BIOLOGICALLY INSPIRED
ROBOTICS GROUP (BIRG)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

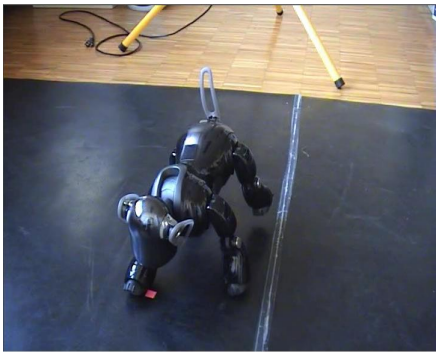
Visual location of a mark by the robot Aibo

Semester Project

Jennifer Meinen

Supervision: Prof. Auke Jan Ijspeert & Sarah Degallier
Biologically Inspired Robotics Group (BIRG)
School of Computer and Communication Science - EPFL

January 12, 2008



Contents

List of Figures

List of Tables

1 Introduction

The goal of this project is to locate a mark using a camera, and then use this information to define a special action. More precisely I will work on Aibo which is a robotic pet produced by Sony, and use its camera to identify and locate a mark. Then if Aibo sees a mark it must move one of its paws onto the mark.

The solution developed must be tested and validated in simulation. Finally it also needs to be implemented on the real robot Aibo.

The project issues are to develop a general method that a robot can use to analyse an image and to use this image to adopt a particular behaviour. For example a robot could use the vision part for its locomotion, or to avoid obstacles. Another example could be a robot on an assembly line which needs advance information from the camera about the position of objects to be able to manipulate them.

2 Solving the problem

Aibo moves its head until it recognizes the mark and then it will stop moving. To make the mark easier to detect, we use a color different from its environment - for exemple red. When it finds a group of pixels with a higher threshold of red we know that there is a strong probability that it is in fact the mark.

The most difficult part will be to find the 3D coordinates corresponding to the mark, as the head and the camera will be moving. Section 4 will detail each step and the difficulties encountered to perform this task. Aibo must be able to calculate the coordinates of the mark whatever the position of its head. We therefore have to do some referential transformations and modelize the concept of the camera. The robot's legs can be in any position but have to remain immobile during the calculation of the coordinates of the mark. The reason for this is that the plane in which the mark is situated is determined by the position of the legs.

Finally, Aibo will move its paw onto the mark using inverse kinematics.

Although Aibo has a distance sensor, we wanted to be able to use the solution on other robots that do not have such sensors. Therefore it was decided to use only the camera to locate the mark. However, section 4.6 will present how the information from the distance sensor could be used to solve our problem.

3 Looking for the red mark

First of all we must define what constitutes a red mark: it is determined by values within the limits of red, green and blue. Figure 1 shows the 3D representation of the rgb color system with a red cube limited by r1,r2,g1,g2,b1,b2. So we can say that a pixel whose rgb values are r,g,b is red if the conditions : r1<r<r2, g1<g<g2, b1<b<b2 are satisfied.

To locate the mark Aibo turns its head and takes a picture. It then scans the image : for each position x,y of a pixel it knows the rgb values r,g,b and compares these values with the predetermined values r1,r2,g1,g2,b1,b2. If a pixel is recognized as being red we know its position. We need a group of pixels to enable the recognition of the mark.

After recognizing a group of pixels as red it will determine the middle of this group (see section 3.1) which will be a point in 2D in the plane of the formation of the image inside the camera. Section 4 will describe how this point can be mapped to a 3D point. If Aibo cannot find a group of pixels which fill the conditions it will again turn its head and scan.

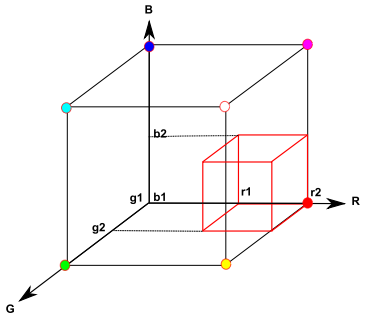


Figure 1: RBG cube

3.1 Middle of the mark

We want to reduce the mark to a single point at which Aibo will aim its paw. This point is logically in the middle of the mark. As the mark can be turned in any position we need to develop an algorithm to calculate this point. Figure 2 illustrates the mark turned obliquely and to determine the middle we must do the following steps :

- Count the number of pixels of each line.
- Determine the zone in which there is the largest number of pixels.
- Take the two lines that limit the zone : we now have A,B,C,D as on Figure 2.
- Determine the middle of the parallelogram formed by those two lines.

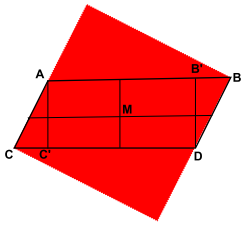


Figure 2: Mark turned

More precisely the solution used counts the number of pixels of the mark as it goes through the image and so the points A, B, C, D can be determined : A and B are the first points where the number of pixels are maximum, along the line between A and C the maximum number will not change. C and D are the last points before the number of pixels decrement.

4 Mapping from 2D point in an image to 3D coordinates

To tackle the problem we have to convert a pixel of an image to a 3D coordinate in the frame reference of Aibo's body. First we have to limit the problem : we suppose that Aibo and the mark lay on the same plane. Let us decompose this problem :

- ◇ We must fix the referential on Aibo's body and we must know how to convert any point of the body referential frame into a coordinate in the camera referential frame and inversely.
- ◇ We must also express the floor in the referential frame of the body, and every time the body moves the frame has to be updated.
- ◇ We have to determine the angle of view and the focal of the camera (horizontal and vertical angle view).
- ◇ We must know how to convert a pixel into a 2D coordinate.

Knowing the referential on Aibo's body the floor (orientation and distance) can be determined in the camera referential frame. As the camera angle of view and focal length are known, the problem is reduced to a geometric transformation. Finally, we convert the coordinates into the referential frame of the body.

4.1 Coordinates in each referential frame

We are going to fix the referential frame of Aibo's body on the Tilt1 Center (Fig. 3), we will use the following notation for this referential frame : Body Ref. frame.

The referential frame of the camera is in the center of the camera (Fig. 4). The distances are important to define each translation matrix. Given a point P(P_x , P_y ,P_z) expressed in the Body Ref. frame which we want to express in the camera referential frame, we have to know the degrees of freedom of the head and all the distances between each axe. Then we can calculate a matrix which is the multiplication of all transformations necessary for a change of referential frame. In any geometry book we can find the well known matrices which represent translation and rotation of a referential :

$$Trans(x,y,z) = \begin{pmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad Rot(x,\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & sin(\theta) & 0 \\ 0 & -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Rot(y,\theta) = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad Rot(z,\theta) = \begin{pmatrix} cos(\theta) & sin(\theta) & 0 & 0 \\ -sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note that if θ is between 0 and π it will be a counter-clockwise rotation in a usual x,y,z referential.

