Departament d'Enginyeria [DΣIM] Informàtica i Matemàtiques Universitat Rovira i Virgili

ENGINYERIA D'INFORMÀTICA

ROBÒTICA INDUSTRIAL

Curs 2008-2009

PRÀCTICA 3

Joc de nens

PROFESSOR:

ALUMNES:

Índex

Pàgina

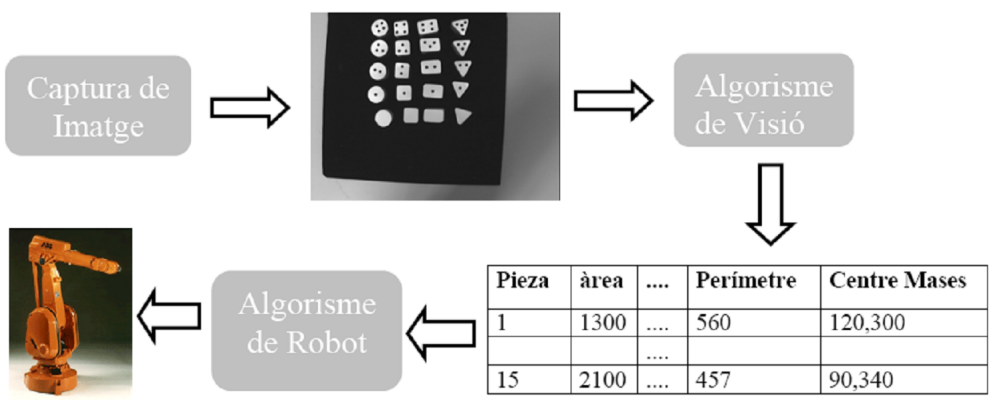
1. Descripció 3
2. Anàlisi 5
3. Disseny 6
4. Implementació 8
5. Joc de proves 12

1. Descripció

Aquesta pràctica tracta de treballar en format Host+UC, de manera que el control del programa es faci des del PC, i no en la U.C. com es habitual treballant en RAPL i el ROBCOMM. La tasca consisteix en treure una de les peces de fusta que estan a la taula i deixar-la a una posició determinada, la que vosaltres vulgueu.

Evidentment, hem de tenir informació de l'entorn de treball per conèixer exactament quina és la situació del taulell i la configuració de les peces. Aquesta informació ens la proporciona un sistema de visió per computador el qual, mitjançant una càmera de vídeo en posició zenital respecte a la zona de treball, obté i analitza una imatge del taulell i les peces, i ens proporciona un fitxer d'informació. Aquesta informació es composa d'una sèrie de característiques dels objectes que es veuen a la imatge (posició, forma, ...). Amb aquest fitxer heu de poder distingir les diferents peces i les seves alçades respectives per tal de construir un "graf de restriccions", en el que tindrem la informació de quines peces estan afectades. Se suposa que a una peça no més pot estar afectada, com a màxim, per les 4 peces que té als seus costats.

La informació de la posició de les peces s'obté del sistema de visió que aporta informació de entorn al sistema robòtic. Aquest sistema està format per una càmera, una placa capturadora de vídeo i un software de tractament i anàlisi d'imatges. Aquest sistema captura una imatge de la càmera de vídeo i la processa de manera que genera un fitxer text amb la informació de les peces del taulell. Aquest fitxer serà una informació d'entrada en el programa que controla el robot.



Pieza	àrea	....	Perímetre	Centre Mases
1	1300	....	560	120,300
		....		
15	2100	....	457	90,340

Com podem veure en el esquema anterior, el procés a seguir es obtindré primer les dades del sistema de visió, utilitzant un algorisme prèviament dissenyat. Com a sortida d'aquest programa tenim el fitxer de dades amb tota una sèrie de característiques de les peces que ens han de servir per classificar-les i situar-les dins l'àrea de treball del robot.
Heu de tenir en compte que tota la informació que tenim en aquest fitxer de dades està definida en "pixels" (una imatge te 512x512 pixels), o sigui que tindrem que fer una conversió de pixels a mil·límetres (unitats de treball del robot).Vegeu l'annex 1.

