5

Actuators and Sensors

Solution to Problem 5.1

At steady state, (5.1)–(5.4) can be written in the time domain as

$$v_a = R_a i_a + v_g \tag{S5.1}$$
$$v_g = k_v \omega_m \tag{S5.2}$$
$$c_m = F_m \omega_m + c_r \tag{S5.3}$$
$$c_m = k_t i_a \tag{S5.4}$$

where

$$v_a = C_i(0) G_v (v_c' - k_i i_a). \tag{S5.5}$$

Combining (S5.1), (S5.2), (S5.5) with $k_i = 0$ and $K = C_i(0)G_v$ gives

$$K v_c' = R_a i_a + k_v \omega_m$$

and yet, using (S5.3) and (S5.4) with $c_r = 0$, it is

$$K v_c' = \left(\frac{F_m R_a}{k_t} + k_v \right) \omega_m.$$

If $F_m \ll k_v k_t / R_a$, then

$$\omega_m \approx \frac{K}{k_v} v_c'.$$

If instead $k_i \neq 0$, on reduction of (S5.1)–(S5.5), it is

$$K v_c' = \left(\frac{R_a + K k_i}{k_t} \right) c_m + k_v \omega_m.$$

If $K k_i \gg R_a$, then

$$c_m \approx \frac{k_t}{k_i} \left(v_c' - \frac{k_v}{K} \omega_m \right).$$

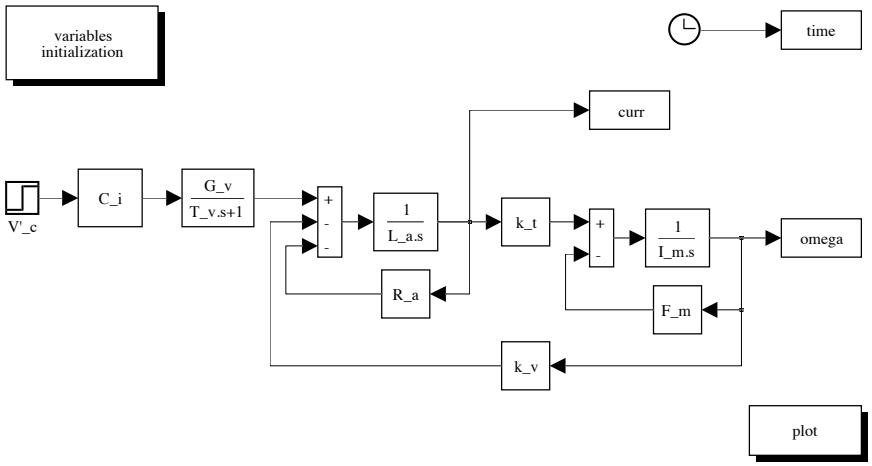

Fig. S5.1. SIMULINK block diagram of electric servomotor

From the block scheme in Fig. 5.3, the output can be computed via the closed-loop transfer functions; namely, the input/output and disturbance/output transfer functions, i.e.,

$$\Omega_m = \frac{\dfrac{Kk_t}{R_a(sI_m + F_m)}}{1 + \dfrac{k_v k_t}{R_a(sI_m + F_m)}} V_c' - \frac{\dfrac{1}{sI_m + F_m}}{1 + \dfrac{k_v k_t}{R_a(sI_m + F_m)}} C_r.$$

If $F_m \ll k_v k_t / R_a$, then

$$\Omega_m = \frac{\dfrac{K}{k_v}}{1 + s\dfrac{R_a I_m}{k_v k_t}} V_c' - \frac{\dfrac{R_a}{k_v k_t}}{1 + s\dfrac{R_a I_m}{k_v k_t}} C_r.$$

Finally, from the block scheme in Fig. 5.4, it is straightforward to compute the output as

$$\Omega_m = \frac{\dfrac{k_t}{k_i F_m}}{1 + s\dfrac{I_m}{F_m}} V_c' - \frac{\dfrac{1}{F_m}}{1 + s\dfrac{I_m}{F_m}} C_r.$$

Solution to Problem 5.2

With reference to the scheme in Fig. 5.2 ($k_i = 0$ and $C_r = 0$) and the given data, the resulting SIMULINK block diagram is shown in Fig. S5.1. The servo-