4.1.1 Degrees of freedom

According to the Model Information for ERS7[2], the head can do the following movements :

- ◇ rotation around the Tilt1 Center
- ◇ rotation around the Tilt2 Center
- ◇ rotation of the head around the neck axis

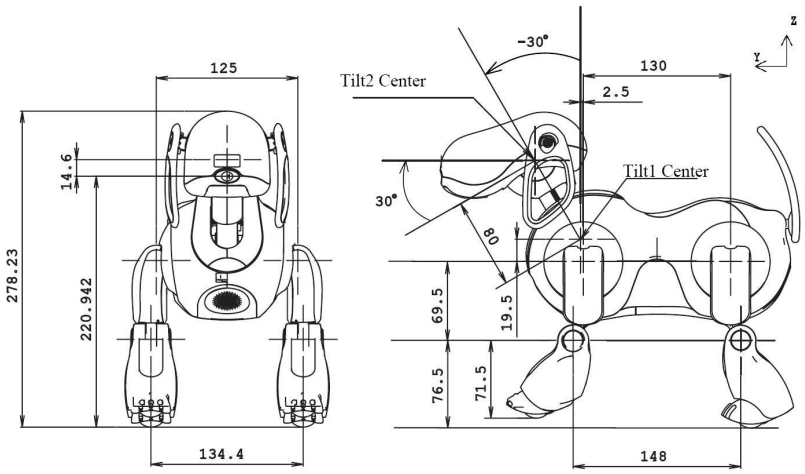


Figure 3: Aibo's body dimensions figure taken from Model Information for Ers7 [2]

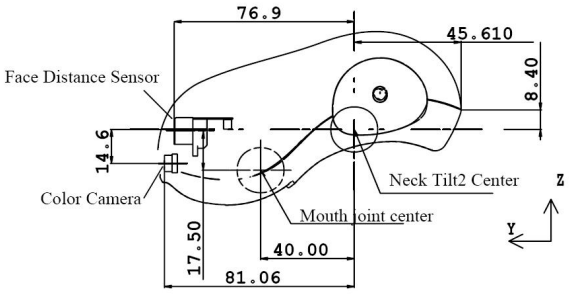


Figure 4: Aibo's head dimensions figure taken from Model Information for Ers7 [2]

4.1.2 Transformation matrices

We can now express the point P in the camera referential frame by doing the following transformations (see Fig. 3 and 4 for distances and to see how the axis x,y,z are oriented). Table 1 summarizes the position of the current referential frame, its orientation and the following geometric transformation. So the point C(c_x,c_y,c_z), which is P expressed in the camera referential frame, will be :

$$\begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix} = M * \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

$$M = Trans(a_1, a_2, a_3)Rot(x, \gamma)Trans(b_1, b_2, b_3)Rot(z, \beta)Rot(x, \alpha)$$

If we had C(c_x,c_y,c_z) and we were looking for P(p_x,p_y,p_z) we could do:

$$\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = M^{-1} * \begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix}$$

From the Model Information for ERS7[2] we know the distances in mm : (a_1,a_2,a_3)=(0,81.06,-14.6) and (b_1,b_2,b_3)=(0,0,80).

Referential frame x,y,z	Transformation on x,y,z
The current referential frame x,y,z is oriented as on figure 3 and is centered on Tilt1.	A rotation Rot(x,α) with α being in the plane yz and representing the angle between z and the neck. Negative angle for head going down.
The current referential frame x,y,z is still centered on Tilt1 but is oriented with z being along the neck axis.	A rotation Rot(z, β) with β being in the plane xy and representing the angle y and the new position of the head once turned. Positive angle for head turning to the right.
The current referential frame x,y,z is still centered on Tilt1 but is oriented with z being along the neck axis and y being parallel to the head.	A translation along the z axis of b_3 mm : Trans(b_1,b_2,b_3)
The current referential frame x,y,z is now centered on Tilt2.	A rotation Rot(x,γ) with γ being in the plane yz and representing the angle between y and the neck. Positive angle for head going up.
The current referential frame x,y,z is still centered on Tilt2 but y is along the head.	A translation Trans(a_1,a_2,a_3) to bring the current referential frame into the referential frame of the camera

Table 1: Transformation of the referential frame

4.2 Distance and orientation of the ground viewed from the body referential

A plane is defined by only three different points. So the idea is to express three points on the floor in the Body Ref. Frame and then as we know how to convert these points into the camera referential frame we will be able to know the distance and the orientation of the floor in the camera referential frame. We assume that after having set Aibo in one specific position and calculated the three points needed, its body referential frame will not change again - it means that after having calculated the points P_0, P_1, P_2 which represent the floor the articulations of the legs should not move anymore otherwise the result of the three points will no longer be valid. In any case the head could still move because the referential is on the body.

4.2.1 Initial position

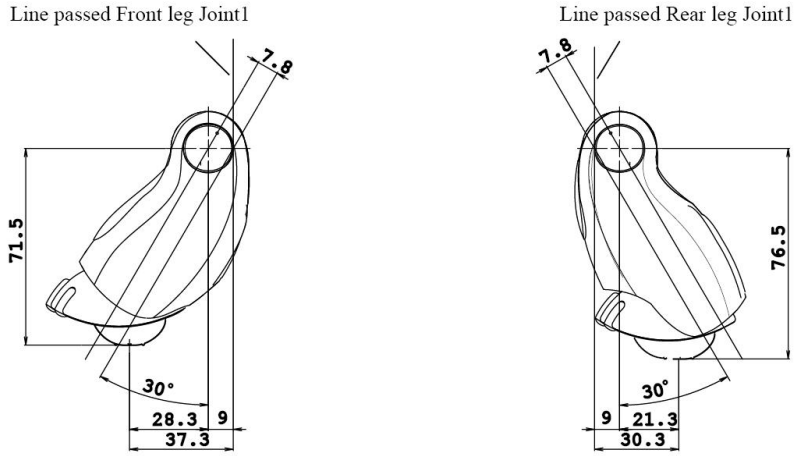


Figure 5: Aibo's legs dimensions [2]

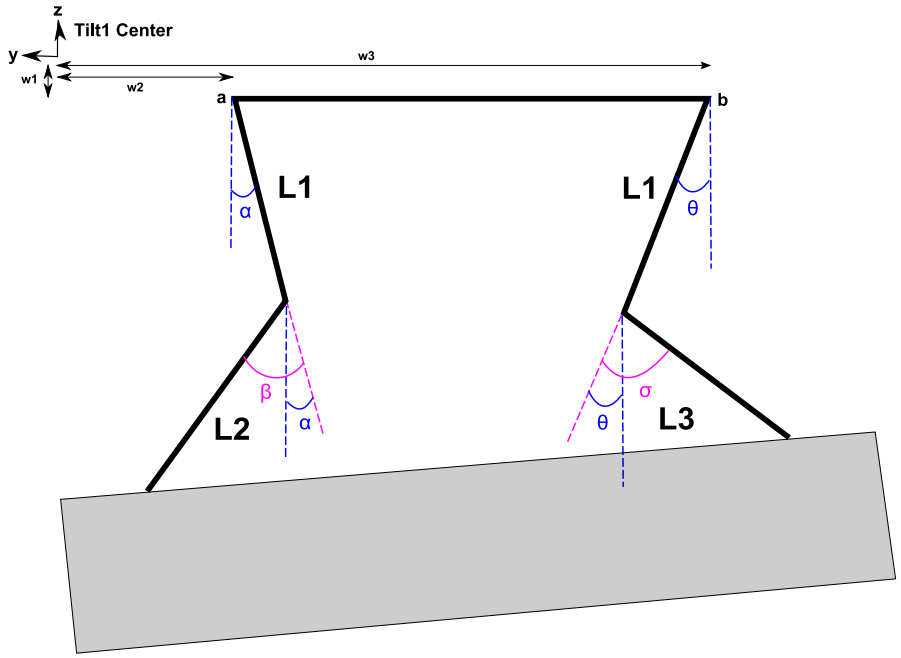


Figure 6: Aibo's body position

The initial position is chosen so that the two front paws are in the same positon as well as the two back paws. We want to determine the coordinates of the floor in the body referential frame placed on the Tilt1 center (cf Fig. 5). See Fig. 6 p. 7 and Table 2 p. 8 for all the variables and explanations.

