CARTS-Demo94: Simulator Interface

March 1994

1 Manufacturing Task

The manufacturing task, as shown in Figure 1, is to assemble products composed of at most three parts (balls), having as goal the maximization of the value of products produced per time unit, given that parts and requests for products arrive in the system in a nondeterministic way.

There are two phases in the assembly process. In the first phase, parts are selected and transferred from an arrival table to an assembly table. During the second phase, parts are assembled into tubes, given the current requests for products. The process in this phase consists of grasping each part and transporting it from the assembly table to a tube by using a hand-arm system robot. Four products can be assembled simultaneously.

A graphic simulator was developed to show the system performance. Details of the simulation and the manufacturing environment are presented in the following section.

2 Simulator

The graphics simulator, shown in Figure 3, was written in C, using the XWindow System environment, and it is currently running in a Sparc 10 Workstation. It can operate in two modes: a **passive mode** that shows the actual real-time system running and presents analysis of its performance, and an **active mode** where the complete system is emulated from the arrival of parts to the tables and hand-arm system control. The **passive/active mode** is selected in compile time. In **passive mode**, the **Arrival** window discussed below is inactive. There is also a compile time define option for selecting which robot and manipulator are being used. The two possibilities are the P50 robot with the UTAH-MIT hand or the A4 robot with a gripper as manipulator, as shown in Figure 2.

The simulator is composed of five windows, as described below:

WorkCell is the initial window and contains the buttons that gives access [1] to the following four windows. The **Start/Stop** button starts/stops the simulation, and the **Quit** button ends the simulation.

[1]Buttons are activated by clicking the left mouse button on its window area.

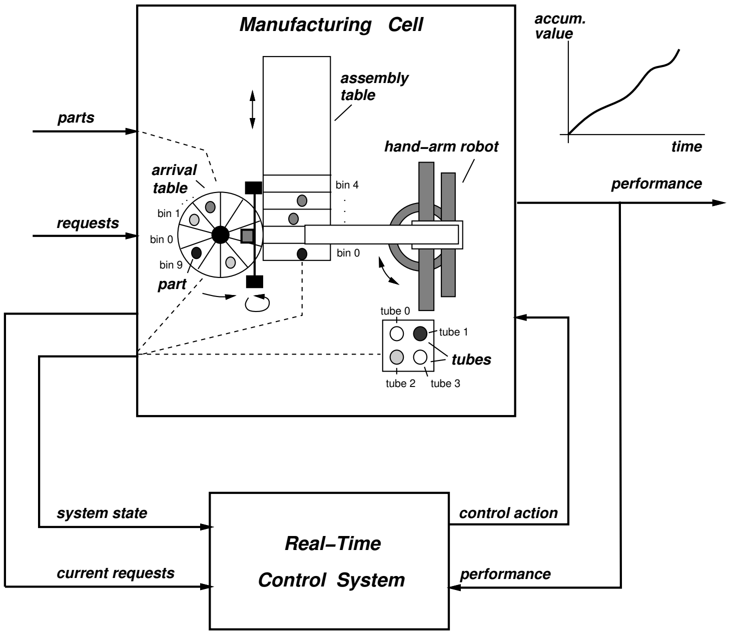

Figure 1: Overview of the manufacturing task

Draw button accesses the **Draw** window that shows the manufacturing cell. In the window, **Start/Stop** button starts/stops the window update, and the **Quit** button closes the window.

In/Out button gives access to the **Request/Arrival** window shown in Figure 4. The arrival of parts and request of products can be generated in a variety of ways at anytime during the execution of the task. The description of this window is presented in the following section.

Status gives access to the **Status** window that displays the requests and products produced.

Analysis button calls the **Performance Analysis** window that presents the system's performance.

2.1 Request/Arrival Window

It allows the selection of product requests and arrival of parts in several different modes. The available options are accessed by the following :

Request/Arrival button differentiates between request selection and arrival selection.

Manual/Auto button discriminates between requests or arrivals in **Manual** or **Auto** mode.

Blue, Red, or Black Ball button allows the selection of requests, or arrivals, if in **Manual** mode.

Poisson Distribution/General Distribution/Periodic/Customized are the available options for requests and arrivals in **Auto** mode. In **Customized** mode, requests or arrivals are produced by a description file.