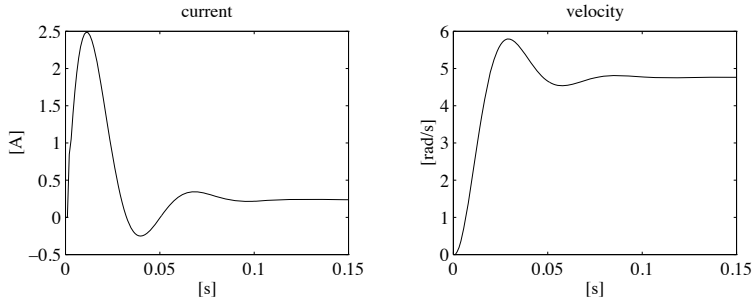

Fig. S5.2. Time history of current and velocity step response for an electric servomotor

motor is simulated as a continuous-time system using a variable-step integration method with a maximum step size of 1 ms. Notice that the amplifier time constant is too small to produce appreciable effects at this sampling time.

The files with the solution can be found in Folder `5_2`.

The resulting current and velocity responses to a unit step voltage input V_c' are shown in Fig. S5.2. It can be seen that the steady-state velocity slightly deviates from the value of 5 rad/s as in (5.7), because of the presence of mechanical friction (F_m). This also implies that the steady-state current is different from zero.

Solution to Problem 5.3

By closing a current loop with large loop gain, the current response is expected to become much faster than the velocity response. Velocity plays the role of a disturbance for the voltage-to-current closed-loop system. Hence, it is worth choosing a PI control structure for $C_i(s)$ in order to obtain a null steady-state error, i.e.,

$$C_i(s) = K_I \frac{1 + sT_I}{s}.$$

With the given data, by setting $\Omega_m = 0$, the closed-loop input/output transfer function ($T_v \approx 0$) is

$$\frac{I_a}{V_c'} = \frac{1 + sT_I}{1 + s\dfrac{0.2 + K_I T_I}{K_I} + s^2 \dfrac{0.002}{K_I}}.$$

Since the settling time within 1% is given by

$$t_s = \frac{4.6}{\zeta \omega_n},$$

either the damping ratio or the natural frequency can be imposed. Choosing $\zeta = 0.7$ with $t_s = 2$ ms yields

$$K_I = 21592 \qquad T_I = 0.000416.$$

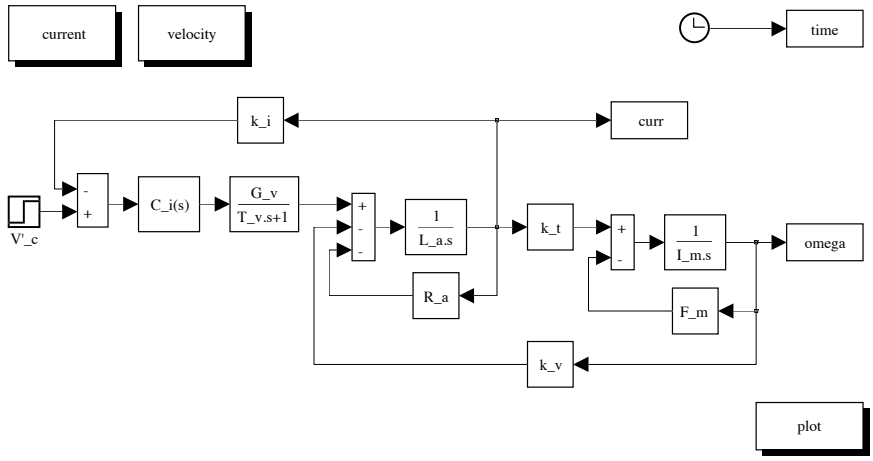

Fig. S5.3. SIMULINK block diagram of electric servomotor with current feedback loop and PI control

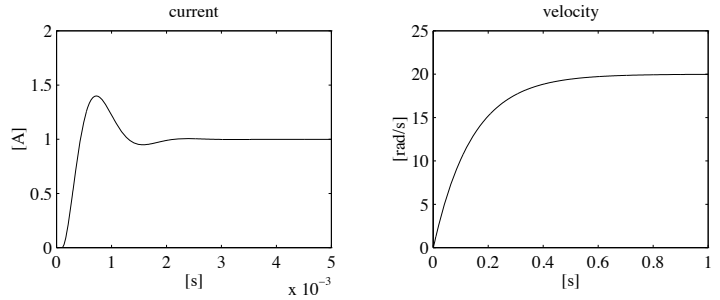

Fig. S5.4. Time history of current and velocity step response for an electric servomotor with current feedback loop and PI control