Given that w_4 is the distance between the two front paws and is also the distance between the two back paws, we can now express the points P_0 ,P_1 , P_2 in the body ref. frame:

$$P_0 = (w_4/2, -w_2 - sin(\alpha) \cdot L1 - sin(\alpha + \beta) \cdot L2, -w_1 - cos(\alpha) \cdot L1 - cos(\alpha + \beta) \cdot L2)$$

$$P_1 = (-w_4/2, -w_2 - sin(\alpha) \cdot L1 - sin(\alpha + \beta) \cdot L2, -w_1 - cos(\alpha) \cdot L1 - cos(\alpha + \beta) \cdot L2)$$

$$P_2 = (w_4/2, -w_3 + sin(\theta) \cdot L1 - sin(\alpha + \sigma) \cdot L3, -w_1 - cos(\theta) \cdot L1 - cos(\theta + \sigma) \cdot L3)$$

We can now compute the equation of the plane using these points. The parametric equation is $P_0 + (P_1 - P_0)u + (P_2 - P_0)v$ as in section 4.4.2.

4.2.2 Recomputing body position

Each time Aibo moves its legs we update the values of α, β, θ, σ which are the only values needed to calculate P_0, P_1, P_2. After having calculated these three points Aibo must not move anymore otherwise it should recalculate the points. The calculation of the position of the mark can be done in any body position. However the only condition is that the angles of the articulations of both front legs must be identical - similarly for the hind legs. However, to avoid Aibo collapsing or being in a position which does not allow it to move a paw : we are going to fix the angles α, β, θ, σ in advance. When it sees the mark, it will move its paws in the desired position and then it will be ready to calculate P_0, P_1, P_2 and determine the position of the mark, and finally move its paw onto the mark (during all the operation it must not move its body).

Variables and constants in Fig. 6	Signification and initial value
α	Placed between the perpendicular of the line ab and L1. Also includes the angle between the upper arm and the lower joint as they are out of axis (cf Fig 5). This initial angle can be calculated and measures $-arctan(9/19.5)$.
L1	Length between the two axis of rotation for the front paws - which is the same for the back paws. It measures $\sqrt{19.5^2 + 9^2}$.
L2	Distance between the L2 rotation axis and the floor contact point. It measures $\sqrt{28.3^2 + 71.5^2}$.
L3	Distance between the L2 rotation axis and the floor contact point. It measures $\sqrt{21.3^2 + 76.5^2}$.
β	Angle between L1 and L2.
θ	Similar calculation to α but for back paws.
σ	Angle between L1 and L3.
w_1	Distance along z between Tilt1 and line ab.
w_2	Distance along y between Tilt1 and line a.
w_3	Distance along y between Tilt1 and line b.

Table 2: Initial position

4.3 Mapping from a pixel to a 2d coordinate

Initially we have an image with n x m pixels. Suppose we know the focal length F, the horizontal (α) and vertical angle of view, we want to know how to convert a particular pixel (u,k) in a 2D coordinate that we can use later for the projection (see Section 4.4.1 on the pinhole camera model). In Fig 7 the image plane is represented : h is half the number of pixels on the horizontal line, u is the number of pixels from the middle of the image to the point (u,k) on the vertical line. The problem is: how to convert a number of pixels into a physical distance? We know that h pixels correspond to $d_h = F \cdot tan(\alpha)$ and so we can deduce that u pixels correspond to $d_u = u \cdot \frac{(F \cdot tan(\alpha))}{(h)}$. Same reasoning can be done for k but using the vertical angle of view.

4.4 Projection on a plane

We know the focal length, a 2D coordinate on the image plane and the equation of the floor plane so we can therefore calculate the 3D coordinate.

4.4.1 The pinhole camera model

We will use the pinhole camera model which is a simple way of representing a camera as a small hole through which light enters to form an image. The distance between the image formed and the camera referential frame is the focal length. In our case we know that any point on the image plane is on a particular plane (the floor plane of which we know the equation in the camera referential frame). Fig. 8 illustrates a coordinate system which is placed just onto the camera - y being along the optical axis, the floor plane and the image plane. P' is the projection of P on the image plane. The problem is to find the coordinates of P knowing the coordinates of P'. We must do as the camera does : a projection on a plane but in our case the plane is the floor plane of which we know the equation. The 3D coordinates of P' are known (focal length on y see Section 4.3 for x and z), the equation of the floor plane is also known from the 3 points fixed in Section 4.2.1 and converted in the camera frame reference. Let us call D the line going through (0,0,0) and P'. The intersection between this line and the floor plane completely determines P.

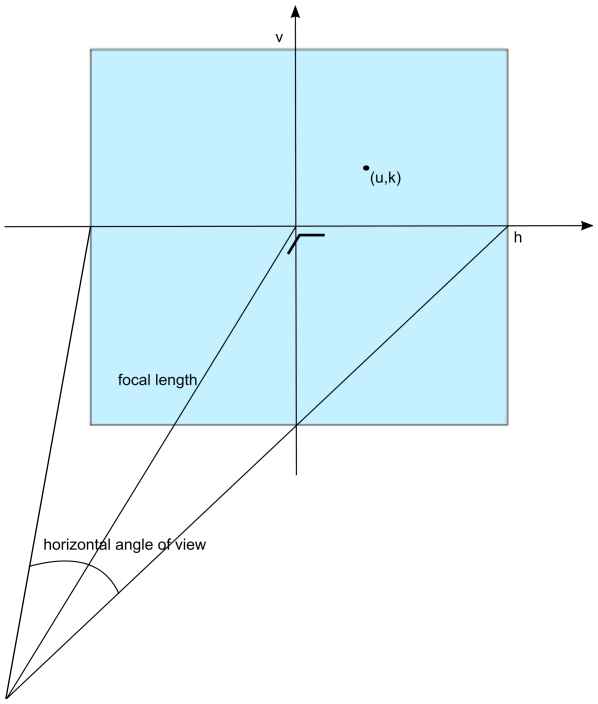


Figure 7: Image plane : How to convert a pixel to distances

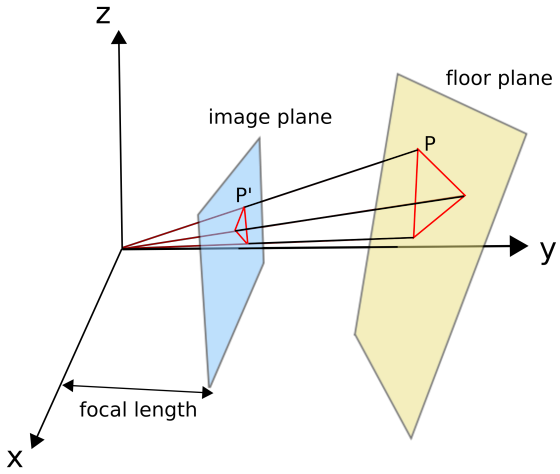


Figure 8: Projection using the pinhole camera model

4.4.2 Intersection between a plane and a line

Supposing we have P_0,P_1,P_2 three points linearly independent which are on the floor, and two other points L_A, L_B representing the line through the middle of the mark and through the center of the camera. As said in the article [4] on intersection between line and plane in wikipedia, the intersection

between the plane $P_0 + (P_1 - P_0)u + (P_2 - P_0)v$ and the line $L_A + (L_B - L_A)t$ can be expressed by :

$$R = L_A + (L_B - L_A) \cdot t$$

With t beeing calculated with :

