Introduction

The Adept vision system provides a method to perform visual part inspection. There are many functionalities of the system and its capabilities are far reaching, however this guide will focus on using the system to recognize predefined prototypes of parts. The most difficult part of visual inspection of parts, for all methods, is maintaining consistent lighting. If possible it is best to set up a consistent light source that projects as little shadow as possible off of the parts to be inspected. This provides information about various functions of the vision system, explain how they work, in what order they should be implemented, and where additional references can be found in the Adept User Guides. This guide will not repeat basics on vision image processing and how the adept vision system interfaces with the robot. An example program is also provided.

Getting Started

To use the vision system you must turn it on with the command Enable Vision. A vision calibration must then be performed so the system can correctly identify edges, shapes and boundaries of an object. To open the Vision window click on the “adept” logo in the top left corner, then choose Vision from the menu. The Vision window allows you to acquire, display, and process images, perform prototype training, view camera status, and change the status of most switches. The vision window contains several menus allowing you to perform various tasks. The vision window is most useful for displaying current acquired images, overlays of recognized parts or edges, and prototype training. It does not provide camera calibration or parameter adjustment options, but will be used to display images when performing calibrations with the Vision Calibration Utility Program.

Vision Calibration Program

Loading the Vision Calibration Utility Program

In order to perform calibration of the vision system and adjust vision system parameters the Vision Calibration Utility Program must be used. To load the Adept vision calibration program, at the command prompt type:

```
load c:\util\adv_cal
```

This will load all programs and subroutines associated with the vision calibration program. To execute the Vision Calibration program type:

```
ex a.adv_cal
```

This will execute the vision calibration program. The program will ask for user input of the physical camera number and the virtual camera number to use for the physical camera. Use the values of 1 upon first calibration. (See Camera Types and Field of View) The table at the top provides an overview of calibration status for available cameras. It describes which virtual camera is associated with each physical camera and if a calibration is loaded describes the type of calibration used. A menu of options is available in the calibration program allowing you to perform various calibration related tasks.
**Adjusting Image Settings**

The first thing that should be done is to adjust the camera settings by choosing the option “Adjust camera/image settings” (*menu option 2*). This will bring up a sub-menu allowing you to adjust the different properties of the camera – physical camera properties, gain and offset (grayscale), binary threshold, and image processing boundaries. When adjusting the image settings it is best to have the calibration object located in the camera field of view so the settings can be adjusted to provide the best image quality of the calibration object. You may also need to adjust the camera aperture and focus to achieve the best quality image.

**Camera Only Calibration**

“Calibrate the current camera” (*menu option 3*), will bring up the calibration method sub-menu. Though it is not necessary, it is suggested to first complete a “Camera Only” calibration (*sub-menu option 1*) using the provided Adept nested squares sheet. Place the sheet in the field of view of the camera and follow the instructions for the “Camera Only” calibration (you will also be given the options to adjust the image settings). The vision system will acquire an image and place a graphic overlay of two nested squares, represented by green lines. Manually locate the overlaid squares to be centered and correspond to the edges of a black or white nested square. To rotate the squares drag the red dot in the upper right corner of the overlaid squares. You will be asked to provide the largest number within the largest nested square you located and will be given information about the accuracy of the calibration. If you are unsatisfied you are given the option to perform calibration again.

*Note:* It is not necessary to have the nested squares sheet oriented in any particular direction, but accuracy can be improved if it is oriented as square as possible within the vision window.

**Camera to Robot Calibration**

After a camera only calibration has been performed the information can be used to more accurately identify the calibration object used for a camera-to-robot calibration (*sub-menu options 9-12*). The recommended methods are 11.) Non-contact method (single configuration) and 12.) Non-contact method (lefty/right). The other two methods involve equipment that is not equipped to the robot or specific calibration object location information that may not be available.

The two non-contact calibration methods require that a circular calibration object be placed in the camera field of view. (A quarter, penny, dime, marker cap, etc can be used). After selecting the desired calibration method you will be given the option to adjust the image settings, asked if you are using a vacuum gripper, and asked if the tool can be rotated about its Z-axis. (No vacuum gripper is attached and the tool can be rotated about the Z-axis.) The robot will then move to several locations acquiring and processing images in order to develop an orientation, in the X-Y plane only, of the relationship between the link2 location and the camera location. After completing the X-Y plane relationship you will be asked to lower the end effector to its operating height several times. After completing this task the information is used to create the transformation `to.cam[]` and the transformation `grip.trans`. These transformations will not be available until the camera-to-robot calibration is completed.
Important: Transformations to.cam[] and grip.trans are relative transformations and must always be used in this manner (see Developing the Global Location of a Recognized Prototype). To.cam[] is relative to the link2 location and grip.trans is the transformation to account for tool offset.

