

Adept startup procedure

1. Turn on the green rotary switch on the front panel
2. wait a long time.... :-)
3. When the intro page and prompt come up you can type "ena pow" at the "." prompt. You should here the pneumatics click.
4. Check that the workspace is clear and the end-effector tooling looks OK.
5. Now type "calibrate" to calibrate the servos. Again, this take a while.

Note: Once the machine has calibrated, you do not need to recalibrate it if the emergency stop switch is pressed to disarm the servos or if an error causes the power to become temporarily disabled. Just enable the power again and continue.

Teach pendant and coordinate systems

The Adept teach pendant supports World, Tool, Free and Joint coordinate systems. You press the Man button to toggle among these. Then select axes (XYZ or 123) and use the + and - speed pots to make it move. Free mode allows you to selectively move axes by hand and is very convenient for teaching.

The gripper can be opened and closed by selecting T1 and pressing the + or - "speed pot" keys.

When you get to a location you like, press the "Comp" button to transfer control back to the computer and type "**here myloc**" where myloc is the name of the location you are saving. You can also do this from within a program using the **TEACH** function. See section on User I/O for another solution.

Note that a location for the robot itself corresponds to the position and orientation (with respect to the world) of a coordinate frame embedded in the robot wrist. That means that a location will include a 180degree flip about the Yaw axis, so that Z is pointing *downward*. This means that if you ask the robot to move to a particular XYZ position in the world, you need to remember to put in the 180 degree flip before telling it to move. If in doubt, use the function **inrange** to check locations; if **inrange(location)** returns 0 the location is OK.)

Manuals and examples

The Adept has two very complete manual volumes (in yellow binders). Vol 1 has an overview of the operating system editor, etc. You'll want to read the section in Chapter 8 that deals with the see editor and debugger. Vol1 also has an index of all Adept V+ functions. You'll notice the language is sort of like Basic.

The functions themselves are in alphabetic order in Vol. 2 in section K.

There is also a green binder of "helpful hints." The section on common Adept commands is most useful. This binder also has a few program examples.

A bunch of useful example programs can be found on the hard disk in `c:\me319\demos`

The file system and loading, saving files

The Adept has a rather primitive DOS-like operating system. To move to a desired directory you set the current default directory. Example: `def d=c:\me319\grp3`

To display the contents of a disk directory the command is `fdir`

To display the contents of the RAM directory (kind of like a RAM disk) the command is `dir`.

When you exit the see editor (see below) the file is NOT saved to disk. You must save it by specifying something like `storep diskfilename.v2 = progname` where `progname` is the name of the function you've been working on and `diskfilename` is the name you want to have on the disk. The operating system will not let you overwrite files; you have to delete the original first.

The See editor and debugger

The see editor is a somewhat primitive screen editor, similar to "vi" in (old) Unix.

First `load` the file from disk. Then type "`see progname`" where `progname` is the program you want to edit. The arrow keys and backspace/delete keys work. To insert new text you have to hit the `insert` key first. Notice that the interpreter checks your syntax as soon as you hit "enter." If the line is OK it will be tabbed and uppercased. If there is something wrong there will be a question mark at the beginning.

I usually like to do program development with the editor and debugger active. Hit "shift debug" (Shift F11) on the keyboard and a second window will open up. If it asks you for a `task number`, type 0 (the default). The debugger window lets you run "monitor" commands and also step through the program.

I find that typing long commands at the regular monitor level is so frustrating that I usually create a little dummy program in the see editor and then I open the debugger so I can edit/modify lines and step through them at will.

Stepping through a program and checking things

In the debug window type "`prime progname,`" where `progname` is your program (not necessarily the same as the name of the file you loaded that contains the program). An arrow cursor `->` will appear at the top of the screen. By hitting "control G" you can step through the program, executing as you go. The arrow keys can move you backward and forward to different places in the program. Whenever you need to change something, hit "edit" (F11 *without* Shift pressed) and you pop back into the `edit` window. Use the arrow keys to fix things and then go back to `debug` mode (hit shift F11).

At any point, if you want to check the value of a variable you can type "`listr myvariable`" in the debug window. Similarly, locations can be checked using "`listl myloc`" This is a good idea to do before telling the robot to move there.

You can also set *Breakpoints* with "Control B" (nuke 'em with Control N) and you can specify to *Proceed* until the next breakpoint with Control P.

Program versus monitor functions

The Adept has *program* and *monitor* functions. The latter can be executed directly by typing them in response to the monitor prompt. The former are meant to be called from programs, but can also be executed in the monitor by preceding them with the word "do," for example `do move loc1` (but see my comment above about how it's often easier to put them into a dummy program and then execute them using "control G").

Locations and moving around

The standard Adept move command is "**move myloc**" where myloc is a location (4x4 transformation) data type. Locations are predefined data types in V+, like integers, strings and reals. To set a location equal to something you need to precede it with the keyword "set," for example:

```
set myloc1 = myloc2.
```

To see the contents of a location on the screen type "**list1 myloc**" in the monitor. If you want to access the individual elements of a location from within a program, you have to first convert it into an array: **decompose myarray[] = myloc**. Then you can access the array elements. For example, the X component of the array is **myarray[0]**.

