Startup and Shutdown Procedure

Please follow these startup and shutdown procedures

to avoid possible damage to the Z axes and

to avoid corrupting the MacRAIL environment!

Startup

1. Sign in the log book
2. Turn on red rocker switches: MAIN, AV90
3. Turn on key on controller (you should hear the standard Mac boot tone -- if it doesn't boot, try again)
4. Open the orange RobotWorld 2.0 Mac folder (if not already open) and double click the blue MacRail icon. This launches the RAIL interpreter and loads the Robot World files. When done you should see a window with a schematic view of the workcell and a top menu bar with: File Workcell Motion Window
5. Now turn on the red AI816 switch in the cabinet.
6. Under Workcell select Power Up Sequence -- this should cause the Z and theta servos on the 2 robots to initialize and the compressed air to come on. If you get an error message see one of the Me319 staff.
7. Select Quit under the File menu. This dumps you back into the RAIL environment where you can execute commands, load files, etc. To get back to the Workcell application at any time type `menu` <enter> in the RAIL worksheets.
8. Set speed of the devices to something safe (like 20)[1]. Have fun!

Shut down

1. Be sure that the units have their Z axes raised
2. Send them `home!`
3. Quit RAIL (Quit under the File menu)
4. Now turn off the AI816 switch
5. Select Shutdown under Special on the Mac.
6. Turn off key switch, AV90, and MAIN switches in the cabinet
7. Log out in the book.

- ***Please do not alter the Mac control panel settings*** (e.g. colors) -- this can give unexpected results with the vision board. Also do not add any new software or upgrades without consulting Cutkosky.
- ***Please be gentle with the Z and theta axes*** -- they are fragile and have broken several times!

[1] **But NOT BELOW 10, this seems to hang up the AI816 processor for unknown reasons...**

Robot World Introduction

For the first tutorial we will cover:

- setup, manuals etc.
- RAIL introduction and how the interpreter functions
- file and user I/O
- basic motion commands MOVEALL(), MOVES(), APPROACH() etc.
- Transforms and how to use them
- random things to watch out for...

Manuals

fat green binder: RAIL manual and some applications programs
medium blue binder: Vision applications
thin brown or red binders: Robot-World instructions and documentation -- *most of what you need to read is in here.*

Unfortunately some of the best stuff (e.g., the calibration software and the vision debugger) is not very well documented.

Coordinates

A location in RAIL describes a cartesian frame and orientation in space. You can also specify a location by typing in a four-element vector, [X,Y,Z,theta], or a six-element vector, [X,Y,Z,a1,a2,a3], which the RAIL interpreter will immediately convert to a location data type.[2] The six-element vector is a carryover from the fact that RAIL is a general-purpose language for robots with any number of axes. The angles a1, a2, a3 are Euler angles. Since RobotWorld has only 4 axes, the interpreter will map them to the four axes. The results are not always intuitive and I therefore recommend using the 4-element vector when specifying locations.

Note that the home position of device #man2 is NOT [0,0,0,0,0,0] or [0,0,0,0] as you might expect

```
;Get the current position of device #man2 and put it
;in the location variable "two_home"
;(Most RAIL functions return "0" if no error.)
position(#man2,two_home)
0
;if you type a variable the interpreter will evaluate it
;and print it out on the screen
one_home
ONE_HOME = [ 1.21, 0.16, 4.01, 0.000, 0.000, -93.164 ]
```

So Y is essentially zero but X is offset by just over an inch. Why? (Good question).

Approx limits of travel in the workspace are as follows. If you try to move beyond them you will et error messages:

Xmin: 2.25 inch (approx. home position of device #man2 and #cam1)

Xmax: ~48 inch (approx. home position of device #man1)

Ymin: 2.25 inch (approx. home position of devices 1,2)

Ymax: ~30 inch (approx. home position of device #man1)

Zmin: 0 inch (fully extended)

Zmax: 4 inch (fully retracted)

---

2. (Ihe internal representation is a 4x4 homogeneous transformation.)

theta_min: -180deg

theta_max: +180deg (theta home for devices #man1,#man2 is 0) Note, angles beyond +/-180deg will be mapped to that range. Also, -90 degrees is the point at which the robot will switch to take "the other way 'round" when reaching a location. This is something to watch for when working in confined spaces.