L'algorisme que heu de dissenyar rebrà informació del fitxer de dades, i tindrà com paràmetres d'entrada el tipus de peça (triangular, quadrada, rectangular, circular) i l'alçada de la peça (0,1,2,3 o 4). Llavors l'algorisme haurà de fer crides a "rutines" fetes en RAPL per tal de treure la peça demanada de la taula. Penseu que probablement haureu de treure algunes peces abans de treure la que es demana.

ON_LINE:

- Lectura del tipus de peça i alçada.
- Lectura del fitxer de peces.
- Càlcul de les posicions de les peces de la taula en coordenades de Robot.
- Càlcul del graf de restriccions.
- Selecció del camí mínim d'extracció.

  Mentre (quedin peces al camí)

  Extreure peça.

  Fmentre
- Deixar peça elegida a posició final

Teniu en compte que els moviments d'extracció de les peces han de ser el més eficients possible. Això significa que els moviments "sense risc" siguin ràpids, mentre que els d'aproximació per agafar o deixar peça, han de ser molt acurats.
També us heu de fixar que les peces estaran ordenades per alçades. Haureu de fer una extracció ordenada (de més altes a mes baixes). Això és degut a que podeu tenir problemes per situar la pinça correctament a l'hora d'extreure una peça.

2. Anàlisi

Per tal de treure la peça s'hauran d'analitzar totes les dependències que té aquesta amb les seves peces veïnes, ja que aquestes poden ser d'una alçada més gran i això impedeix l'extracció d'aquesta peça. Per tant, si volem treure una peça que té una peça veïna amb una alçada major, prèviament haurem de treure la peça veïna abans de la desitjada.

Per treure la peça veïna haurem de fer el mateix procediment realitzat amb la primera peça i així successivament fins trobar una peça sense dependències.

S'ha de tenir en compte que per a l'extracció d'una peça en concret pot haver-hi varies solucions, però d'aquestes solucions s'ha de buscar la que impliqui treure el mínim nombre de peces.

Per tal de trobar la solució amb el mínim camí s'implementarà una estructura d'arbre, que representarà les dependències entre les diferents peces.

3. Disseny

Inicialment fem que es mostrin totes les peces del taulell amb la seva posició i el nombre de taques corresponents per pantalla, perquè l'usuari pugui escollir la posició a treure.

Per llegir les peces del taulell el nom del fitxer és "*captura.seg*", i el fitxer per la matriu de calibració "*matriu.txt*".

S'ha considerat que dues peces de la mateixa mida no es molesten.

El taulell amb les peces pot ser de diferents mides, i hi pot haver peces que no tinguin alguna peça veïna.

S'ha decidit crear un arbre que reflexa les dependències que hi ha entre peces per poder treure la peça desitjada.
Cada node d'aquest arbre emmagatzema la següent informació:

```
struct node{
        int id;
        int pare;          //si pare=id es la arrel
        int orientacio;  //0 horitzontal 1 vertical
        int germa;       //posició a la taula del germa
        int fillsv;        //0 te fills 1 no en te
        int fillsh;       //0 te fills 1 no en te
        int treta;       //-1 si no ha estat treta 1 si s'ha tret
}t[1000];
```

- **id:** Identificador únic de cada peça de la imatge.
- **pare:** Ens indica una dependència. Per a poder treure la peça pare hem de treure abans les peces filles (en cas de que en tingui).
- **orientacio:** Orientació de la peça respecte de la peça pare. Si orientació pren el valor 1, significa que aquesta peça molesta a la peça pare en una orientació vertical. En cas que el valor sigui 0 significa que la molesta en orientació horitzontal.
- **germà:** Indica el node que té una dependència entre el node actual i ell en una orientació determinada per a poder treure la peça pare. Si aquest camp pren

com a valor -1 indica que el node no té germà i per tant la peça pare tan sols depèn del node actual.

- **fillsv:** Indica el nombre de fills que té un node en vertical. En cas que no tingui fills aquest valor valdrà -1.
- **fillsh:** Indica el nombre de fills que té un node en horitzontal. En cas que no tingui fills aquest valor valdrà -1.
- **treta:** Ens indicarà si la peça ja s'ha tret del taulell.

S'ha decidit mostrar els diferents camins que podria escollir el robot per treure la peça i la solució que escull finalment per pantalla perquè l'usuari pugui veure que el camí escollit és el camí mínim. També s'indica la peça a treure mostrant la orientació en la que s'ha de treure, on 0 vol dir orientació horitzontal i 1 orientació vertical.