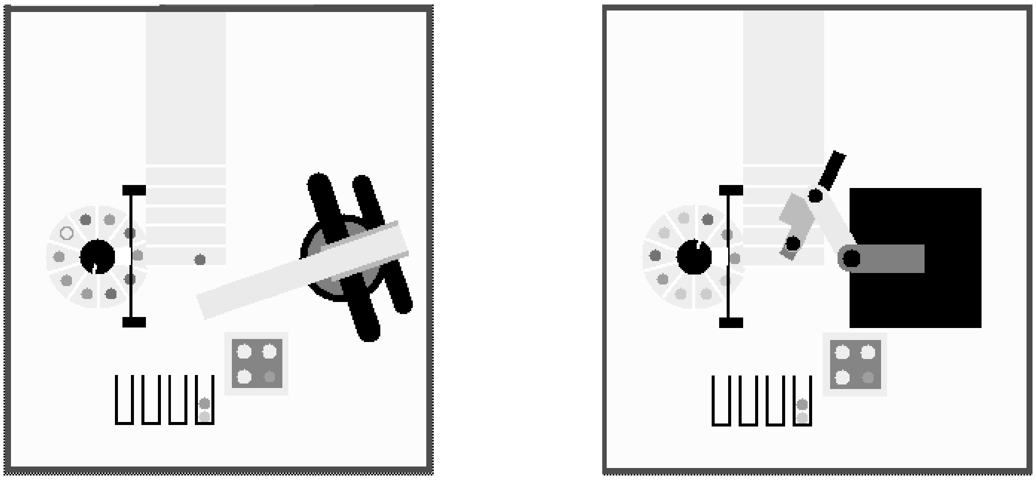

Figure 2: Manufacturing Cell considering P50-UTAH-MIT hand (left), and A4-Gripper (right)

Setup window presents a way of selecting the request or arrival in more detail. The options presented by this window are determined by previous selections, such as Request/Arrival, Manual/Auto, and Poisson Distribution/.../Customized.

Start/Stop button starts/stops the requests or the arrivals, if in **Auto** mode.

Status button reports the current product requests in the case of the **Request** window; the parts arrived so far, in the case of the **Arrival** window.

Quit button closes the **Request/Arrival** window.

2.1.1 Customized Arrival

The format of a customized arrival file should be as follows:

First line: number of arrivals (Maximum 300);

Following lines: each arrival entry [<arrival time> <ball color>].
The lines must be organized by increasing arrival time.
The <ball color> is in the range [0..2], [Blue,Red,Black] respectively.

Example:

```
6
0.1 0
0.9 1
1.5 2
1.6 0
```

```
1.7 1
1.8 2
```

The default filename is *arv.dsc*, and the directory is *xdemo94*.

2.1.2 Customized Request

The format of a customized request file should be as follows:

First line: number of requests (Maximum 100);

Following lines: each request entry

[<arrival time> <order> <partial> <products> <balls per product> <value> <deadline1> <deadline2> <ball colors>].

Lines must be organized by increasing arrival time.
The <order> flag means **in order** when 0, and **any order** when 1.
The <partial> flag means **total request** when 0, and **partial request** when 1.
<products> means the number of products [1..10].
<balls per product> means the number of balls [1..3] in each product.
The <value> can be in the range (0.0 .. 100.0].
<deadline1> and <deadline2> are relative to the arrival time and can assume values in the range [0.0 .. 240.0]
The <ball color> is each one of the <balls per product> color. It can assume values in the range [0..2], [Blue,Red,Black] respectively.

Example:

```
3
1.0 0 0 1 3 10.0  3.0 4.0 0 1 2
1.5 0 0 1 3 50.0  3.0 4.0 0 0 2
1.9 0 0 1 3 100.0 3.0 4.0 0 1 1
```

The default filename is *req.dsc*, and the directory is *xdemo94*.

2.2 Request Setup Window

This setup window presents options as:

Ordered/Any Order - **Ordered** specifies that the order of the balls has to be maintained, being **Any Order** the selection when the ordering is not relevant.