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \begin{pmatrix} L_{Ax} - L_{Bx} & P_{1x} - P_{0x} & P_{2x} - P_{0x} \\ L_{Ay} - L_{By} & P_{1y} - P_{0y} & P_{2y} - P_{0y} \\ L_{Az} - L_{Bz} & P_{1z} - P_{0z} & P_{2z} - P_{0z} \end{pmatrix}$$

We have to take into account that there is no solution if the straight line is parallel to the floor which corresponds to Aibo looking far away, its head parallel to the floor. This case should not be a problem because if Aibo is looking far away it is not looking at the mark on the floor, and the calculation of the 3D point is done only if we have detected a point. The case of the straight line being inside the floor plane is physically impossible (head cannot go inside the floor). Finally we can conclude that the intersection between the plane and the line will always be a point and that this point is R.

4.5 Focal length and angle of view

To complete our task we must know the following about Aibo's camera : the focal length and the angle of view. They never change and are specific to the camera. We can either find them in the Model Information for ERS7[2] or calculate them.

4.5.1 Specification of the camera

According to the Model Information for ERS7[2], the color camera has the following specifications :

The number of picture elements :416(H) x 320(V), 30 FPS
Lens :F 2.8, f = 3.27mm
Angle of view :Horizontal angle 56.9 degrees Vertical angle 45.2 degrees
Default :White balance 5000K fixed, Shutter speed 1/100 sec fixed, Gain 0dB fixed
CMOS part :1/4 inch

The most interesting point is the angle of view and the focal length, both are necessary for our calculations.

4.5.2 Camera calibration

If we want to verify that these parameters are correct we can calibrate the camera. For exemple we could do the following steps:

- place on the floor a mark whose exact position we know;
- place Aibo's head in such a way that the camera reference frame is parallel to the floor - to simplify the problem;
- use the intercept theorem to find the focal length;
- determine the horizontal and vertical angle of view by placing two other marks whose exact positions are known.

4.6 Using distance sensor

Aibo also has a distance sensor in its head which has not been used in resolving our problem, because we wanted to use a solution only with the camera. However, here are some ideas on how we could have used its distance sensor :

- We could use it to determine the floor plane instead of the articulations of Aibo as explained in section 4.2.1. For example if we put a referential on neck Tilt2 (cf Fig. 3) and then take one measure of the distance to the floor, afterwards we could rotate around Tilt2 by an angle α to take another measure of the distance and finally rotate β to take the last measure of distance. The three measures construct a pyramid which helps us to calculate the plane in Tilt2 referential frame.
- Another utilization could have been to use the camera to move the head until the middle of the mark is on the line of the distance sensor. Then we know the distance to the mark from the head, the angles of the head thus we can calculate the position of the mark. The advantage of this solution would be that it will work even with a floor which is not flat as supposed in our solution.

5 Inverse kinematics

Knowing the position of the mark in the body referential, we first adapt the coordinates of the mark in the referential of the shoulder. Then we can move the paw using the inverse kinematics. From Uwe Düffert's thesis [7] we know how to obtain the direct and inverse kinematics :

First the direct kinematics, which allows us to find the position of the paw in the coordinate system of its shoulder, knowing the angle of each joint. See Fig. 9 for the reference of each angle and distance:

$$x = l_2 cos(\theta_1) sin(\theta_3) + l_2 sin(\theta_1) cos(\theta_2) cos(\theta_3) + l_1 sin(\theta_1) cos(\theta_2)$$
$$y = l_1 sin(\theta_2) + l_2 sin(\theta_2) cos(\theta_3)$$
$$z = l_2 sin(\theta_1) sin(\theta_3) - l_2 cos(\theta_1) cos(\theta_2) cos(\theta_3) - l_1 cos(\theta_1) cos(\theta_2)$$

The inverse kinematics is the problem of finding the angle θ_1, θ_2, θ_3 knowing the position x,y,z of the paw. We use the three equations of the direct kinematics and the cosinus theorem : $a^2 = b^2 + c^2 - 2bccos(\alpha)$. In fact this over-determines our problem, we have four equations and only three unknowns. Finally we find the position of each shoulder θ_1, θ_2, θ_3 (still from Uwe Düffert[7]) :

$$\theta_1 = \arctan(\frac{z}{x} + \arctan(\frac{(l_1 + l_2 cos(\theta_3)) cos(\theta_2)}{l_2 sin(\theta_3}))$$
$$\theta_2 = \arcsin(\frac{y}{l_1 + l_2 cos(\theta_3)})$$
$$\theta_3 = \arccos(\frac{(x^2 + y^2 + z^2) - l_1^2 - l_2^2}{2 l_1 l_2})$$

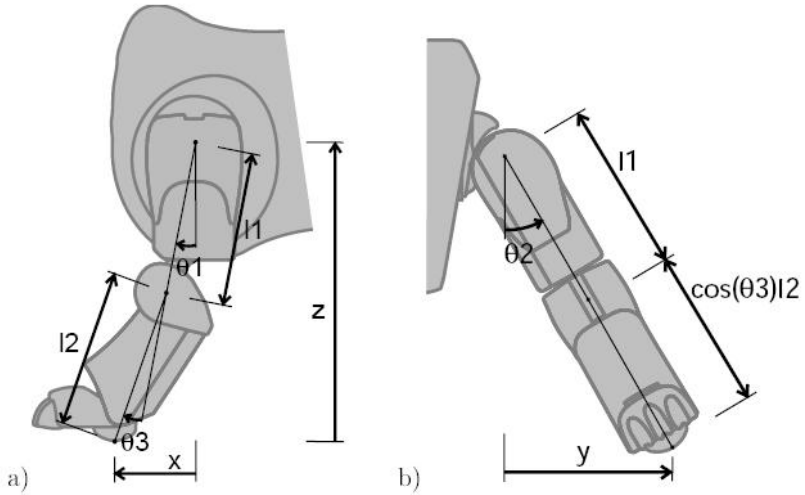


Figure 9: Angles and distances needed for inverse and direct kinematics taken from Uwe Duffert[7]

6 Implementation in simulation

The running application which will be put on the real robot functions as shown in the schema (Fig 10). The implementation was done in a few steps. First tests were done placing the mark anywhere in space, in a plane known to be parallel to the head. Then afterwards the plane became the floor and the mark was moved on the floor: a supervisor was done to test systematically a great number of positions of the mark. A few positions with different angles of the head were also tested. The last step was to make it work on the real robot. The following sections will detail each step.