Moving around:

For the most part you will use `moveall(device,location)`. This type of move retracts the Z axis to a safe height before taking the robot to its final XYZ, and theta.

To move only a particular axis you can specify `emove(device,location,#axisflag)`

where `#axisflag` is a keyword given on page ?? of the RobotWorld2.0 manual. For example, the command `emove(#man2,location,#T_Axis)`will only rotate the wrist of #man2. In this case, all values of location must be valid (within the workspace), but only the theta argument is acted upon.

There are also some built-in commands like `rotate(device,location)` but they do not appear to work as reliably as `emove()` in the current beta version of the RobotWorld software.

There are various other flavors of move commands useful for special applications. The most useful are the `moves()` and similar commands that enforce synchronization.

Transformation manipulations:

Suppose that the robot is at the location `robot_loc` and the relative transformation between the robot and a grasped object in the hand is `grasp_transform`. Then the location of the grasped object with respect to the world is `grasp_world = robot_loc:grasp_transform`.

Another example: Find the relative transformation between two objects which have the locations loc_1 and loc_2, respectively, in world coordinates. The relative transform would be defined by the relationship: `loc_1:rel_transform = loc_2`. Therefore,

`rel_transform = inverse(loc_1):loc_2`.

If you are not getting the results you expect when doing transform manipulations in RAIL check that the a1, a2 and a3 angles are what you think they are. If one is reversed by 180deg. the meanings of X,Y,Z will be very different!

The following is from a RAIL file called "locs and coords" in the Cutkosky directory

```
speed(1,20)
accel(1,50)

loc1= [10,10,2,90]
;this executes a sequence of depart, move, approach
moveall(1,loc1)

;bring it back home
home(1)
depart(1,[0])
rotate(1,[0])

;transform a location into a 7-element array. If this statement
;were made within a function you'd have to do an array declaration
;coord_array[7]
loc_to_coord(loc1,coord_array)
```

```
;Have to select both lines to make this 2-line statement work
write ('\n>coord_array x: ',coord_array[1,1],' y: ',coord_array[1,2],' z:
',coord_array[1,3],' theta_z: ',coord_array[1,6])

;Define a new location relative to loc1.  Where will it be?
loc2 = loc1:[2,2,0,-90]
loc_to_coord(loc2,coord_array)
write ('\n>coord_array x: ',coord_array[1,1],' y: ',coord_array[1,2],' z:
',coord_array[1,3],' theta_z: ',coord_array[1,6])

;Notice that X absolute DECREASES since we just moved +2,+2 inch in the rel-
ative frame to loc1

moveall(1,loc2)

;notice that z axis departs and approaches automatically.
```

Example Program

```
;*****************************************************************
;; Simple circle function (inefficient) to show use of relative
;; coord transforms     3-4-90  M. Cutkosky

;first bring everything home and set safe speeds
speed(1,20)
accel(1,50)
home(1)
depart(1,[0])
rotate(1,[0])

;center of circle and starting point
loc1= [10,10,2,90]
loc2 = loc1:[2,2,0,-90]
moveall(1,loc1)
moveall(1,loc2)

;define relative transformation from loc1 to loc2
rel_loc = inverse(loc1):loc2

;note that you don't format each element in a write statement
write('\n> rel loc: ',rel_loc)

;do a circle about loc1 by first rotating by theta, then moving
;out by rel_loc, then rotating
;back by negative theta, for each new location.

moveall(1,loc1)
maxsteps = 24
dtheta = 360/maxsteps
theta = 0.0
; Unlike in 'C', you don't have to explicitly increment the counter
FOR count = 0 TO maxsteps
     DO
     BEGIN
     rotz = [0,0,0,theta]
     next_loc = loc1:rotz:rel_loc:inverse(rotz)
     move(1,next_loc)
     write('\n> next loc: ',next_loc)
     theta = theta + dtheta
     END
```

User and file input and output

With RobotWorld2.0 software the much preferred way to do used IO is to use the built-in `DLOG()` functions covered toward the rear of the manual. There are functions for printing simple messages, obtaining yes/no input and obtaining specific numerical values. On the numerical value ones, be sure to initialize the returned result values (e.g., by setting them to 0 or 0.0) so that RAIL knows whether to treat them as reals or integers.