The resulting SIMULINK block diagram is shown in Fig. S5.3. The servomotor is simulated as a continuous-time system using a variable-step integration method. Since the current dynamics is much faster than the velocity dynamics, the maximum step size has been chosen much smaller than in Problem 5.2, i.e., 0.05 ms. Further, in order to make a comparison between the velocity response in this case and the case of Problem 5.2, the block diagram in Fig. S5.3 features a separate execution with a longer final time and a sampling time of 1 ms.

The files with the solution can be found in Folder `5_3`.

The resulting current and velocity are shown in Fig. S5.4. The current response goes as expected, while the velocity response is slowed down with respect to the response in Fig. S5.2 because it is dominated by the time constant I_m/T_m.

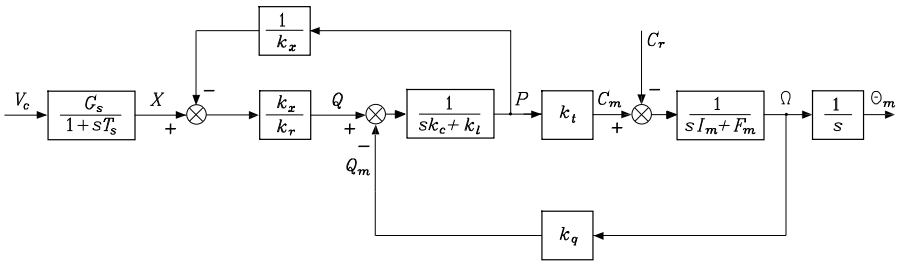

Fig. S5.5. Reduction on the block scheme of a hydraulic motor with servovalve and distributor

Solution to Problem 5.4

From this block scheme, the following relations can be found:

$$\begin{aligned}
\Theta_m &= \frac{1}{s}\Omega_m \\
\Omega_m &= \frac{C_m - C_r}{sI_m + F_m} \\
C_m &= k_t P \\
P &= \frac{1}{sk_c + k_l}\left(\frac{k_x}{k_r}\left(X - \frac{P}{k_x}\right) - k_q\Omega_m\right) \\
&= \frac{k_x X - k_r k_q \Omega_m}{1 + k_r k_l + s k_r k_c} \\
X &= \frac{G_s}{1 + sT_s} V_c.
\end{aligned}$$

Substituting (S5.6) into (S5.6), and then (S5.6) into (S5.6), yields

$$\left(1 + \frac{k_t k_r k_q}{(sI_m + F_m)(1 + k_r k_l + s k_r k_c)}\right)\Omega_m = \frac{k_t k_x}{(1 + k_r k_l + s k_r k_c)(sI_m + F_m)} X - \frac{1}{sI_m + F_m} C_r.$$

Solving for Ω_m, substituting it into (S5.6), and using (S5.6) leads to

$$\Theta_m = \frac{\dfrac{k_x G_s}{k_r k_q}}{s(1 + sT_s)\left(1 + \dfrac{F_m}{k_t k_r k_q}\left(1 + s\dfrac{I_m}{F_m}\right)(1 + k_r k_l + s k_r k_c)\right)} V_c$$

$$-\frac{\dfrac{1+k_rk_l+sk_rk_c}{k_tk_rk_q}}{s\left(1+\dfrac{F_m}{k_tk_rk_q}\left(1+s\dfrac{I_m}{F_m}\right)(1+k_rk_l+sk_rk_c)\right)}C_r.$$

Solution to Problem 5.5

Without loss of generality, the case of 4 bits is considered. The truth table of the interconversion logic circuit is reported in Table S5.1.