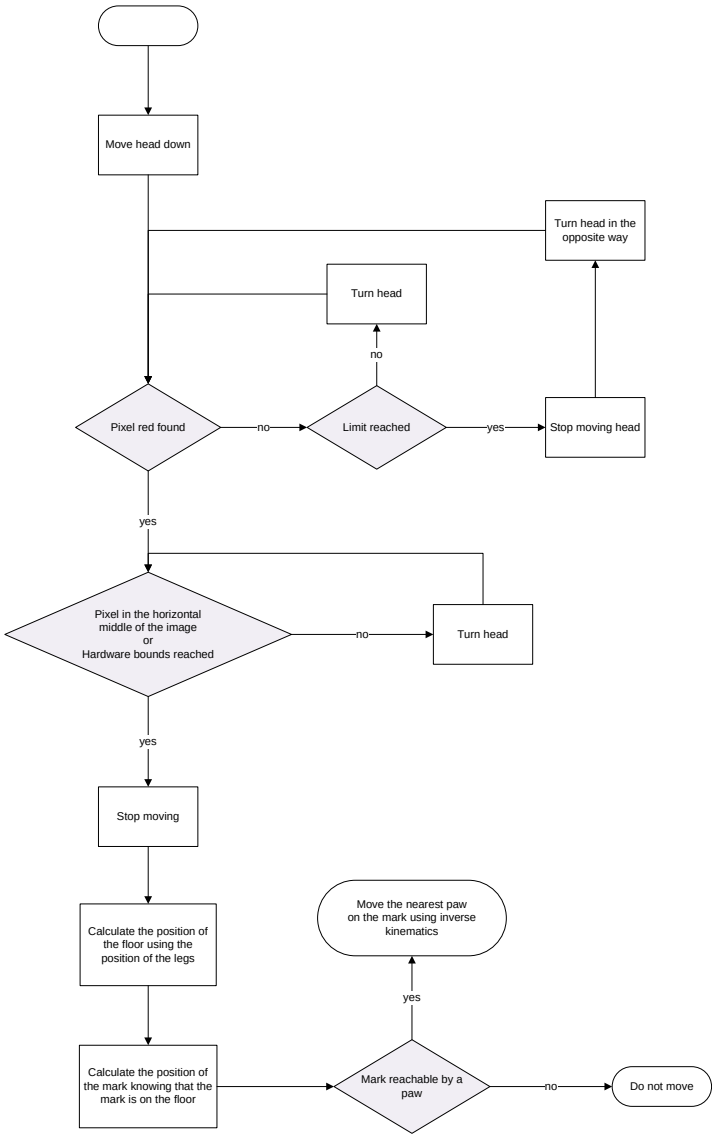


Figure 10: Schema of the application

6.1 Test knowing the plane

In this section we give ourselves the three points P_0, P_1, P_2 which we know are in the same plane as our mark. The mark will be placed artificially in the position using the scene tree of webots - we create a node children of tilt1 and so its coordinate will be in the same referential as the referential used in our calculations. Then we proceed with some tests to check the correctness of the position of the mark. Observing the tests has helped us to improve the precision of the mark position. For example the following formula from cyberbotics[3] $verticalFOV = \frac{fieldOfView*height}{width}$ which I was using was discovered to be unapplicable because the result found is not the same as in the camera specifications. In fact doing the tests showed me that the incertitude was growing towards the side of the picture but only on the vertical axe and not on the horizontal axe. It could have been normal for a phenomenon of distortion but in this case it would not have been only on one axe.

In Table 3 we place a mark at position (0,800,65.4)mm expressed in the body ref. frame - this position is chosen to center the mark in the middle of the image, we then make a few mesures changing the size of the mark. We want to verify that the position does not change because if it does it would mean that there is a problem calculating the middle of the mark.

For the next tests we are going to fix a size of the mark at : 0.01 x 0.01 m.

In table 4 we modify only the coordinate on x axis. We want to verify that the imprecision does not grow too much as the mark approaches the side of the image to see if we are victims of a phenomenon of distortion. Table 5 is also done for the same reason. We change the coordinate z. We can observe in both cases that the imprecision does not grow far away from the center of the image.

In table 6 we modify the distance between the camera and the plane in which lays the mark, coordinates x and z remain unchanged. As the plane is defined precisely the coordinates of y will always be precise but we can see as the distance increases the incertitude on x and z grows. The reason is the approximation that we use with our camera model. A pixel is a unit which cannot be divided into smaller elements therefore the precision of our calculation depends on the resolution of the camera. When the mark is further away the number of pixels that will be used to represent the mark will be less than if it is near. Therefore, a more distant pixel will be mapped to a 3D coordinate with a bigger imprecision.

To see if this assertion is correct we will try another approach : suppose we know the position of the mark and that we want to determine to which pixel it corresponds. This problem is much more simple : mapping 3D coordinates to a 2D coordinate in an image. In the book Robot Modeling and Control[5] they describe the mapping from a 3D world ccordinate to a pixel in the image. Knowing that x,y,z is the position of the 3D point(see Fig 11), λ is the focal length, s_x is the horizontal size of a pixel : $\frac{tan(\text{horizontal angle of view}/2)*\text{focal}}{\text{width of image}/2}$. Similarly s_y is the vertical size of a pixel. We calculate c and r the pixel positions (from the center of the image) Using this equation from Robot Modeling and Control[5]:

$$r = \frac{\lambda}{s_x}\frac{x}{z} \quad c = \frac{\lambda}{s_y}\frac{y}{z}$$

For example if we take the position in mm (2,800,2) and we calculate r=0.47 and c=0.48 we can

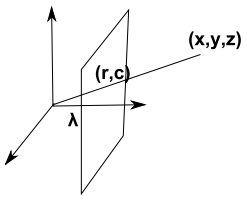


Figure 11: Mapping 3D to 2D point from Robot Modeling and Control[5]

see that the result is fractional and so as a pixel cannot be divided we see that we have anyway an approximation, so the positions (0,800,0),(1,800,1) and (2,800,2) map to the same (0,0) pixel, in other words a pixel at position (0,0) could map to each of them, hence the imprecision. We then calculate

with a nearer position : the position (2,100,2) gives us r=3.83 and c=3.84 which is different from the result from (1,100,1) and (0,100,0) so we can distinguish them, hence nearer the mark more precise will be the result. We can also say that more the mark is distant more it means that an imprecision as to which pixel is the center of the mark leads to an imprecision much bigger in the real world.

We finally want to check with a different angle of the head, to see if rotating the head brings an incertitude. Table 7 modifies the angle around Tilt1, the mark stays in the same position (0,200,80). The two other tables modifying the angle around Tilt2 and the other angle around Tilt1 will not be reported here because they still have the same imprecision 1-2mm as in the other tables and are therefore not interesting. The imprecision that we still have for these tests is around 2mm. The reason is the imprecision that is introduced each time we change referential; matrice calculations introduce imprecison.