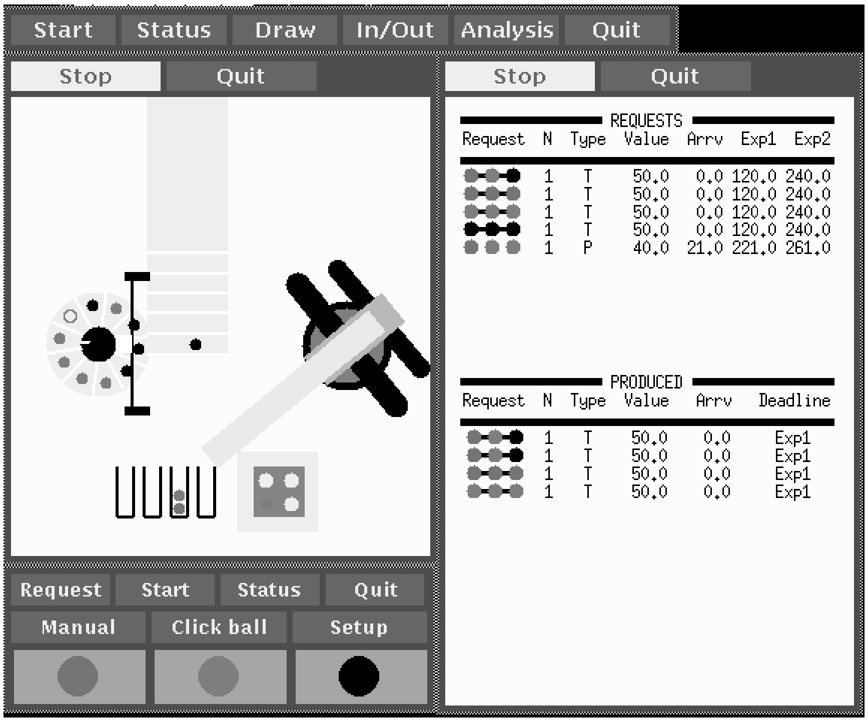

Figure 3: Graphics display of the simulator

Figure 4: Arrival and request window

Total Request/Partial Request - **Total Request** means the products only generates a value if all completed before the deadline. **Partial Request** accepts partial credit for requests partially fulfilled, such as number of products less than the number requested.

Products - number of products;

Balls/Product - number of balls in each product;

Value - value of each product;

Deadline1 - deadline for product production without depreciation. All deadlines are relative to the request arrival time (in seconds);

Deadline2 - product has no value if produced after this deadline. The product value suffers a linear depreciation between **Deadline1** and **Deadline2**;

Done - closes this window.

In the case of **Auto** mode, other options, such as **Rate** (number of requests per second), **Standard Deviation**, **Mean**, and **Period** appears depending on the **Auto** mode option. In **Customized** mode, only the name of the description file is necessary.

2.3 Arrival Setup Window

This setup window presents options as:

Value - value of an arrival (ball);

Deadline - expiration time of a ball;

Done - closes this window.

In the case of **Auto** mode, other options, such as **Rate** (number of arrivals per second), **Standard Deviation**, **Mean**, and **Period** appears depending on the **Auto** mode option. In **Customized** mode, only the name of the description file is necessary.

3 Interfacing to the System

To run and interface to the simulator as well as to the real system a variety of functions are provided.

<u>**NOTE:**</u> Code (real system or simulation) can be run on machines in the robotics lab <u>**ONLY**</u> while physically present in the lab !

<u>**Additional note:**</u> **ANY CODE** in directory **carts/xdemo94/main** and **ANY LIBRARY** used by a file in this directory can <u>**ONLY**</u> be changed by the robotics lab personnel. If any change is required, contact araujo@rabbit or huber@rabbit.

