Search, Actuate and Navigate Labreport Lastweek 2003-2004

Aziz Baibabaev & Marc Bron
cknr: 0222704 & 0130486

June 25, 2004

1 The problems

The assignment for the last week: Using the camera of the Aibo robot recognise a single move on a chess board and calculate the boardcoordinates from a snapshot of the board.

1.1 How to calibrate the camera

In the head of the Aibo is a camera. The Aibo is placed on top of a personal computer, the camera facing the chessboard. To do any calculations it is necessary to know exactly where the camera is, in what angle it is positioned and that the board is aligned perfectly straight in front of it.

1.2 Getting the pictures

We have to call the Aibo and tell it to take a snapshot of the board. To do this we have to call the TekkotsuMon java programs that control the Aibo. The camera takes two snapshots; one before the move and one after the move. These pictures have to be compared to find out where some pixels have disappeared and where some have appeared.

1.3 How to calculate the board coordinates

When we have the pictures and we know where the board is, we have to calculate the board coordinates. In the picture we have to find the camera coordinates of the piece that has been moved. Then we need to create a projection matrix. This matrix translates the camera coordinates into world coordinates.

2 The solutions

We were given a lot of code[1] that created a picture that showed where a piece had been moved. So now we only had to find a way to calibrate the camera and how to calculate the world coordinates.

We have assumed a couple of things:

- The Aibo is not moving.
- The chessboard is not moving.
- There are no shadows.
- Every part of the board is visible.[2]

[1] By E.H. Steffens
[2] Even when there are pieces on it

2.1 Calibrating the camera

First of all we shall put four stickers of different coulours on the corners of the board. We shall name them from left to right, top to bottom: A,B,C,D (from the perspective of AIBO). We wrote a little program that calls the Aibo and reads a picture and calculates the coordinates of the four diffrent corners. In the picture the board looks like a trapezium. When the board is exactly in front of the camera the Y coordinates of the top corners are equal, the Y coordinates of the bottom corners are equal and the absolute distance between the X coordinates dX(A,C) and dX(B,D) is equal.

The pseudocode for calibrating the board:

1. We appoint a precision margin[3]
2. A loop checks if the board is straight enough.
3. If By > Ay rotate clockwise.
4. If By < Ay rotate counter clockwise.
5. If Bx > Dx translate left upon x-axis.
6. If Ax < Cx translate right upon x-axis.

At first we wanted the Aibo to move it's head untill the camera was calibrated correctly. This however was to much work for an issue that had nothing to do with the real task. So the program prompts the user to move the board. First you have to rotate it and then you have to translate it and this process repeats itself untill the board is in front of the camera.

2.2 Calculating the board coordinates

To project the 2d camera image on the 3d world coordinates we need:

- To rotate the 2d image on the x-axis negatively over the

$$\theta = \arctan \frac{aiboheadheight}{aiboheadtocentreofboarddistance}$$

- Than translate the 2d image over the camera viewing direction

$$Tz = -aiboheadheightTy = -aiboheadtocentreofboarddistance$$

When we combine the translation and the rotation we get the projection matrix.

[3]Something like 5-10 pixels

$$\textbf{projectionmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & -Ycameradist \\ 0 & \sin\theta & \cos\theta & -Zcameradist \\ 0 & 0 & 1/f & 1 \end{pmatrix}$$

To find the camera coordinates a program searches the picture array for the specific colour of the blop (the piece that moved). The resulting x,y coordinates at the base of the blop are multiplied by the projection matrix. This results in the world coordinates.

3 Conclusion

All in all we didn't have enough documentation for the available code and eventhough we roughly knew where we had to go, we couldn't test our ideas (and correct them) because of our limited knowledge of used platforms and the needed interfaces to make it work with our code. We had a lot of fun though working on this project in the last week. We got to work with the Aibo[4] and we learned to work with some new programs (matlab, tekkotsu, latex). Maybe the project was too ambitious for one week, but we get another chance next year in the computer vision course.

[4] Personally I think they are evil and they will try to take over the world someday