Per últim, dir que a mesura que es treuen la peça del taulell es van deixant en un cantó de la taula, una al costat de l'altra.

4. Implementació

Identificar les peces veïnes:

Restem la columna i la fila de la peça a treure amb cadascunes de les columnes i files de les peces del taulell, obtenint així la diferència entre les files i columnes que ens permetrà saber si és una peça veïna horitzontalment o verticalment.

Codi per mirar les peces veïnes en horitzontal:

```
comp_objectius_col = peces[t[node_actual].id][0] - peces[i][0];
comp_objectius_fil = peces[t[node_actual].id][1] - peces[i][1];

if ((comp_objectius_col<100) &&
   (comp_objectius_fil<15) && (comp_objectius_col != 0))
```

Codi per mirar les peces veïnes en vertical:

```
comp_objectius_col = peces[t[node_actual].id][0] - peces[i][0];
comp_objectius_fil = peces[t[node_actual].id][1] - peces[i][1];

if ((comp_objectius_col<30) && (comp_objectius_fil<100) && (comp_objectius_fil != 0))
```

Creació de l'arbre:

A partir de la peça que volem treure creem un arbre amb les dependències d'aquesta peça amb les altres peces. La peça que volem treure és l'arrel de l'arbre.

Dues peces amb el mateix pare i la mateixa orientació seran peces germanes, i això significarà que s'hauran de treure les dues per tal de poder treure posteriorment la peça pare.

Per trobar les dependències analitzem la peça que volem treure:

- En cas de que es pugui treure directament serà l'únic node de l'arbre (no tindrà fills).
- En cas de que no es pugui treure directament, les peces veïnes que li molesten seran els seus nodes fills i aquests emmagatzemaran l'orientació amb la que molesten al seu pare.

Per la creació de l'arbre realitzem aquests passos successivament per tots els nodes fills que vagin apareixent.

Analitzar els camins per treure la peça:

El següent codi analitza tots els camins possibles que hi ha a l'arbre per a poder treure la peça arrel i emmagatzema en una taula auxiliar el camí mínim.

1. Mentre no s'hagin analitzat tots els camins busquem un node que no tingui dependències, és a dir, que es pugui treure directament en vertical o horitzontal i l'afegim a una taula de camí auxiliar.
2. Un cop tenim aquest node mirem si té germans, en cas de que en tingui busquem els seus fills fins a trobar un node que es pugui treure directament i l'afegim a la taula auxiliar.
3. Si aquest node no té germans podem pujar al node pare per afegir-lo a la taula de camí auxiliar. Anem pujant així successivament per l'estructura d'arbre fins arribar a l'arrel (peça que vol treure l'usuari).
4. Cada cop que s'afegeix un node a la llista de camí auxiliar s'incrementa un comptador auxiliar. Aquest comptador cada cop que s'analitza un camí nou és inicialitzat a 0. En cas de que el comptador auxiliar sigui més petit que el comptador del camí mínim, el camí mínim passa a ser el camí que s'acaba d'analitzar i el comptador del camí mínim passa a ser igual al comptador auxiliar. D'aquesta manera de tots els camins ens quedem amb el camí que passa per menys nodes.

Codi per buscar el camí mínim:

```
while (fi==0){
    i=k;
    saltsaux=0;
    if (t[k].fillsv==0 || t[k].fillsh==0){
        while (t[i].id!=peca){
            saltsaux++;
            t[i].treta=1;
            printf ("TREC: %i\n",t[i].id);
            toaux[saltsaux-1]=i;
            if (t[i].germa==-1){
              i=t[i].pare;
```

```
}else{
    j=t[i].germa;
    if (t[j].treta==1) {
          i=t[i].pare;
    }else{
         if (t[j].fillsv==0 || t[j].fillsh==0 ){
           i=t[i].germa;
         }else{
         //cas en que el germà té fills
         //si el germà té fills busquem a la dreta nodes en que el pare sigui el germà
             j=i;
            trobat=0;
               while (j<node_actual && trobat==0){

                    if (t[j].pare==t[i].germa && t[j].treta==-1){
                                if (t[j].fillsv==0 || t[j].fillsh==0){
                                 i=j;
                                 trobat=1;
                                }else{
                                  //si té fills busquem els fills dels fills cap a la dreta
                                  for (z=j;z<node_actual;z++){
                                     if (t[z].pare==j && t[z].treta==-1){
                                       if (t[z].fillsv==0 || t[z].fillsh==0){
                                          i=z;
                                       }else{
                                            i=z;
                                       }
                                     }else{
                                          i=z;
                                     }
                                  }
                                }
                    }else{
                       if (j==node_actual){
                          i=j;
                       }
                    }
                    j++;
               }
         }
```

```
                }
            }
        }
        //mirem si el número de salts és inferior al que teniem
        if (saltsaux<salts){
            salts=saltsaux;
            node_minim=k;
            for (z=0;z<salts;z++){
                to[z]=toaux[z];
            }
        }

    }
    printf ("Número de salts %i\n",saltsaux);
    if (k==0){
            fi=1;
        }else{
            k=k-1;
            for (j=0;j<node_actual;j++){
                t[j].treta=-1;
            }
        }
}
```