Example:

If an object is identified by a successful VLOCATE() command its position can be stored in a location variable such as obj.loc. The following determines the global object location and accounts for the tool offset:

\[ \text{global.loc} = \text{link2:to.cam[1]:obj.loc:grip.trans} \]

Note: In the example program provided grip.trans was modified to account for offsets reported about the centroid location of recognized objects and is replaced with ggrip.trans and bgrip.trans. (See Defining and Modifying Transformation and Location Variables)

Storing and Retrieving Calibration Data

The vision calibration program will ask to if you want to store the calibration information upon successfully completing a calibration. You are give the options of where to store the file, the number of the calibration area to use, and comments to help identify the conditions under which the calibration was performed.

Vision calibrations are stored as areaX.dat. Where X is a number specified by the user.

After storing a vision calibration file there are two methods for loading vision calibration data:

1.) Within the vision calibration program use main menu (option 2) to load or store vision calibration data. You must provide the location where calibration areas are stored and will be shown all available calibrations areas in the directory to choose from.
2.) You can load the vision calibration data from within a program by calling the adept load.area function.

To use the LOADAREA utility program you must first load the program into memory by typing at the command prompt:

```plaintext
load c:\util\loadarea
```

The load.area function can now be called from within program instruction by using the following code:

```plaintext
CALL load.area($area, cam.virt, threshold, backlight, to.cam, cam.cal[], pmm.to.pix[], pix.to.pmm[], pmm.to.mm[], mm.to.pmm[], $error)
```

When this function is called and calibration data is successfully loaded the vision system will automatically be enabled. If there is not an error in loading the calibration data the string argument $error will be given the value of “ “ (blank character). This can be used to
provide a useful check to ensure vision calibration information is properly loaded. *(See example program)* If the `call load.area( )` function is used within a program, when the program is loaded into memory the **LOADAREA** utility program will automatically be loaded.

*Note*: This function call must be typed as shown in order to properly restore the vision calibration information stored in the file.

**Camera Types and Field of View**

The adept vision system provides options for using up to four physical cameras and up to 32 (max) virtual cameras. In the provided example program only one physical and one virtual camera is used. Multiple virtual cameras can be assigned to each physical camera, and each virtual camera can possess image and calibration information specific to itself. This way you can process one image from one camera with many virtual cameras to increase effectiveness of the vision system and use many inspection methods if needed. Each virtual camera can also posses its own set of enabled **switches** and **parameter** values.

The camera attached to the robot is has a field of view of 640x480 pixels. Reference within the field of view is made from the bottom left corner, which has coordinates (1,1). The processing window can be changed to be smaller than 640x480 from within the **Vision Calibration Utility Program** or by setting parameter values to create the desired processing window.

**Image Acquisition, Processing and Recognition**

**Image Acquisition and Display**

Images can be acquired by using options from the Pict menu in the vision window or by using the following commands.

To acquire an image the `VPicture( )` command is used with the following syntax:

```
VPicture(cam.virt) mode
```

- **cam.virt**: The current virtual camera number used to take the picture.
- **mode**: -1: a new image should be acquired and immediately processed
  2: an image should be acquired for later processing

*Note*: In some cases it may help recognition accuracy if the image is not processed immediately

To display an image the `VDisplay` command is used with the following syntax:

```
VDisplay mode, overlay
```