Files and functions can be loaded from the workspace either by selecting them and hitting "enter" or by using the FILE, LOAD commands in the menus. You can also load files from within programs by specifying the full path name: `load('ai90:RobotWorld2.0:Me319:Cutkosky:my-file')`. Make sure you have the pathname exactly right (I think it's case sensitive).

Rail also provides standard functions for opening/closing and reading/writing data in files. Note that weird things could happen if you write to a file whose window is open on the desktop and then you subsequently type into the same window...

Vision calibration.

This section still under construction for new RW2.0 software. However, the example on the next page is up to date.

Follow Startup procedure steps 1-6.

Under CALIBRATION you will see SYSTEM, MOBCAM, FIXCAM ROB.
If both robots have their default base plates with no tooling mounted you can specify SYSTEM and which calibrates everything. Otherwise first select MOBCAM. Then do the calibration for the fixed camera using the bracket as shown in class. Finally, if either of the robots has a tooling plate with no grippers you can specify ROB for those robots.

Occassionally calibration may fail due to lighting. If it fails you can specify "debug" and try modifying the threshold and minimum object size parameters. Then try it again. When all is done you are back to the image board. You can quit the ROBOTWORLD application to get into the RAIL environment.

The calibration routines set many threshold and scaling factors which are useful for accessing later. They are stored in a data structure called ROBSCHED.

Vision tutorial

Here is a vision tutorial program called VisionDemo97 that is in the ME319:Cutkosky directory.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 4.16.97 -mrc
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Set convenient directory path
directory = 'Ai90:RobotWorld2.0:me319:'

; Load arrays of flags and switches that set up image buffer and picture
; taking and processing settings. It's worth looking at this file so you
; understand what is going on.
load( concat(directory,'Cutkosky:Vision_settings') )

; declare array for holding some results (you may want different ones)
; See Section 3.3 of Chapter 3 for other options.
nfeatures = 4
array newfetarray[nfeatures]
newfetarray = {#xcent, #ycent, #area, #peround}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Basic procedure is first you TAKE a picture, then you PROCESS it.
; Update the vision buffer and set it to update with each new picture
buf_update(0)
buf_auto_update(0, on)

; TAKE & DISPLAY picture according to mypicarray[] (defined in
'Vision_settings')
; See Cha 2 in MacRail Vision manual.
pic_go_and_display(mypicarray, pichd)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; PROCESS the picture using the Connectivity package
; (Cha 3 in MacRail Vision) according to the settings in myprocarray[]
; (defined in the 'Cutkosky:Vision_settings' file)
c_sched_init(1)
c_sched_set(myprocarray, sched)
c_go(myprocarray, pichd)

; How many blobs did we find? Count ’em and highlight them.
nblobs = c_nblobs(pichd)
c_hilite_tree(pichd, #yellow)

; I'm declaring this array down here because I didn't know how many blobs
; we would find :-)
array blobres[nblobs,nfeatures]
c_results(pichd, newfetarray, blobres)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; CONVERT a blob centroid location to a world coordinate
; Let's suppose the first blob is the one we want...
chosen = 1
array pixarr[1,2]
pixarr[1] = {blobres[chosen,1], blobres[chosen,2]}

; Convert the pixel location of the center of the block
; into world frame coordinates (assumes camera is calibrated correctly)
position(#cam1, camloc)
dev_transform_at(#cam1, pixarr,camloc, wldloc)
```