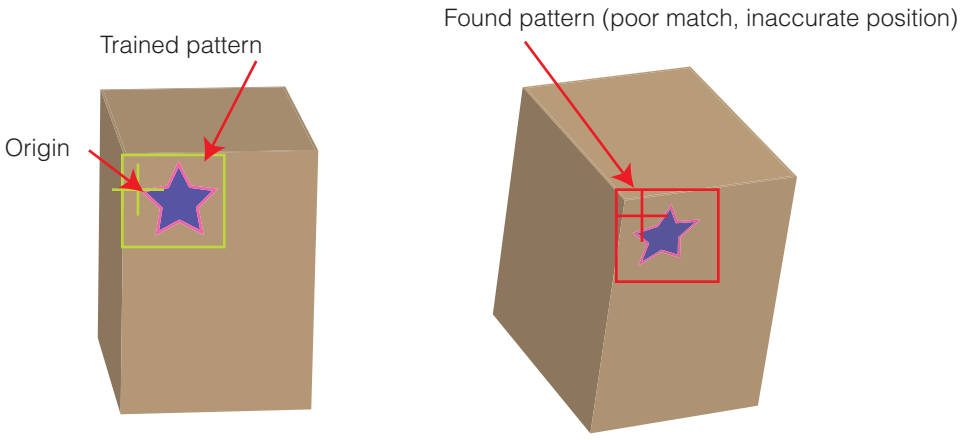


Figure 45. Using pattern alignment tools for 3D alignment

If your application experiences limited rotation at run time with respect to the trained pattern, however, you might have success using a search tool.

Using Linear Features

If a planar surface of your object includes linear features, you can use the same techniques described in the section *Obtaining Accurate 2D Model Points* on page 97 and shown in the section *Part Edges* on page 100, to locate part features.

Figure 46 shows how you can use the CVL 2D line finding tool to establish the edges of a rectangular feature, then locate the corner of the feature by computing the intersection of the found lines.

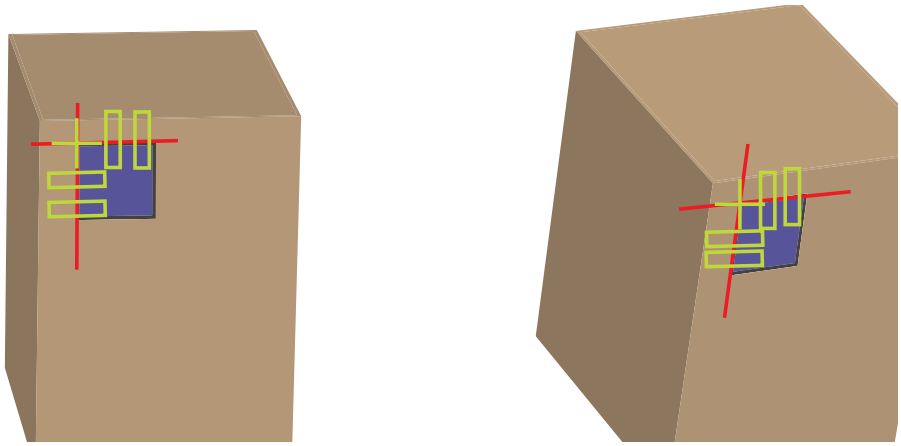
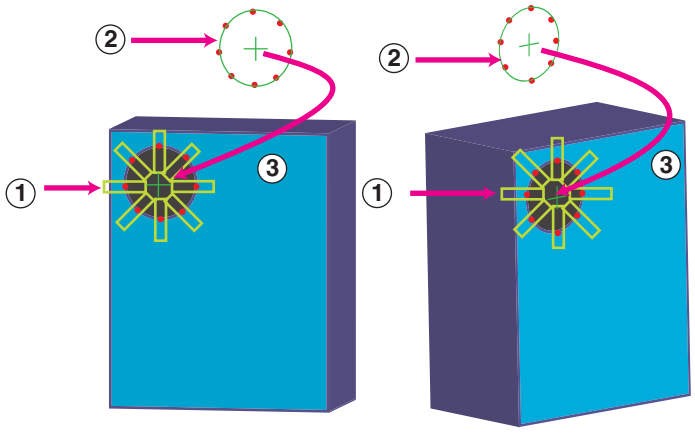


Figure 46. Using linear features to locate a point on a planar surface.

Using Circular Features

Circular part features offer an important advantage for 3D applications: 3D-Locate provides direct support for fitting a set of 2D points from one or more 3D calibrated cameras directly to a 3D circle, as described in the section *3D Shape Pose Estimation* on page 95.

If your part includes circular features, particularly circular features on a planar surface, and if you can obtain sufficient 2D edge points from the image, you can directly obtain a 3D circle that fits those points. The center point of the circle will be highly stable as the pose of the part changes in three dimensions. Figure 47 shows the use of such a feature.



(1) Find 2D edge points.
(2) Fit 3D circle.
(3) Vision tool gives 3D point.

Figure 47. Using circular features on planar surfaces

Note You should *not* use a 2D circle finding tool for this technique. Instead, manually configure and run individual caliper tools to find edge points on the circle (after first performing a coarse alignment to locate the circular feature), then fit the 2D edge points directly to a 3D circle.

CVL Header Files and Sample Code

This section lists the classes, header files, and sample code that implement the 3D vision location tools discussed in this chapter.

3D Location

The following tables provides a summary of the 3D triangulation, 3D fitting, and 3D pose estimation functions.

Inputs/Outputs	API/Header file
Input: - Camera calibrations - 2D points Output: - 3D point	**cf3DTriangulatePointPhys3DUsingPointsRaw2D()** in *ch_c3d/cmp3dpos.h*
Input: - 3D points in spaceA -Corresponding 3D points in spaceB Output: - 3D rigid transform	**cf3DFitPhysA3DFromPhysB3D()** in *ch_c3d/cmp3dpos.h*
Input: - Camera calibrations - 2D image points corresponded to their 3D model points Output: - 3D rigid transform	**cf3DComputePhys3DFromModel3DUsingPointsRaw2D()** in *ch_c3d/cmp3dpos.h*
Input: - Camera calibrations - 2D image points Output: - 3D circle	**cf3DFitCircle3DUsingPoints2D()** in *ch_c3d/fit2d.h*
Input: - 3D points Output: - 3D line	**cf3DFitLine()** in *ch_c3d/fit3d.h*

Input: - 3D points Output: - 3D plane	**cf3DFitPlane()** in *ch_c3d/fit3d.h*
Input: - 3D points Output: - 3D circle	**cf3DFitCircle()** in *ch_c3d/fit3d.h*
Input: - Camera calibrations - 2D image points corresponded to their 3D model points Output: - 3D rigid transform	**cf3DComputePhys3DFromModel3DUsingPointsRaw2D()** in *ch_c3d/cmp3dpos.h*

Sample Code

The sample code file *c3d_*runtime.*cpp* located in *%VISION_ROOT%\sample\cvl\c3d\basic* shows how to triangulate a set of 3D points from 2D points and then check the coplanarity of the 3D points by fitting a 3D plane through them. See the function **cf3D_Triangulate3DPointsFrom2DPointsAndCheckCoplanarity()**.

*c3d_*runtime.*cpp* also shows how to perform pose estimation using a single camera as well as using multiple cameras. See the functions **cf3D_ComputePartPoseUsingMultipleCameras()** and **cf3D_ComputePartPoseUsingSingleCamera()**.