- **mode**: -1: Live grayscale
  0: Live binary
  1: Acquired Grayscale Image
  2: Acquired binary image display
  3: Graphical representation of processed image

```
Don’t erase graphics with subsequent image acquisition

overlay: 0 No user graphics overlay
          1 user and system graphics overly
          2 Don’t erase graphics overlay on subsequent image acquisition

Vision System Switches and Parameters

Presented is a list of useful vision switches and parameters with a brief description of their effect. Switches are binary and are either ON or OFF. Parameters can be given specified values, for example the binary threshold. Switches and Parameters provide means for adjusting how an image is processed and how the acquired image appears. Many switches can be accessed from the Switches menu in the vision window.

Switches and parameters can be set for each virtual camera.

Setting Switches

   Enable switch[cam.virt]
   Disable switch[cam.virt]

   If the argument cam.virt is not provided the switch is set for all virtual cameras.

Setting Parameters

   Parameter parameter_name[cam.virt] = value

   If the argument cam.virt is not provided the parameter is set for all virtual cameras.

At any time a status list of all the switches or parameters can be see by typing at the command prompt:

   Switch or Parameter

   The status list is helpful if parameters are changed within the Vision Calibration Utility Program to retrieve a list of the value for all parameters than may have been changed.

<table>
<thead>
<tr>
<th>Switches</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VISION</td>
<td>Must be enable to use vision system</td>
</tr>
<tr>
<td>V.2ND.MOMENTS</td>
<td>2nd moments of inertia are calculated</td>
</tr>
<tr>
<td>V.BACKLIGHT</td>
<td>Indicates light background and dark objects</td>
</tr>
<tr>
<td>V.BINARY</td>
<td>Enables binary vision processing</td>
</tr>
<tr>
<td>V.BOUNDARIES</td>
<td>Must be enable for vision model processing</td>
</tr>
<tr>
<td>V.CENTROID</td>
<td>Image processing finds x,y location of centroid</td>
</tr>
<tr>
<td>V.DISJOINT</td>
<td>Can Recognize objects that may look like two objects</td>
</tr>
<tr>
<td>V.OVERLAPPING</td>
<td>Can Recognize objects that may overlap</td>
</tr>
<tr>
<td>V.TOUCHING</td>
<td>Can Recognize objects that may be touching</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>V.RECOGNITION</td>
<td>Enable prototype recognition</td>
</tr>
<tr>
<td>V.SHOW.RECOG</td>
<td>Enable graphics overlay showing recognized prototypes</td>
</tr>
<tr>
<td>V.FIT.ARCS</td>
<td>Enable fitting of arcs during vision processing</td>
</tr>
<tr>
<td>V.SHOW.BOUNDS</td>
<td>Enable Graphics overlay of object boundaries identified</td>
</tr>
<tr>
<td>V.HOLES</td>
<td>Statistics are gathered about identified holes</td>
</tr>
<tr>
<td>V.SHOW.EDGES</td>
<td>Enable Graphics overlay of edges identified in processed images</td>
</tr>
</tbody>
</table>

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>V.THRESHOLD</td>
<td>Sets binary threshold value</td>
</tr>
<tr>
<td>V.2ND.TRESH</td>
<td>Sets 2nd threshold value (Values between the two thresholds will be black)</td>
</tr>
<tr>
<td>V.EDGE.STRENGTH</td>
<td>Sets intensity change across edges to be recognized (used for grayscale image processing)</td>
</tr>
<tr>
<td>V.FIRST.COL</td>
<td>First column for vision processing window</td>
</tr>
<tr>
<td>V.LAST.COL</td>
<td>Last column for vision processing window</td>
</tr>
<tr>
<td>V.FIRST.LINE</td>
<td>First line for vision processing windows</td>
</tr>
<tr>
<td>V.LAST.LINE</td>
<td>Last line for vision processing window</td>
</tr>
<tr>
<td>V.GAIN</td>
<td>Sets the gain value for the grayscale image</td>
</tr>
<tr>
<td>V.OFFSET</td>
<td>Sets the offset value for the grayscale image</td>
</tr>
<tr>
<td>V.MAX.AREA</td>
<td>Sets maximum object area system will try to process</td>
</tr>
<tr>
<td>V.MIN.AREA</td>
<td>Sets minimum object area system will try to process</td>
</tr>
<tr>
<td>V.MAX.TIME</td>
<td>Sets maximum time to spend trying to recognize a region</td>
</tr>
</tbody>
</table>

**Object Recognition with Prototype Definition**

There are alternatives to object recognition, which will not be discussed in this guide, and require more processing time and steps in order to inspect parts. These can also be used in addition to object recognition to provide closer inspection and obtain more detail about what is in the picture frame. Either grayscale or binary object recognition can be performed. Binary image processing is simpler and perhaps more accurate, but it cannot make use color gradients. Presented is binary object recognition. (Using grayscale image processing is similar to binary image processing and must be used in conjunction with the parameter V.EDGE.STRENGTH and the V.BINARY switch disabled.)

In order to perform binary object recognition a prototype must first be defined. Prototypes must be defined and used with the same calibration status. When calibration is changed the prototypes should be redefined to ensure accuracy. To define a prototype the following switches must be enabled:

- V.BOUNDARIES
- V.SHOW.BOUNDS
- V.BINARY
- V.BACKLIGHT
- V.FIT.ARCS