5. Joc de proves

Joc de proves 1:

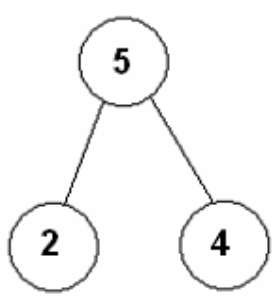
Taulell:

- **Número de peça:** Taques

0: 4	**2:** 4	**1:** 1	**3:** 3
4: 3	**5:** 2	**6:** 1	**7:** 1
8: 4	**9:** 1	**10:** 1	**11:** 0
12: 0	**13:** 0	**14:** 3	**15:** 2

Volem treure la peça número 5. En aquest cas ens quedarà un arbre de la següent manera:

Peça a treure en vertical



Peça a treure en horitzontal

Veurem que tant horitzontalment com verticalment hem de treure una peça per poder treure la peça desitjada.

- Si treiem la peça horitzontalment haurem de treure primer la peça 4 en orientació horitzontal.
- Si treiem la peça verticalment haurem de treure primer la peça 2 ja sigui en orientació vertical com el horitzontal.

Camins analitzats:

2-5

4-5

Camí escollit:

```
SOLUCIO:
PEÃA: 2
qualsevol orientacio serveix: 0
PEÃA: 5
orientacio: 1
```

2-5

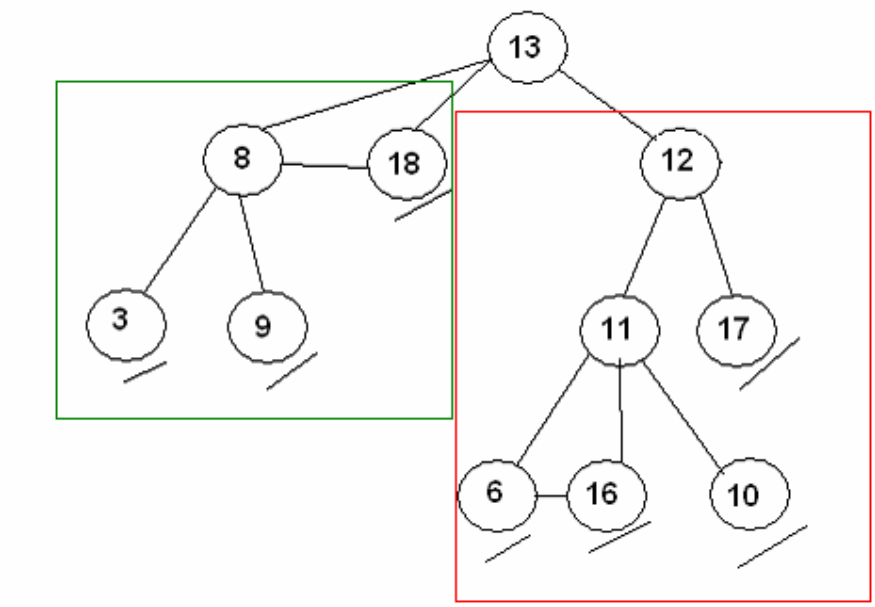
Joc de proves 2:

Taulell:

- **Número de peça:** Taques

0: 4	**1:** 4	**2:** 2	**3:** 3	**4:** 4
5: 3	**6:** 4	**7:** 1	**8:** 2	**9:** 4
10: 4	**11:** 3	**12:** 2	**13:** 0	**14:** 0
15: 0	**16:** 4	**17:** 3	**18:** 2	**19:** 4

Volem treure la peça número 13. En aquest cas ens quedarà un arbre de la següent manera:



El requadre en vermell ens indica les peces veïnes en horitzontal de la peça a treure.
El requadre en verd ens indica les peces veïnes en vertical de la peça a treure.
Per exemple, veiem que la peça 17 no té cap node fill, ja que en vertical es pot treure directament.