Mark size	Mark position calculated
0.01 x 0.01	(-1.87279,800,67.2704)
0.1 x 0.1	(-1.87279,800,67.2704)
0.2 x 0.2	(-1.87279,800,67.2704)

Table 3: Changing size of the mark

Mark position	Mark position calculated
(0,500,80)	(-1.09131,500,80.1247)
(100,500,80)	(98.218,500,80.1247)
(150,500,80)	(148.418,500,80.1247)
(200,500,80)	(198.619,500,80.1247)
(220,500,80)	(218.262,500,80.1247)

Table 4: Modifying only x coordinate

Mark position	Mark position calculated
(0,500,80)	(-1.09131,500,80.6589)
(0,500,100)	(-1.09131,500,101.368)
(0,500,200)	(-1.09131,500,195.819)
(0,500,220)	(-1.09131,500,220.169)
(0,500,-100)	(-1.09131,500,-99.1786)

Table 5: Modifying only z coordinate

Mark position	Mark position calculated
(0,200,80)	(-0.309831,200,80.253)
(0,300,80)	(-0.570324,300,80.2096)
(0,600,80)	(-1.3518,600,81.601)
(0,1000,80)	(-2.39378,1000,82.1352)

Table 6: Modifying only y coordinate

α	Mark position calculated
0	(-1.09131,500,80.6589)
-0.1	(-1.06409,500,79.6456)
-0.2	(-1.02387,500,80.1457)
-0.25	(-1.99863,500,79.9168)
-0.28	(-0.983289,500,79.592)

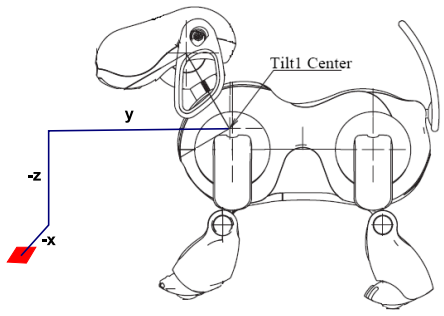
Table 7: Modifying α without changing mark position

6.2 Test using supervisor

To achieve systematic tests I have done a supervisor that moves a mark on the floor. The mark is placed in the initialization of the program and moved in each iteration. As the floor is not parallel a lot of tests have been done to see how the mark on the floor can be moved (not too high, not too low as it will not be visible).

The three points P_0, P_1, P_2 are now calculated using the position of each servo and the dimensions of Aibo's leg. For each position known x,y,z of the mark we calculate the position x_c, y_c, z_c using the camera and the algorithm described in previous sections which calculates the position of the mark using the image of the camera and knowing the articulations of the head.

In table 8 we can see a printscreen of the simulation : we clearly see the zone visible by the camera (the purple lines) and the window "Aibo(don't hide)" represents the vison of the camera. To see the evolution of the mark we will do some plots using the coordinates x and y of the mark as the horizontal and vertical axis, and each point at coordinate (x,y) on the plot will represent the corresponding incertitude : the surface of the point equals the corresponding incertitude.



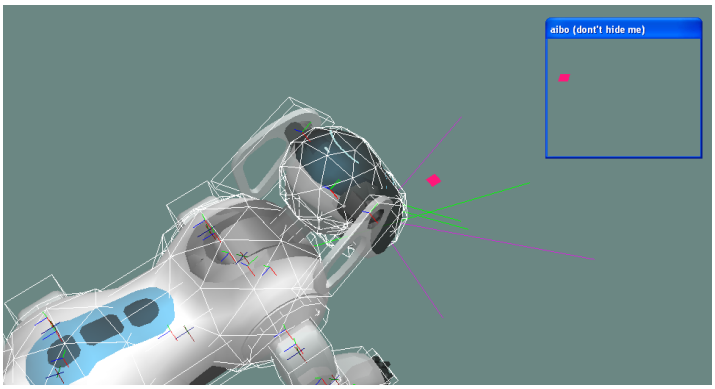


Table 8: First figure shows the distances x,y,z in the referential of Tilt1, second figure shows a printscreen during the simulation

We will start by placing the head of Aibo in a position looking down on the floor. For this reasons we choose the following angles around tilt1 and tilt2 respectivly : -1.2,0.2 radian and we will not use the pan rotation (head is rotated only around x). Figure 12 shows the incertitude on x : $|x - x_c|$ figure 13 shows incertitude on y and figure 14 shows incertitude on z.

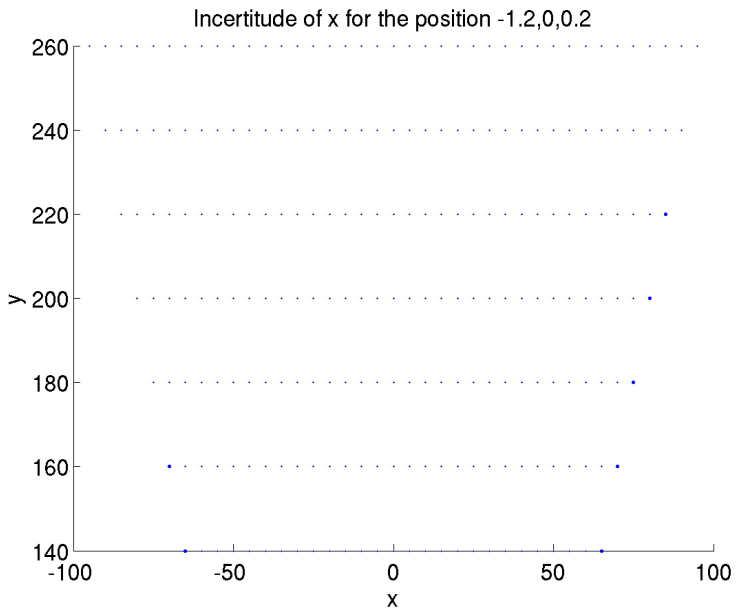


Figure 12: Incertitude on x in position -1.2, 0, 0.2

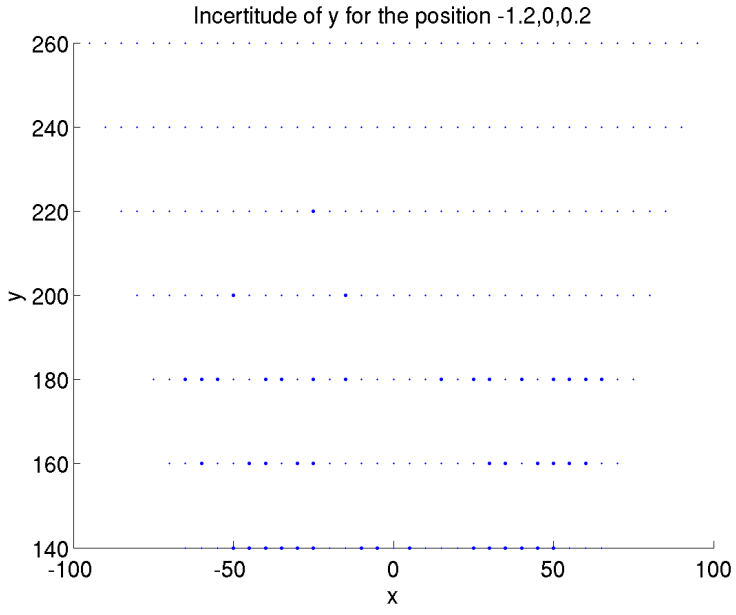


Figure 13: Incertitude on y in position -1.2, 0, 0.2

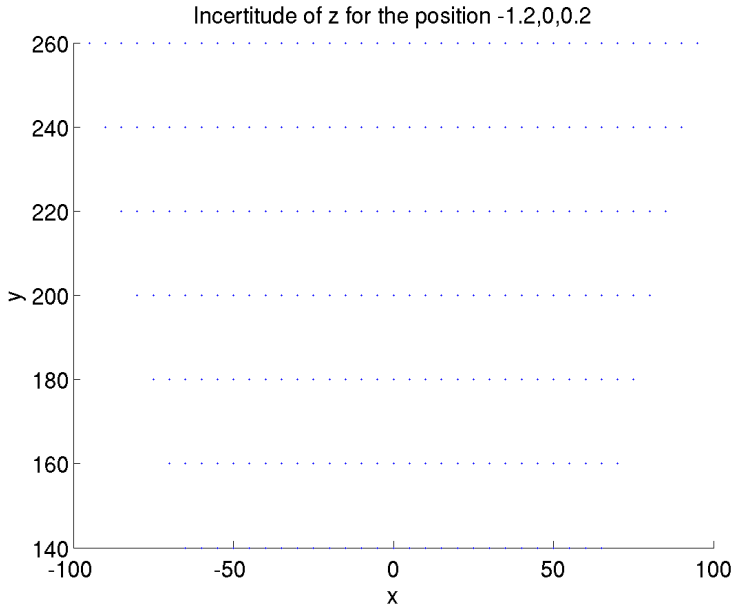


Figure 14: Incertitude on z in position -1.2, 0, 0.2

For a more quantitative analysis Table 9 shows the first quartile , the median and third quartile of the values of the previous plots (in the same order x,y,z).