3.1 Description of the Control Actions

The following is the list of functions available as control actions.

```
/*========================== Linear Table ===================================*/

int move_ltable_bin(/* int bin */);
    /* aligns a given bin of the linear table with the flapper and returns */
    /* 0 if successful or -1 if the bin does not exist.                    */

int move_ltable_empty();
    /* aligns the closest empty bin with the flapper and returns the bin   */
    /* number if successful or -1 if no empty bin was found.               */


/*======================== Linear Table Camera===============================*/

int fball_ltable_bin(/* int bin */);
    /* determines the precise location of the ball within the given bin   */
    /* and returns 0 if a ball of the expected color was found or -1 if    */
    /* either the bin does not exist or no such ball was found.            */

int fball_ltable_color(/* int color */);
    /* determines the precise location of the first ball of a given color  */
    /* on the linear table and returns the bin number if successful or -1  */
    /* if no such ball is on the table.                                    */


/*============================= Flapper ====================================*/

int flap_bin(/* int bin */);
    /* flaps the ball in the given bin of the round table onto the linear  */
    /* table and returns 0 it successful or -1 if either the bin does not  */
    /* exist, no ball was detected in this bin, or no empty bin of the     */
    /* linear table is aligned with the flapper.                           */

int flap_color(/* int color */);
    /* flaps the closest ball of the given color onto the linear table and */
    /* returns the its bin number on the round table or -1 if either no    */
    /* ball of this color exists or no empty bin of the linear table is    */
    /* aligned with the flapper.                                           */


/*============================= Robot ======================================*/

int reach_to_bin(/* int bin */);
    /* moves the robot to the given bin and picks up the ball in this bin. */
    /* Returns 0 if successful or -1 if the position is not reachable, the */
    /* bin does not exist or no ball was detected in the given bin.        */

int reach_to_color(/* int color */);
    /* moves the robot to the first bin with a ball of the given color and */
    /* picks it up. Returns the bin number of this ball if successful or   */
    /* -1 if no ball of the given color was detected or its location is    */
    /* not reachable.                                                      */
```

```
int deliver(/* int tube */);
    /* moves the robot to the given delivery tube and drops the ball.      */
    /* returns 0 if successful or -1 if the tube does not exist, is        */
    /* unavailable, contains an already completed product, or is not       */
    /* reachable.                                                          */


/*############################################################################*/
/*                        Resource conflicts                                  */
/* The following pairs of functions can not be executed in parallel :         */
/*                                                                            */
/*   move_ltable   and   fball_ltable                                         */
/*   move_ltable   and   flap                                                 */
/*   move_ltable   and   reach_to                                             */
/*   fball_ltable  and   reach_to                                             */
/*   fball_ltable  and   deliver                                              */
/*   reach_to      and   deliver                                              */
/*############################################################################*/
```