Camins analitzats:

17-12-13

10-11-12-13

6-16-11-12-13

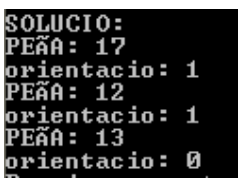
3-8-18-13

9-8-18-13

18-3-8-13

18-9-8-13

Camí escollit:



17-12-13

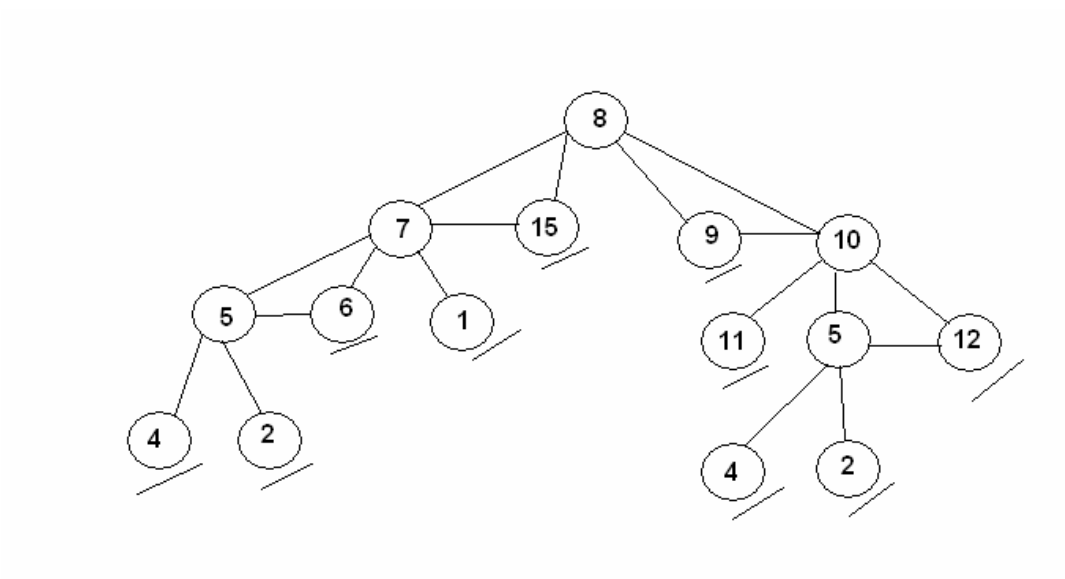
Joc de proves 3:

Taulell:

- **Número de peça:** Taques

0: 4	**2:** 3	**1:** 2	**3:** 0
4: 3	**5:** 2	**7:** 1	**6:** 2
11: 2	**10:** 1	**8:** 0	**9:** 1
14: 3	**12:** 2	**15:** 1	**13:** 2

Volem treure la peça número 8. D'aquesta manera ens quedarà el següent arbre.



Camins analitzats:

2-5-6-7-15-8

4-5-6-7-15-8

2-5-12-10-9-8

4-5-12-10-9-8

1-7-15-8

6-4-5-7-15-8

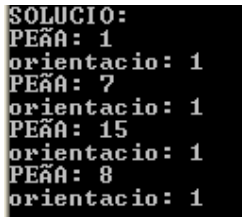
12-4-5-10-9-8

11-10-9-8

15-2-5-6-7-8

9-11-10-8

Camí escollit:



1-7-15-8

Joc de proves 4:

Taulell:

- **Número de peça:** Taques

0: 4	**1:** 0	**2:** 0	**3:** 4	**4:** 4
5: 4	**6:** 1	**7:** 2	**8:** 3	**9:** 4
10: 4	**11:** 1	**12:** 0	**13:** 4	**14:** 0
15: 4	**16:** 4	**17:** 4	**18:** 4	**19:** 4

Volem treure la peça número 6.