To *create* a transform from a collection of elements you use the trans() function :
SET MYLOCATION = TRANS(MYARRAY[0],MYARRAY[1],Z,0,180,ROTZVALUE). Note that locations have 6 elements because V+ is general purpose robot language, even though the Adept only has four axes. To check if a location that you've assembled is reachable, use the **INRANGE(myloc)** function. This returns zero if the location can be reached.

Locations can be multiplied as 4x4 homogeneous transforms. The rules of matrix algebra apply and a built-in inverse function exists. For example, if the offset between the robot wrist and a grasped block is given by the transform (i.e., location) **offset** then the following set of commands will cause the robot to move to a new location that has the effect of making the grasped block rotate 90 degrees about its own centerline:

```
HERE ROBOT                ;the current robot wrist location
SET BLOCK = ROBOT:OFFSET  ;block location in the world
SET BLOCKROT = RZ(90.0)   ;create a transform for 90deg Z rotation
SET NEWROBOT = ROBOT:OFFSET:BLOCKROT:INVERSE(OFFSET)
```

Don't forget that to send the Adept somewhere, you have to move it to a location that also includes the 180degree flip. For example, to drive the wrist to the position {x=50,y=30,z=700} and an orientation of theta=30 degrees anticlockwise in world coordinates you would say

```
SET NEWLOC = TRANS(50,30,700,0,180,-30} ;minus 30 because Z is downward now
```

On the Adept, it is often easiest to figure out intermediate coordinate frames in Z upward world coordinates and then add the 180deg Yaw flip at the end. When the Adept is in tool mode, you will not that the coordinate frame is rotated 180 degrees about the Y axis (pitch angle = 180) so that the Z axis points *downward*.

The *speed* of motion is set using the SPEED command. The speed is a percentage of 100. Used by itself, a command like **speed 10** only affects the *next* robot motion in your program. To affect all future motions (until you reset the speed) use a command like **speed 15 always**.

User input and output

The Adept provides the standard read and write commands that you would expect. The keyboard/monitor and the teach pendant are both treated as general purpose serial I/O devices. They must be "attached" to use them. The default is that the keyboard and monitor are attached. It is also useful to **attach** the teach pendant when you have a program that asks a user to do something at the teach pendant.

The following set of commands, taken from the examples given at the end of the description of the **pendant()** function in the Adept manual, VOI2., attaches the teach pendant, writes a message to the LCD panel on it, and waits for the user to hit the "**done**" button on the teach pendant before continuing:

```

TYPE /B, "Get ready to teach the first point"
ATTACH (1) ;attach the teach pendant device and detach the keyboard
DETACH (0)
KEYMODE 8 = 2 ; event will be the "done" button pressed
DO
WAIT
UNTIL PENDANT(8) ; wait until pendant(8) returns true.
DETACH(1)
ATTACH(0)
HERE LOC1

```

Vision on the Adept

The Adept robot has a comprehensive vision package. The vision commands are described in the large Vision binder and calibration is described in a small spiral-bound booklet titled Advanced Camera Calibration Program User's Guide.

Useful vision settings

1. Make it so that graphics commands are in millimeters not pixels (e.g., `dline()`)
`parameter v.scale.mode = 2`
`;d.scale.mode?`
2. If you don't enable this you get the wrong result for centroid of feature!
`enable v.centroid`
3. To get the Vision window use middle mouse button at top left. Then, using the menu bar in in the vision window you can set switches & things. You probably want to set `vdisplay 2,1` and set to `acquire & process`

The most useful vision commands

4. Basic picture taking sequence:
`cam = 1`
`vpicture(cam) ; take picture`
`vlocate(cam,2,1) "?" ,vis.loc ; locate a blob, put location`
`; in "vis.loc"`
`vfeature(cam) ; get blob features`
5. Get the X and Y coords of vis.loc (alternative is to use `decompose()`)
`blobx = DX(vis.loc)`
`bloby = DY(vis.loc)`

Other tips:

- You can set breakpoints in the debugger mode with `^B` ("*control b*") and nuke 'em with `^N`
- You can execute forward several steps to the next breakpoint with `^P` ("*proceed*")
- You can execute a single step with `^G` (*control g*).
- You can hop between programs *copying* and *pasting* in the See editor using the command `N <program name>` to get to the new program and then `COPY` desired lines. Then `^N` again
- to get back to you previous program and `PASTE` the lines.
- You can `LOAD` previously saved camera calibration data using `call load.area()` in your program (see section 8.4, p. 40 of the Advanced Camera Calibration Program User's Guide for details).