The switches can be enabled using the command prompt syntax or through the vision window by using the Switches menu and then selecting which switch to enable or disable. The switches above will allow an acquired image to be processed so that regions, shapes, and boundaries can
properly be identified. These regions and shapes will be used to define a prototype and instances of that prototype that the system will be able to recognize later.

To begin prototype definition the vision switch must be enabled and the vision window must be open on the monitor.

The first step is to ensure that the necessary switches are turned on. Do this by selecting the Switches menu on the far right. The next step is to display a live grayscale or binary image with graphics overlay. Do this by selecting the Display menu and checking the necessary boxes. Place the object in the vision field of view. Acquire a picture and process it by selecting Acquire & Process: -1 from the Pict menu. If successful, an outline of the boundaries should appear overlain on the current image and you are ready to begin prototype training. (If unsuccessful you may need to adjust the image properties) The boundaries will be made up of various line segments and arcs.

**Prototype Training**

**Main Prototype:**

Select the Models menu and choose Train prototype. Enter the prototype_name. The Training side window will appear next to the vision window you will select the virtual cameras to associate with the prototype. A live grayscale image will be shown so you can place the object in the field of view and acquire an image. The image will be processed and a graphical representation of the boundaries of the object will be shown. (The vision window will now provide three menus, Operation, Undo, and Done.) In addition any other identified edges may appear also and in most cases are unwanted. (When defining a prototype and its instances it works best to make sure that the desired object is the only region being given defined edges) If this occurs select the Done menu and choose Discard Example. Remove any extra objects from the field of view and try again to acquire a better image.

**Note:** It may be necessary to change the vision processing boundaries to exclude unwanted regions. If so you must first exit the prototype training process, return to the Vision Calibration Utility Program (option number 2) and then adjust the processing boundaries. The vision processing boundaries can also be changed by setting values for the V.FIRST.COL, V.LAST.COL, V.FIRST.LINE, and V.LAST.LINE parameters.

When you have acquired an acceptable representation for the main prototype instance you can start editing the boundaries and defining the geometry. In the editing mode the Training side window will show information about the current operation, allow you to move amongst regions and lines, and display information about the object boundaries. There are several operations that can be performed and they are chosen by accessing the Operation menu.

**Operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete Corner</td>
<td>Delete a point</td>
</tr>
<tr>
<td>Restore Corner</td>
<td>Restore a point</td>
</tr>
<tr>
<td>Arc &lt;=&gt; Line</td>
<td>Change arcs to lines and lines to arcs</td>
</tr>
<tr>
<td>Delete Region</td>
<td>Delete all boundaries defining a region</td>
</tr>
<tr>
<td>Delete Edge</td>
<td>Delete a line between two points</td>
</tr>
</tbody>
</table>
Create Corner Create a new corner

To execute the selected operation either click the OP button or click on the point/region you wish to perform the operation on. For the first prototype instance it is best to have the least number of points and edges. This will make it easier to identify the orientation of future prototype instances in relation to the main instance. After you are finished editing points select Done and choose use example. You will define the parameters Verify Percentage and the Effort Level to use upon completing the main prototype.

Prototype Instances:

You can add prototype instances by choosing New example from the menus. It is best to define as many prototype instances as possible and in as many orientations as possible to increase accuracy of object recognition. Prototype instances are associated with the main prototype and they are all stored under the main prototype name. Unfortunately it is not possible to view the multiple instances defined for a prototype. You may only view the main prototype.

When adding prototype instances you will go through the same acquiring and editing process as for the main instance, however you are no longer given the option to delete unwanted regions or lines. Therefore it is best to make sure there are no unwanted items in the field of view. When finished editing an instance select Done, use example. You will now be asked to develop the orientation of the prototype instance in relation to the main instance. This is done by selecting corresponding corners, first from the new instance, and then from the prototype. Both the prototype and instance will be displayed and the currently selected edges will flash to help confirm the correct association. When you have defined enough point, edge, and region correlations the vision window will display the main instance with a blue overlay of the current instance in the developed orientation and ask for verification of the alignment of the two instances.