Camins analitzats:

En aquest cas com la peça es pot treure directament en orientació vertical, el node no té cap node fill.

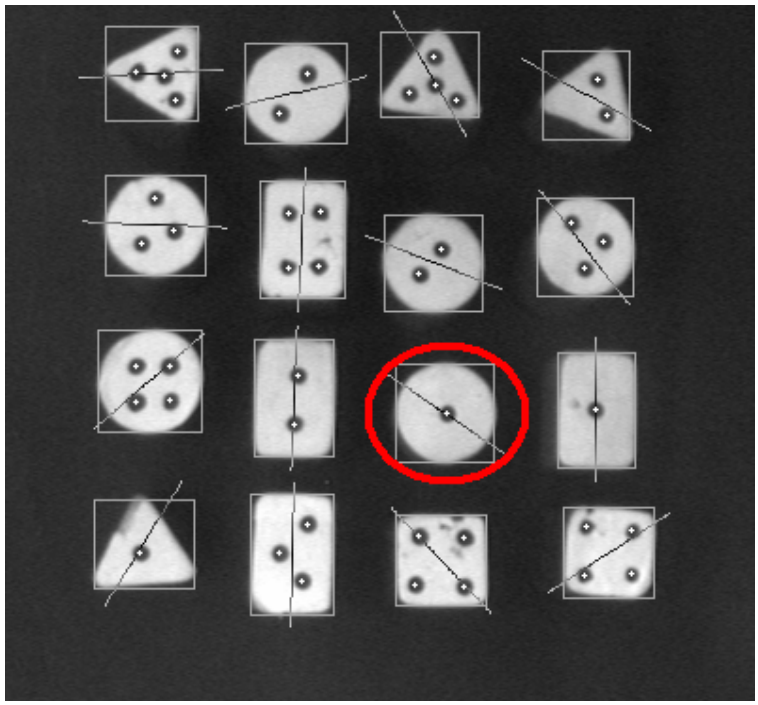
Camí escollit:

```
SOLUCIO:
PEÃA: 6
orientacio: 1
```

6

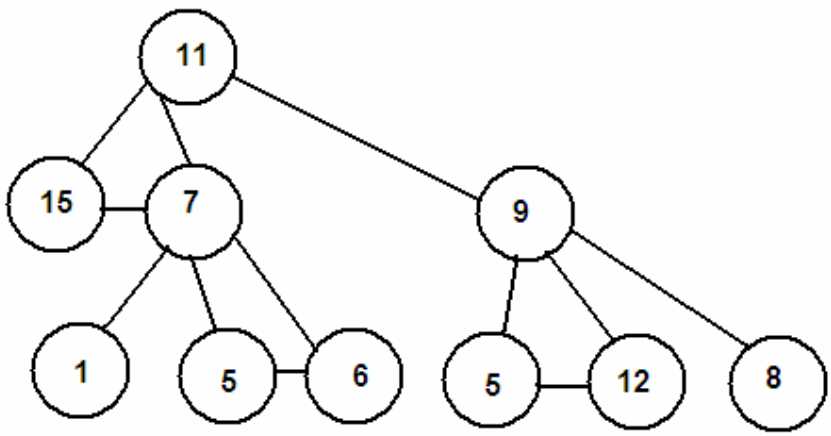
Joc de proves 5:

Taulell:



0: 4	**2:** 2	**1:** 4	**3:** 2
4: 3	**5:** 4	**7:** 2	**6:** **3**
8: 4	**9:** 2	**11:** 1	**10:** 1
13: 1	**12:** 3	**15:** 4	**14:** 4

Volem treure la peça número 11. D'aquesta manera ens quedarà el següent arbre:



Camins analitzats:

12-5-9-11

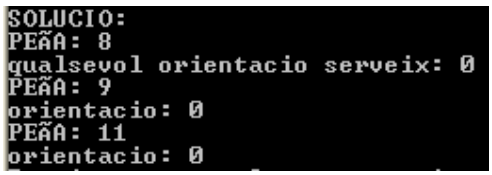
5-12-9-11

8-9-11

1-7-15-11

6-5-7-15-11

Camí escollit:



8-9-11