Table S5.1. Truth table of the interconversion logic circuit from Gray-code to binary code

	Gray Code	Binary Code
#	$x_1\,x_2\,x_3\,x_4$	$y_1\,y_2\,y_3\,y_4$
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 1	0 0 1 0
3	0 0 1 0	0 0 1 1
4	0 1 1 0	0 1 0 0
5	0 1 1 1	0 1 0 1
6	0 1 0 1	0 1 1 0
7	0 1 0 0	0 1 1 1
8	1 1 0 0	1 0 0 0
9	1 1 0 1	1 0 0 1
10	1 1 1 1	1 0 1 0
11	1 1 1 0	1 0 1 1
12	1 0 1 0	1 1 0 0
13	1 0 1 1	1 1 0 1
14	1 0 0 1	1 1 1 0
15	1 0 0 0	1 1 1 1

The first columns of the two codes are equal, and thus

$$y_1 = x_1.$$

From the second column at the output, it can be recognized that $y_2 = 1$ when $x_1 = 0$ and $x_2 = 1$, or else when $x_1 = 1$ and $x_2 = 0$, and thus

$$y_2 = \bar{x}_1x_2 + x_1\bar{x}_2 = x_1 \oplus x_2 = y_1 \oplus x_2$$

where "$\oplus$" is the symbol for the logical operator XOR. From the third column at the output, it can be recognized that $y_3 = 1$ when $y_2 = 0$ and $x_3 = 1$, or else when $y_2 = 1$ and $x_3 = 0$, and thus

$$y_3 = y_2 \oplus x_3.$$

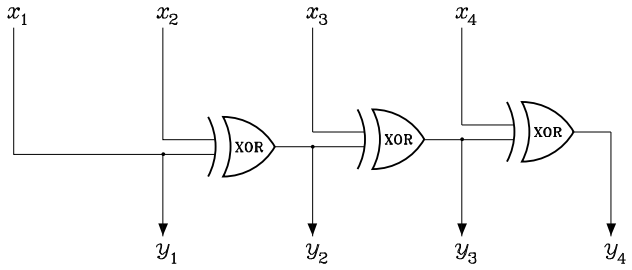

Fig. S5.6. Interconversion logic circuit from Gray code to binary code

Finally, from the fourth column it is

$$y_4 = y_3 \oplus x_4.$$

The resulting interconversion logic circuit is illustrated in Fig. S5.6 where the conventional symbol of XOR is used.

Solution to Problem 5.6

The skew-symmetric operator is

$$\boldsymbol{S}(\boldsymbol{r}_{cs}^{c}) = \begin{bmatrix} 0 & -0.2 & 0 \\ 0.2 & 0 & 0.3 \\ 0 & -0.3 & 0 \end{bmatrix},$$

and thus applying (5.32) gives

$$\begin{bmatrix} \boldsymbol{f}_c^c \\ \boldsymbol{\mu}_c^c \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0 & 1 \\ 0.3 & 0 & 0.2 & 0 & -1 & 0 \\ 0 & 0.3 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 20 \\ 0 \\ 0 \\ 0 \\ 6 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 20 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

that is,

$$\boldsymbol{f}_c^c = [\,0 \quad 0 \quad 20\,]^T \text{ N} \qquad \boldsymbol{\mu}_c^c = [\,0 \quad 0 \quad 0\,]^T \text{ N·m}.$$

As can be seen, there is no resulting moment in the contact frame.

Solution to Problem 5.7

Let the end-effector frame coincide with Frame 4 in Fig. S2.3. Then, with reference to (S2.3), the homogeneous transformation from the base frame to the end-effector frame at $\boldsymbol{q} = [\,0 \quad \pi/4 \quad 0.1 \quad 0\,]$ is

$$\boldsymbol{T}_c^b = \begin{bmatrix} 0.707 & -0.707 & 0 & 0.854 \\ 0.707 & 0.707 & 0 & 0.354 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Using (5.35) gives

$$\tilde{\boldsymbol{p}}^c = \begin{bmatrix} 0.066 \\ 0.141 \\ 0.8 \\ 1 \end{bmatrix}$$

and, in view of (5.41), it is

$$\boldsymbol{\Omega} = \begin{bmatrix} 633.6 & 0 & 250 \\ 0 & 964 & 250 \\ 0 & 0 & 1 \end{bmatrix}$$

and thus from (5.40)

$$\begin{aligned} x_I &= 241.82 \\ y_I &= 335.92 \\ \lambda &= 0.8. \end{aligned}$$

Hence

$$X_I = 302.27 \qquad Y_I = 419.90.$$

6

Control Architecture

Solution to Problem 6.1

With reference to the hierarchical control architecture in Fig. 6.1, the solution can be articulated at the task and action levels as follows.