3.2 Description of Tables Functions

This is the description of the available functions to access global memory structures relevant to the manufacturing task.

```
#define WCE_RTABLE 20.0 /* worst case time for one turn of the round table */
#define BCE_RTABLE  5.0 /* best case time for one turn of the round table */

typedef struct _Tube_Content {
  int
    nballs,             /* number of balls in tube */
    color[4];           /* ball colors in order */
  double
    value,              /* value of intended product */
    exp_time;           /* expiration time */
} Tube_Content;

typedef struct _Request {
  int
    id,                 /* ?????? */
    order,              /* in order = 0  | any order = 1 */
    total_req,          /* total_req = 0 | partial_req = 1 */
    nprod,              /* number of products */
    status,             /* empty=0 in_process=1 */
    nballs,             /* number of balls in product */
    ball_color[4];      /* ball colors in order of assembly */
                        /* ball color Blue, Red, Black */
  double
    value,              /* value of product */
    arv_time,           /* arrival time of the request */
    exp_time1,          /* expiration time1 */
    exp_time2;          /* expiration time2 */
} Request;
```

```
typedef struct _Product {
  int
    order,              /* in order = 0  | any order = 1 */
    total_req,          /* total_req = 0 | partial_req = 1 */
    nprod,              /* number of products */
    status,             /* empty=0 in_process=1 */
    nballs,             /* number of balls in product */
    ball_color[4];      /* ball colors in order of assembly */
                        /* ball color Blue, Red, Black */
  double
    value,              /* value of product */
    arv_time,           /* arrival time of the request */
    exp_time1,          /* expiration time1 */
    exp_time2,          /* expiration time2 */
    finish_time;        /* product finish time */
} Product;


/* number of ball colors :           */
/*    #define NBALL_COLORS    3       */
/* Ball colors :                     */
/*    #define BLUE_BALL       0       */
/*    #define RED_BALL        1       */
/*    #define BLACK_BALL      2       */


/*========================== Round Table ===================================*/

/* number of bins on round table :    */
/*    #define RBINS           10      */
/* status of round table bins :       */
/*    #define FREE            0       */
/*    #define UNDETECTED      1       */
/*    #define DETECTED        2       */

int current_bin_rtable();
    /* returns the bin number currently in front of the flapper.          */

int get_color_rtable(/* int bin, int *color */);
    /* reads the color of the ball in a given bin and returns the either  */
    /* the status of the bin or -1 if the bin does not exist.             */

int get_atime_rtable(/* int bin, double *arv_time */);
    /* reads the arrival time of the ball in a given bin and returns      */
    /* either the status of the bin or -1 if the bin does not exist.      */

/*========================== Linear Table ==================================*/

/* number of bins on linear table :   */
/*    #define LBINS           5       */
/* status of linear table bins :      */
/*    #define FREE            0       */
/*    #define UNDETECTED      1       */
/*    #define DETECTED        2       */

int current_bin_ltable();
    /* returns the bin number currently aligned with the flapper.         */
```

```
int get_color_ltable(/* int bin, int *color */);
    /* reads the color of the ball in a given bin and returns the either  */
    /* the status of the bin or -1 if the bin does not exist.             */

int get_atime_ltable(/* int bin, double *arv_time */);
    /* reads the arrival time of the ball in a given bin and returns      */
    /* either the status of the bin or -1 if the bin does not exist.      */


/*========================== Delivery Table ================================*/

/* number of bins on delivery table : */
/*    #define DBINS           4       */
/* status of delivery table bins :    */
/*    #define EMPTY           0       */
/*    #define IN_PROCESS      1       */
/*    #define DONE            2       */
/*    #define UNAVAILABLE     3       */

int get_tube(/* int tube, Tube_Content *content, double *arv_time */);
    /* reads the content of a delivery tube and the arrival time of the   */
    /* first ball, and returns either the status of the tube or -1 if the */
    /* tube does not exist.                                               */

int get_tube_req(/* int tube, Tube_Content *content */);
    /* reads the request assigned to a given delivery tube and returns    */
    /* either the status of the tube or -1 if the tube does not exist.    */

int set_tube_req(/* int tube, Tube_Content *content */);
    /* assigns a request to a given delivery tube and returns 0 if        */
    /* successful or -1 if the tube does not exist.                       */

int set_tube_done(/* int tube */);
    /* marks the product in a given tube as done and returns 0 if         */
    /* successful or -1 if the tube does not exist.                       */

int sim_set_tube_empty(/* int tube */);
    /* if in simulation mode this routine marks a given tube as empty     */
    /* and returns 0 if successful or -1 otherwise.                       */


/*============================= Requests ===================================*/

/* maximum number of requests :    */
/*    #define MAXNREQ         10   */
/* status of requests :            */
/*    #define EMPTY           0    */
/*    #define IN_PROCESS      1    */
/*    #define DONE            2    */
/* order for requests :            */
/*    #define IN_ORDER        0    */
/*    #define ANY_ORDER       1    */
/* total_req for requests :        */
/*    #define TOTAL_RQ        0    */
/*    #define PARTIAL_RQ      1    */
```

```
int set_request(/* Request *req */);
    /* enters a new request and returns the request number if successful  */
    /* of -1 if there are too many requests.                               */

int get_request(/* int num, Request *req */);
    /* reads a given request and returns its status or -1 if the request  */
    /* number does not exist.                                              */

int set_request_done(/* int num */);
    /* marks a request as done and returns either 0 or -1 if the request  */
    /* does not exist.                                                     */

int set_request_in_process(/* int num */);
    /* marks a request as in process and returns either 0 or -1 if the    */
    /* request does not exist.                                             */

int set_request_empty(/* int num */);
    /* marks a request as empty and returns either 0 or -1 if the         */
    /* request does not exist.                                             */


/*============================= Product =====================================*/

/* maximum number of products :    */
/*    #define MAXNPROD        10   */
/* order for product request :     */
/*    #define IN_ORDER        0    */
/*    #define ANY_ORDER       1    */
/* total_req for product request : */
/*    #define TOTAL_RQ        0    */
/*    #define PARTIAL_RQ      1    */

int set_product(/* Product *prod */);
    /* enters a new product in the list of done products, keeping always  */
    /* MAXNPROD most recent list entries, and returns the entry number.    */


/*============================== Timing =====================================*/

int get_time(/* double *time */);
    /* reads the current system time and returns 0 upon completion.       */

int time_to_flap(/* int bin, double *min_time, double *max_time */);
    /* gives a window which includes the time when the actual flap of a   */
    /* ball in the given bin of the round table can be made and returns 0 */
    /* if successful or -1 if the bin does not exist.                      */
```