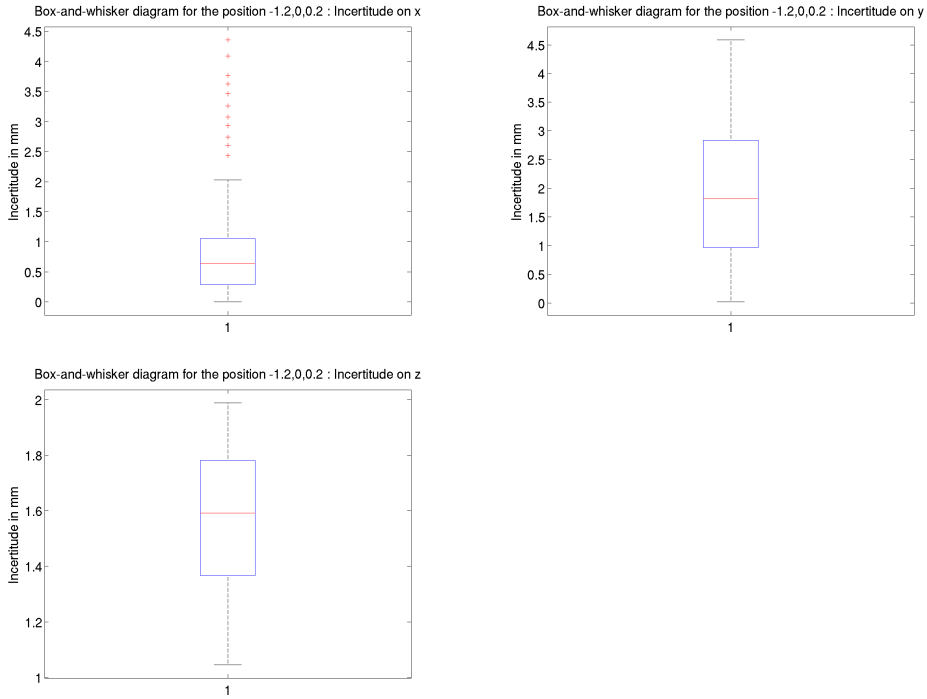


Table 9: a) Incertitude on x : $x - x_c$ b) Incertitude on y : $y - y_c$ c) Incertitude on z : $z - z_c$

In figure 12 we can see that the incertitude on x is small. As we can see in table 9 the average value is of 0.6 mm which is very good. The "extreme" outliers values that we have are still less than 4.5 mm and are situated in the border of the image. The reason why it is bigger in the border is that the mark is only half visible and so the middle of the mark is calculated only on the visible part of the mark.

In figure 13 the mean incertitude on y is of 1.7mm. It is bigger than the incertitude on x. We think that the reason for this little difference is the resolution of the camera which is not the same horizontally as vertically. We can also notice that sometimes incertitude seems bigger but not uniformly. This is due to the fact that the shape of the mark is never the same and so because the middle of the mark depends on the shape of the mark this can give irregular incertitudes. Anyway we can see that these incertitudes, which are about 3 mm (not too big : the size of the mark for these tests is of 1 cm), are not significative.

In figure 14 we can see that the incertitude is about 1.6 mm and looks uniform. The difference between the first quartile and third quartile is of 0.4 mm and so we can see that globally the incertitude stays uniform. This incertitude is only introduced by the calculation of the distance and orientation of the floor. The calculation of P_0 , P_1, P_2 are based on the geometry of the robot but cannot be exact: for exemple, do we calculate the distance to the floor with the touch sensor pressed ? And where exactly is the contact point (the end of the paw is round)? Anyway the incertitude stays reasonable. In practice the incertitude of z is less important because Aibo will anyway advance its paw until it touches the floor.

From now on we will interest ourselves in the incertitude on each axis calculated using :

$$i = \sqrt{|x - x_c|^2 + |y - y_c|^2 + |z - z_c|^2}$$

We want to see how this incertitude evolves in other positions of the head. We first start studying the incertitude in the same position as already done then we will turn the head left, up and finally up and left, to get a square.

Table 10 shows the evolution of the incertitude for (from top left to down right):

- an angle of -0.8 rad on Tilt1, 0 on Pan rotation, 0.1 on Tilt2
- an angle of -0.8 rad on Tilt1, -1 on Pan rotation, 0.1 on Tilt2
- an angle of -1.2 rad on Tilt1, 0 on Pan rotation, 0.2 on Tilt2
- an angle of -1.2 rad on Tilt1, -1 on Pan rotation, 0.2 on Tilt2

Table 10 is useful for a qualitative analysis but to see the corresponding values we also plot the first quartile, the median and third quartile as done previously in table 11. Table 12 also gives us a small sample of the values used for the plots to give us an idea for a position(x,y) which is represented by the corresponding superficies of the points in table 10.

We can see that the incertitude grows as the mark is further away. We think this is mainly due to the limitations imposed by the camera (as explained in section 6.1). In addition, the further away the mark, the error as to which pixel is in the center of the mark will have a greater effect on the real coordinates. Some other causes of incertitude can be :

1. The imprecision of the calculation of the three points P_0, P_1, P_2. As the floor is not parallel to the referential an imprecison on these points will be more noticeable as the distance to the mark increases.
2. The imprecision on the placement of the mark - is it effectively on the floor? The method used in the supervisor was to move the mark on the floor (the mark being in the referential of tilt1) and as the floor was not parallel in tilt1 referential we had to find manually which increment was needed at each iteration to be able to lay the mark on the floor.
3. The imprecision of matrice calculation for each transformation of referential.

4. Imprecisions inside the scene tree: the initial placement of the mark.

Incertitude is sometimes bigger at the end of the scanning line, because the mark is only half visible - so why is it not always the case? We think it is because the size of the zone visible for the camera is not a square (because of the orientation of the head) and so sometimes the mark can be entirely inside or outside and sometimes partially inside and outside. In fact it depends on the shape of the mark which is visible.

Incertitude does not seem to be affected a lot by the rotation of the head, but more by the distance of the mark. Figure 15 (top left in table 14 is particularly interesting because we see that imprecision increases clearly on the left and on the top of the figure.

To see another extreme example for far away values we will put Aibo's head nearly parallel to the floor (but enough inclined so that it can see the ground) : (tilt1=-0.2 pan=0 tilt2=0.1). In table 13 we see clearly the evolution of the incertitude growing bigger as distance increases, anyway even with a distance of 80 centimeters the incertitude remains less than 3cm which can give Aibo the good direction to the mark. If the mark has to be calculated from far away, the position can be calculated again when nearer. The only problem is the size of the mark: from far away if the mark is too small it could be impossible to see it. So if our solution has to be used from far away the mark has to be bigger. In fact these measures have been done to see the evolution but will not be needed in our case, because they are completely unreachable when Aibo does not move.