At the task level, in order to decompose the task into a sequence of actions, the decision module shall consult the knowledge base available in the modelling module concerning the manipulator and the environment. If the environment is scarcely structured, the knowledge base shall be updated with the information to be presented by the sensor module via the use of heteroceptive sensors, e.g., vision. Once the environment is structured, the modelling module provides the relative location between the manipulator, the object and the obstacle. Feasibility of the task shall be verified, and a possible sequence of actions is generated: approach position A, pick object, move object while avoiding obstacle, approach position B, place object. The two approach actions are opportune to ensure smooth pick-and-place of the object.

At the action level, the symbolic commands coming from the task level are interpreted by the decision module which shall take the resolutions needed to execute the given sequence of actions, that is, choice of reference frame, motion in operational space, intermediate path or via points C, D, E, F (see Fig. S6.1), duration of time intervals, path primitives off the obstacle, and type of interpolating functions. By doing so, the decision module shall consult the knowledge base available in the modelling module to verify path feasibility. As for the approach actions, the decisions imparted to the primitive level shall regard low values of velocities and vertical orientation of the end effector, whereas the knowledge base shall be updated with the information available in the sensor module via the use of range or proximity sensors.

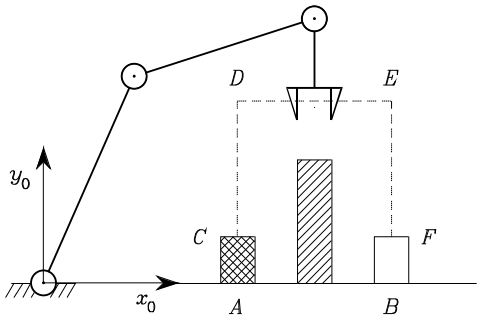

Fig. S6.1. Choice of reference frame and intermediate positions for an object pick-and-place task

Solution to Problem 6.2

With reference to the sequence of points illustrated in Fig. S6.1, the operational space motion primitives can be determined in the two cases of path points and via points, respectively, as follows.

In the case of path points, the various points can be connected by segments. To ensure safe grasp of the object at point C and safe release of the object at point F, it is advisable to choose null velocity and acceleration at each of those points. As for the intermediate points D and E, it is sufficient to impose null velocities. Hence, the timing law along the segments CD and EF shall be polynomials of at least fourth order, whereas the timing law along the segment DE shall be either a cubic polynomial or a trajectory with trapezoidal velocity profile.

In the case of via points, the actual path in the neighbourhood of D and E is of no concern, as long as it is kept at a safety distance from the obstacle. The passage in the proximity of the two via points can be obtained by using the same interpolating functions as for the previous case, on condition that the generation of the timing laws along the segments DE and EF are suitably anticipated in time. As above, null initial and final velocity and acceleration are to be imposed.

Finally, the presence of a redundant DOFs relative to a bidimensional position task can be exploited to keep end-effector orientation constant along $-y$.

Solution to Problem 6.3

Before executing the peg-in-hole task, the end effector shall approach the hole. It can be argued that it is difficult to guarantee exact positioning of the peg relative to the hole; this may render the insertion phase quite critical. It is assumed that the main cause of uncertainty regards only position, that is the axis of the peg can be considered to be aligned with the axis of the hole during