Example Vision Program

The following is a vision tutorial program that you will find in c:\me319\demos\newvdemo.v2 on the Adept hard drive.

```
.PROGRAM v.demo()
;*****
;
; Short Vision demo for the Adept.
; Modified 4.20.97 for new camera -mrc
;
;*****

;      set approximate grip transformation ( angular offset )
      SET grip = TRANS(,,,180,20)

;      correct to.cam to account for incorrect z value
      SET to.cam.2 = to.cam:TRANS(,,100)
; Set the gripping height for a block
; Something is still not quite right here... should probably recalibrate
; camera and manually tell it the correct tabletop height. 4.20.97 -mrc
      gripheight = 734

; Common Vision switch settings.
      ENABLE V.BINARY
      ENABLE V.BACKLIGHT
      ENABLE V.SHOW.BOUNDS
      ENABLE V.CENTROID
      ENABLE V.2ND.MOMENTS

      PARAMETER V.FIRST.COL = 1
      PARAMETER V.LAST.COL = 512
      PARAMETER V.FIRST.LINE = 1
      PARAMETER V.LAST.LINE = 484

; These values change with lighting etc.
; Click middle mouse button on upper left window corner
      PARAMETER V.GAIN = 256
      PARAMETER V.OFFSET = 10
      PARAMETER V.THRESHOLD = 11
      PARAMETER V.2ND.THRESH = 0
; The next settings are fussy. Depending on what the previous values
; are (type 'par' in monitor to see), you will have to
; enter them in a certain order. Max.Area can never be set lower
; than the present value of Min.Area, which can never be less
; than Min.Hole.Area and so forth.
      PARAMETER V.MAX.AREA = 40000
      PARAMETER V.MIN.HOLE.AREA = 500
      PARAMETER V.MIN.AREA = 10000
      PARAMETER D.SCALE.MODE = 2

      cam = 1
      thresh = 11
      backlight = TRUE
; This calibration was done on 4.20.97
      $file = "c:\cam_cal\area7.dat"

; If stepping through this program, use ControlP here (not ControlG) to
; execute all the way through it down to the prompt '10' below.
; Note that this file was save along with v.demo() in vdemo.v2
```

```
CALL load.area($file, cam, thresh, backlight, to.cam, cam.cal[],
pmm.to.pix[,], pix.to.pmm[,], pmm.to.mm[,], mm.to.pmm[,], $error)
```

```
IF $error > "" THEN
```

```
    TYPE /C2, "There was an error loading camera calibration!"
    GOTO 100
```

```
END
```

```
10PROMPT "Is the camera over the block to be lifted?", $ready
```

```
IF ($ready == "y") OR ($ready == "Y") THEN
```

```
    VDISPLAY -1, 1; set display to binary
    VPICTURE -1; Take a picture and process it
    VLOCATE (1, 2) "?", vis.block; Locate first fiducial
```

```
; This is a good place to insert a break point (control B; to remove
; use control N) so you can see what the camera sees
```

```
IF VFEATURE(1) THEN
```

```
    TYPE /C2, " Object Found  "
ELSE
    TYPE /C2, " No Object Found ! "
    TYPE /C2
    PROMPT "Try again", $again
    IF $again == "y" THEN
        GOTO 10
    ELSE
        GOTO 100
    END
END
```

```
ELSE
```

```
    TYPE /C2, " Position Camera over Object "
    GOTO 100
```

```
END
```

```
xc = DX(vis.block) ; x coordinate of block centroid
```

```
yc = DY(vis.block) ; y coordinate of block centroid
```

```
; Draw 2 lines through the supposed centroid of the object.
```

```
DLINE xc, yc-5, xc, yc+5
```

```
DLINE xc-5, yc, xc+5, yc
```

```
; Get the angle of the major axis of the blob
; This is actually NOT a very accurate way to get the orientation
; of a 'compact' shape like a square block. Get corner points or
; edges would be better...
```

```
angle = VFEATURE(48)
```

```
; Draw a line corresponding to computed block major axis
```

```
xa = 20*COS(angle)
```

```
ya = 20*SIN(angle)
```

```
DLINE xc-xa, yc-ya, xc+xa, yc+ya, 1, 10
```

```
; Now do some coordinate transformations so that we can try to
; grasp the block that we have just found.
; Obtain current robot wrist location in joint coordinates
; Use this to make a frame corresponding to current camera location
  HERE #block_loc
  DECOMPOSE joint[1] = #block_loc
  SET link2 = HERE:RZ(-joint[4]):TRANS(,,-joint[3])

; Define grasp location (we hope!)
; This is another good place to set a breakpoint :-)
  SET over.block = link2:to.cam.2:vis.block:RZ(angle):grip

; If everything looks cool, try to grasp the block and rotate it
  IF INRANGE(over.block) == 0 THEN
    APPROX over.block, 50
    BREAK
    OPENI
    MOVE over.block
    CLOSEI
    DEPARTS 50

    SET new.block = over.block:RZ(180)
    APPROX new.block, 50
    MOVE new.block
    OPENI
    DEPARTS 50
    MOVE #block_loc
  ELSE
    TYPE "Ooops.. Could not reach computed block location!"
  END

100STOP
.END
```