4 appendix

A projection algorithm code

Partial implementation of the projection algorithm:

```
//Has? the ability to project a
//2d camera coordinate upon board.

public class CameraPoint?
{
 double xcam,
     ycam,
     //set distance from zero point of the camera manually
     ycampos = 0,
     //must remain zero since the camera is looking
     // directly through the middle of the board
     xcampos = 0,

     //set height of the camera manually
     zcampos = 0,
     //y coordinate of aibocam - y coordinate of
     // the middle of the board
     ycamdist,
     //height of aibocam - height of board
     zcamdist,
     xchess,
     ychess;

 String chesspos = "";
 //the angle between the camera viewing direction
 // and the normal of the board
 double camangle,
 //aibocam focuspoint.
       f = 0;

 ChessBoard? b;

 public CameraPoint?(int xcam, int ycam, ChessBoard? board)
 {
   this.xcam = xcam;
   this.ycam = ycam;
   b = board;
```

```
    ycamdist =  ycampos - (b.coords.y+b.sur_y+b.delta_y*4);
    zcamdist = zcampos - b.coords.z;
    camangle = Math.atan(ycamdist/zcamdist);
  }

  public void projectToChess?()
  {
    //first we rotate around x-axis of the
    //board in the positive direction
    //than we translate to the chessboard
    double[][] projection =
 {{1,            0,                      0,                0},
 {0, Math.cos(camangle), - Math.sin(camangle), - ycamdist},
 {0, Math.sin(camangle),   Math.cos(camangle), - zcamdist},
 {0,            0,                      0,                1}};

     xchess = xcam;
     ychess = ycam*projection[1][1] + projection[1][3];
//amount of squares from h to a on the board
     char column =
     (char)('h' - (b.coords.x - xchess - b.sur_x)
     /(double)b.delta_x);

     char row =
     (char)('8' -(ychess - b.coords.y - b.sur_y)
     /(double)b.delta_y);
     chesspos = chesspos + column + row;
  }
 }
```

B calibration algorithm code

partial implementation of the calibration algorithm:

```
 // calibration.java probeerd het midden van
// het bord recht voor de aibo te leggen.
 // author:  marc bron & aziz baibabaev
 // cknr: 0130486 & 0222704

 import tio.*;

 public class Calibration
 {
```

```
private point A;
private point B;
private point C;
private point D;

public void Calibration(point A, point B, point C, point D)
{

 A = this.A;
 B = this.B;
 C = this.C;
 D = this.D;

}
public void doCalibration()
{
  While ( (A.y != B.y || C.y != D.y) ||
( Math.abs(B.x - D.x) != Math.abs(A.x - C.x) ) )
  {
   System.out.println(" rotate ")

   if(B.y > A.y)
     System.out.println(" rotate clockwise ");
   else if (B.y < A.y)
     System.out.println(" rotate counter clockwise ");

   System.out.println(" translate ");

   if(B.x > D.x)
     System.out.println(" translate left upon x-axis ");
   else if (A.x < C.x)
     System.out.println(" translate right upon x-axis ");

  }
}

}
```

C main method calibrate

`Calibrate.java implementation`

```
//naam:  marc bron & aziz baibabaev
//cknr:        0130486   & 0222704
// de main methode die calibration aanroept

// Note: tio.jar is an io package available
// at: ftp://ftp.cse.ucsc.edu/pub/charlie/jbd/tio-v2/tio.jar

import tio.*;
import org.tekkotsu.mon.*;
import java.awt.*;

public class Calibrate
{
 public static void main (String [] args)
 {
  System.out.println(" This program tries to calibrate " +
  "the camera.\n Press any key to begin. ");
  Console.in.readChar();

  int x;
  VisionListener image = new VisionListener();
  System.out.println(" To stop press 0, to go on press 1. ");
  x = Console.in.readInt();

  if(x == 1)
  {

    System.out.println(image.getImage());
  }

  // lees een plaatje en vindt de hoekpunten
  while(x != 0)
  {
  System.out.println(" To stop press 0, to go on press 1. ");
   x = Console.in.readInt();

   image = VisionListener.getImage();

   for(int i = 0; i < image.length; i++)
   {
    //  image is een BufferedImage? dat
    // (hebben we gehoord) zou een
    //  array zijn van pixels
    //  met een for loop kunnen we die scannen en zoeken naar
```

```
    //  de kleuren van de stickers
    //  op de hoekpunten.

   }

   //  als de hoekpunten A,B,C,D zijn
   //gevonden wordt Calibration(A,B,C,D)
   // aangeroepen
   //  die vertelt ons dan om het bord te verplaatsen,
   // todat het bord goed ligt.
  }

 else
  System.out.println("goodbye");

 }
}
```

References

[1] D.H. Ballard and C. M. Brown. *Computer Vision.* Prentice Hall, first edition, 1982.

[2] Leo Dorst. *An Introduction to Robotics for the Computer Sciences.* University of Amsterdam, first edition, 2001.

[3] S. G. Hogger. *Mathematics for Computer Graphics.* Cambridge, first edition, 1992.

[4] Y. Daniel Liang. *Introduction to Java programming.* Prentice Hall, third edition, 2001.

[5] G. Lindfield and J. Penny. *Numerical Methods Using Matlab.* Ellis Horwood Limited, first edition, 1995.