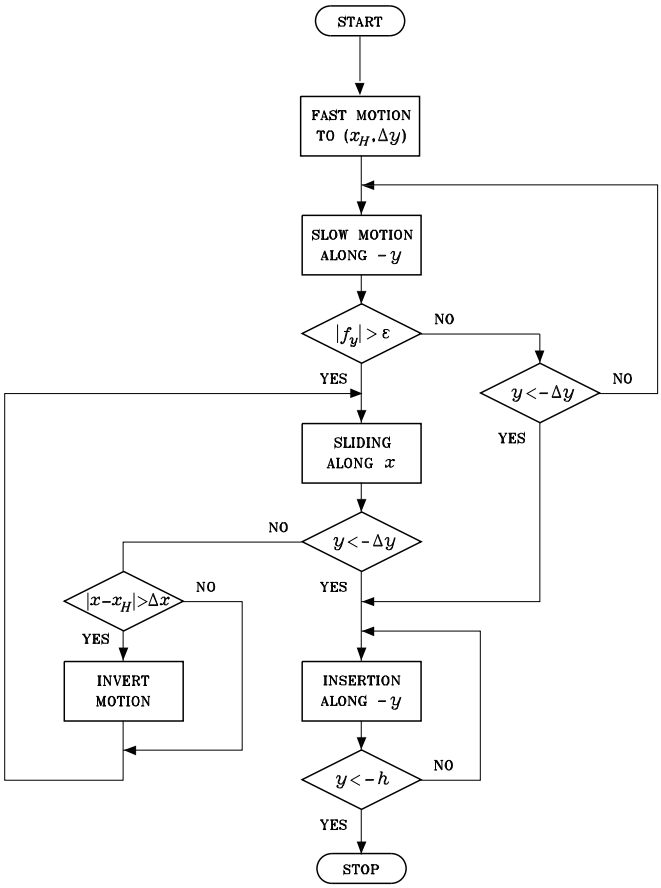

Fig. S6.2. Flow chart for the execution of a peg-in-hole task

the approach to the hole; also, it is assumed that no jamming occurs during insertion.

Choose a reference frame at the base of the arm, with axis x along the horizontal direction (positive from the base to the hole) and axis y along the positive vertical direction. A possible solution is to split the whole task into three subtasks as follows.

At first, a fast motion towards the position $(x_H, \Delta y)$ shall be executed, where x_H is the estimated x coordinate of the center of the hole and Δy is a small distance from the estimated height of the hole ($y = 0$). This is meant to avoid colliding the surface around the hole during the approach phase. A pure motion control strategy is needed to accomplish this subtask.

Next, a slow motion along $-y$ shall be executed, still with a pure motion control strategy. This goes on until y is below the given distance $-\Delta y$

(when the peg is considered to have entered the hole), unless a contact force f_y is sensed above a threshold ϵ, due to a collision with the surface. In that case, before proceeding with insertion, it is necessary to slide along the surface in either positive or negative horizontal direction until a force below the threshold is sensed; if the x coordinate of end-effector position goes too far from x_H (greater than a distance Δx), then the motion shall be inverted. A force/position control strategy is needed to accomplish this subtask where position is controlled along x and force is controlled along y to a desired pushing force. This goes on until the peg enters the hole.

Finally, the insertion along $-y$ is accomplished by using a force/position control strategy where position is controlled along y, force is controlled along x, and moment is controlled about z; moment control is needed to keep the peg in axis with the hole during insertion.

The flow chart corresponding to the articulation of the whole task into the above three subtasks is illustrated in Fig. S6.2.

Solution to Problem 6.4

The task can be divided into the following sequence of actions:

- command new object on conveyor,
- fast motion of end effector to a location above object to pick,
- slow motion of end effector towards object,
- pick object from conveyor,
- fast motion of object to a loading location above pallet,
- slow motion of object downwards,
- release object.

At the beginning, the end effector is moved to a home location from which the task starts. The above sequence is iterated until the pallet is filled, and then the end effector is taken back to the home location.

Below a PASCAL program is listed to execute the given task.

```
program FILL_PALLET;
type  point: array[1..6] of real;
const home: point=(------);      {end-effector home location}
      object: point=(------);    {object location}
      pallet: point=(------);    {lower-left pallet location}
      lift: -.-;                 {z-offset above object}
      length: -.-;               {object x-dimension}
      width: -.-;                {object y-dimension}
      fast: -.-;                 {fast velocity}
      slow: -.-;                 {slow velocity}
var   i,j: integer;
      current: point;            {end-effector current location}
procedure MOVETO(var endpoint:point,velocity:real);
procedure GRASP;
procedure RELEASE;
```

```
procedure NEW_OBJECT;
begin
  MOVETO(home,fast);
  for i:=0 to 3 do
    for j:= 0 to 3 do
      begin
        NEW_OBJECT;
        current:=object;
        current[3]:=current[3]+lift;
        MOVETO(current,fast);
        MOVETO(object,slow);
        GRASP;
        MOVETO(current,slow);
        current:=pallet;
        current[1]:=current[1]+i*length;
        current[2]:=current[2]+j*width;
        current[3]:=current[3]+lift;
        MOVETO(current,fast);
        current[3]:=current[3]-lift;
        MOVETO(current,slow);
        RELEASE;
        current[3]:=current[3]+lift;
        MOVETO(current,slow);
      end
  MOVETO(home,fast);
end
```