Defining corresponding edges

*Note:* You cannot match corners of arcs to corners of lines.

**Prototype Parameters:**

After the main prototype has been defined the **Parameter** menu will be available. It is important to define the Parameters associated with the prototype. If the system has trouble correctly identifying objects the right combination of these parameters will increase the accuracy. Available prototype parameters are:

- **Verify Percent** (0-100) Percentage of edge lengths necessary for recognition.
- **Effort Level** (0-4) Amount of effort to use when recognizing the prototype.
- **Min/Max Areas** Min/Max Area that allows recognition of the prototype.
- **Limit Position** Constrain recognition of the prototype to certain orientations.
- **Edge Weights** (0-100) Give importance to edges or regions of the prototype.
- **Assign Cameras** Change virtual camera association for the prototype.

These parameters are important in creating accurate recognition. If there are problems with recognition or orientation of recognized prototypes you must adjust these parameters until you can develop consistent recognition (*see images below*). (It may also help to increase the value of the V.MAX.TIME parameter.) **Edge Weight** assignment and **Min/Max Area** are the most useful in properly determining the orientation and recognized prototype when you are dealing with objects that are fairly close in area and appearance (*i.e. the widgets*)
After prototype training is completed it is necessary to store the prototypes to the hard disk. To store prototypes use the following command at the prompt:

\[ \text{Vstore filename.spec = prototype1, ..., prototypeN} \]

You can store multiple prototypes under one file name.

To load prototypes into memory for recognition use, type the following at the command prompt:

\[ \text{Vload filename.spec} \]

This will load all prototypes stored in the file into memory.

**Getting Recognized Prototype Information**

When an image is processed the vision system will automatically recognize defined prototypes as long as the switch V.RECOGNITION is enabled and there is a prototype model available in memory. When any region or prototype is recognized VQUEUE(cam.virt) will be created and provide information of all regions appearing in the current frame with information about their location and orientation (provided the correct switches are enabled). It will also list the names of
recognized prototypes. If display is set to allow overlay, a red overlay of the recognized prototype will be shown on the acquired image.

In order to determine the location of a recognized object the following command is used:

\[ \text{VLOCATE} \left( \text{cam.virt, mode} \right) \$\text{protoname}, \text{proto.loc} \]

\begin{tabular}{|l|p{12cm}|}
\hline
\textbf{cam.virt} & Virtual camera to use with VLOCATE function \\
\textbf{mode} & 0: removes a regions data from the queue \\
& 4: remove a hole’s data from the queue \\
\textbf{order} & 1: remove largest region from the queue first \\
& 2: remove smallest region from the queue first \\
\textbf{$\text{protoname}$} & A string that contains the name of the prototype to VLOCATE. Can be replaced with “prototype_name” \\
\textbf{proto.loc} & A transformation variable that defines the location of the recognized object in the camera field of view. (obj.loc) \\
\hline
\end{tabular}

\textbf{Note:} The variable proto.loc is a relative transformation and must be use in the following manner as shown previously:

\[ \text{global.loc} = \text{link2:to.cam[1]:proto.loc:grip.trans} \]

When the command:

\[ \text{VLOCATE} \left( \text{cam.virt, 2} \right) \text{“prototype_name”}, \text{proto.loc} \]

is performed successfully the recognized prototype will be removed from the queue of regions to process and the location in the field of view will be stored in the variable proto.loc. When the next \text{VLOCATE}() command is executed, the next recognized prototype is removed from the queue and proto.loc is updated with the new location information.

To locate and remove unrecognized regions replace “prototype_name” with “?”. When an image is processed, all unrecognized regions in the frame are give the name “?”. Upon a successful \text{VLOCATE}() of a prototype all the information about that prototype is available through the \text{VFEATURE}() function. This function stores values and is not a command. Its value information can be used for logical expressions or for calculations:

Example:

\begin{verbatim}
If \text{VFEATURE}(1) Then 
   Type “Successfully located”, \$\text{protoname} 
End
\end{verbatim}

The following is a list of useful \text{VFEATURE}() values and a brief description when using object recognition. There are a total of 50 values store in \text{VFEATURE}() and availability is dependent on enabled switches."

\begin{center}
\begin{tabular}{|c|c|}
\hline
\text{VFEATURE INDEX} & \text{Value} \\
\hline
\end{tabular}
\end{center}

12
When an image is processed the **VQUEUE***(cam.virt)* is created for the virtual camera associated with the image. It contains information about each region found in the current frame. The following command displays the current queue for the specified virtual camera:

```plaintext
VQUEUE(cam.virt)
```

The **VQUEUE**() function can be used in the following way:

```plaintext
WHILE VQUEUE(cam.virt) DO
  Vlocate(cam.virt, 2) $protoname, proto.loc
  ;code for each prototype location
END
```

This code will locate all recognized prototypes of “protoname” and store the location in **proto.loc**, updating the location and removing the instance from the queue upon a successful **VLOCATE**.( ).

**Note**: It is possible to get into an infinite loop if there are unrecognized regions that exist in the queue and they are not being removed. (*This is addressed in the example program*)

To remove an unrecognized region from **VQUEUE**() use the following command:

```plaintext
Vlocate(cam.virt, 2) “?”, unrecog.loc
```

This will remove one unrecognized region from the queue per execution.

**Using the Recognized Prototype Information**

**Developing the Global Location of a Recognized Prototype:**

After successfully locating a recognized prototype its position information is available as a location transformation **proto.loc**, which is defined relative to the field of view of the camera. The field of view of the camera in relation to **link2** is available as **to.cam** and is developed from vision calibration. Therefore it is necessary to define the location transformation **link2**. The **link2** transformation is a global transformation locating the link relative to the base frame. Its transformation must therefore be defined at the camera location where the frame was acquired and used to determine **proto.loc**. To define the link transformation you must get the joint values of the robot at the picture location, **pic.loc**.
Example:

For each camera position acquiring frames the following calculation of link2 is computed with the following commands:

\begin{verbatim}
HERE #pic.loc
DECOMPOSE jt[1] = #pic.loc
SET link2 = HERE:INVERSE(TOOL):RZ(-jt[4]):TRANS(,-,-jt[3])
\end{verbatim}

A precision location of where the picture was acquired, #pic.loc is defined, which contains each of the robot joint values. The values are decomposed into an array called jt[x]. Where jt[x] holds the value of joint x. In the example code, link2 is defined as the current position HERE, multiplied by the inverse TOOL transformation, rotated opposite the current gripper rotation jt[4] and translated by the opposite gripper Z location jt[3]. Now that the link2 transformation has been computed, the recognized object global location can be determined as:

\begin{verbatim}
global.loc = link2:to.cam[1]:proto.loc:grip.trans
\end{verbatim}

Now the location global.loc can be used to acquire the object. However, global.loc orients the gripper along the least axis of inertia and additional rotation may be needed do deal with alignment issues of the gripper. In the example program these additional transformations to deal with gripper alignment for object acquisition are aligngood and alignbad and contain values to account for any misalignment of the gripper and a 90° rotation of the gripper to grasp the part. As mentioned previously grip.trans is created during vision calibration and accounts for the transformation of the gripper in defining where the go to acquire the recognized object.

Note: The code from the example program uses modifications of the grip.trans transformation developed during vision calibration to account for slight discrepancies in the reported centroid location of the objects. Modifications are not always necessary but can be useful. (see Defining and Modifying Transformation and Location Variables)

**Helpful Commands and Functions**

**Defining and Modifying Transformation and Location Variables**

To list the values of a transformation or location variable:

\begin{verbatim}
listl variable_name
\end{verbatim}

To create a new transformation or location variable

\begin{verbatim}
point new.variable
\end{verbatim}

This will list the values of the variable and ask you if you want to change them.

You should not change the grip.trans transformations values. Instead copy them to another transformation and edit the values there:

\begin{verbatim}
set grip.trans2 = grip.trans
\end{verbatim}
point grip.trans2

It is often useful to define a TOOL orientation variable. The TOOL transformation variable is a default and is always available, however it may not be defined correctly. To redefine the TOOL transformation use the following commands:

- **point new.tool**
  - This will create a new transformation

- **TOOL new.tool**
  - This will create the new TOOL transformation with the information entered for new.tool

- **Listl TOOL**
  - This will display the new TOOL transformation values

**Defining Camera Positions**

The example program uses 8 camera positions to inspect a grid and ensure that the entire area is inspected for recognized parts. In order to cycle through the camera positions an array of precision locations was created. To create an array of precision locations of predefined camera positions use the following commands:

With the camera located in the desired position:

```
HERE #location1
#campos[1] = #location1
```

Repeat this for each new camera location. You will then have an array of camera positions available under the variable name `#campos[]` and can cycle through positions using a for loop:

```
For j = 1 to 8
  Move #campos[j]
End
```

**Additional Reference Information:**

2. Advanced Camera Calibration Program User’s Guide Ch. 4
3. Advanced Camera Calibration Program User’s Guide Ch. 6
4. Instructions for Adept Utility Programs p. 64
5. Adept Vision VME User’s Guide Ch. 3
6. Adept Vision VME User’s Guide Ch. 4 and Appendix A
7. Adept Vision VME User’s Guide Ch. 6
8. Adept Vision VME User’s Guide Ch. 7