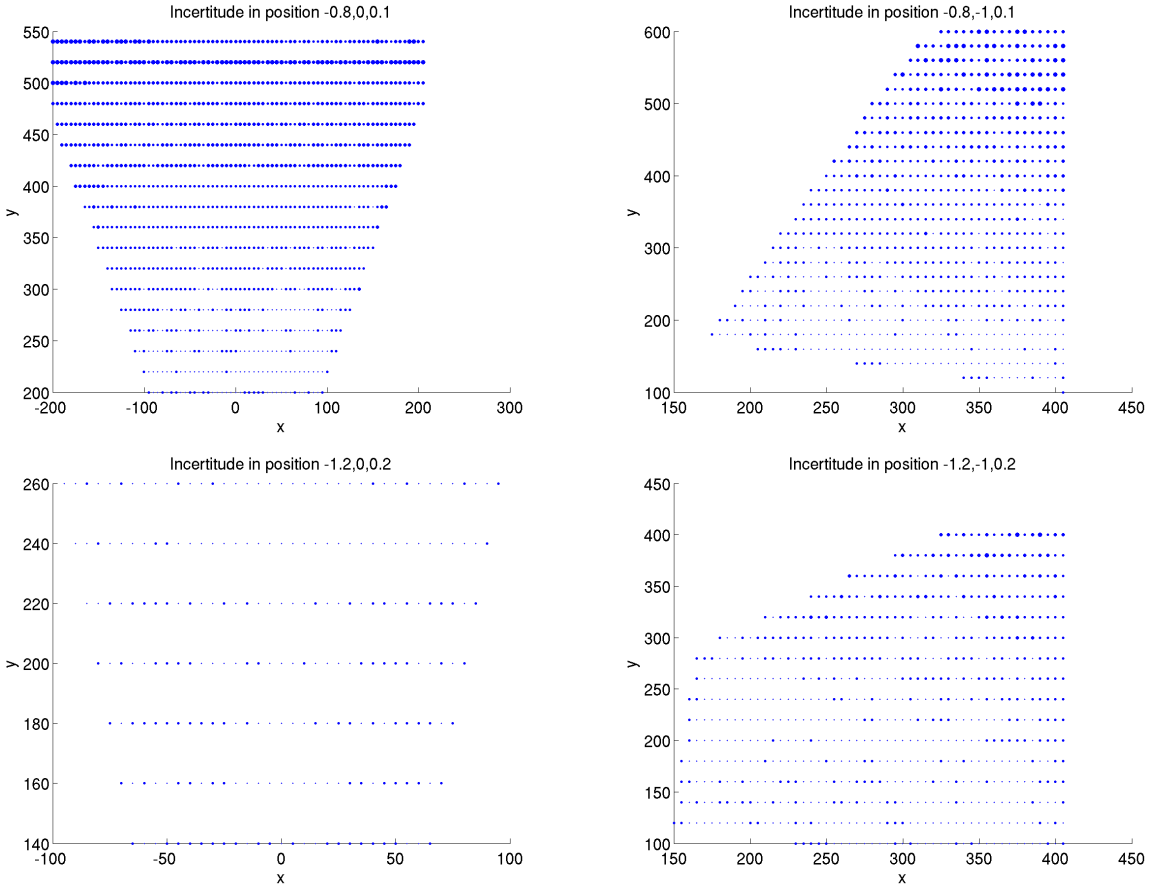


Table 10: Incertitude between the position calculated and the real position of the mark for qualitative analysis

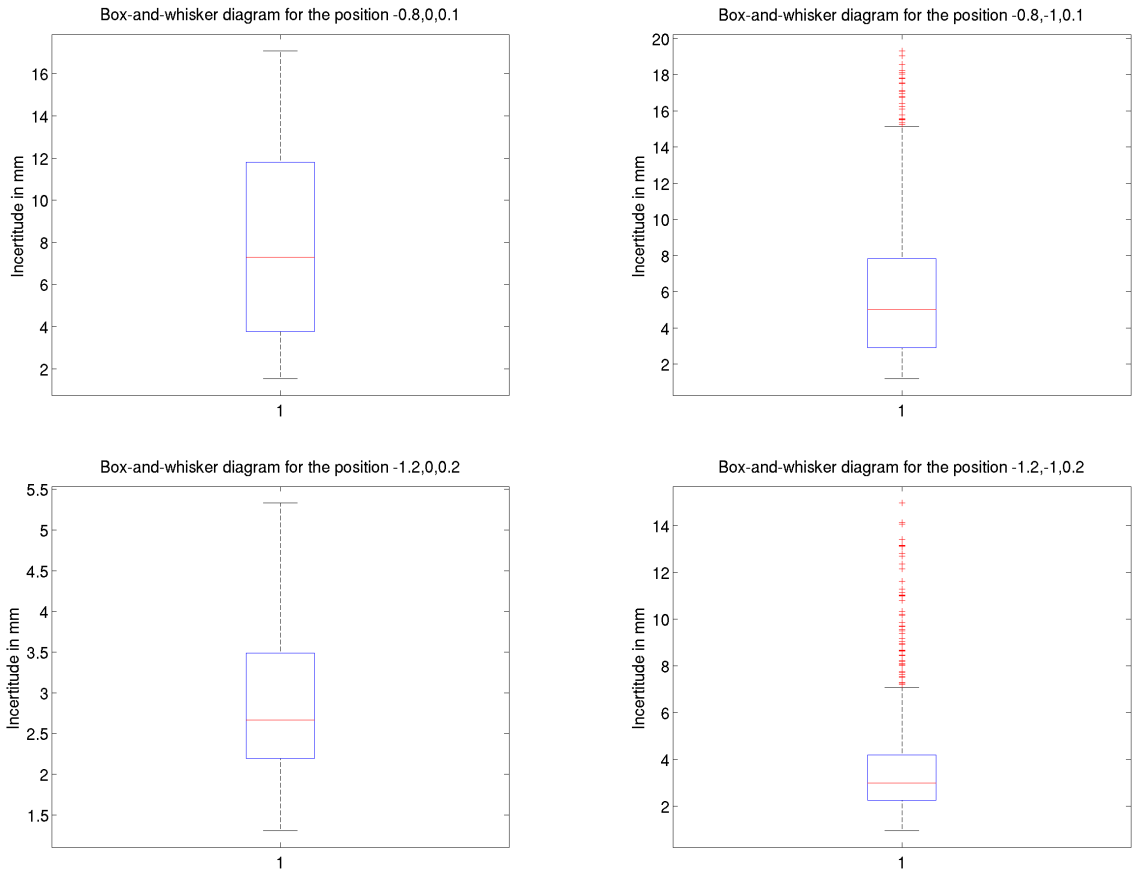


Table 11: Incertitude between the position calculated and the real position of the mark for quantitative analysis

Mark position	Incertitude at position -1.2, 0, 0.2
x= 25.00 , y= 140.00	3.9463
x= 0.00 , y= 140.00	1.6872
x= 95.00 , y= 260.00	3.4477
Mark position	Incertitude at position -1.2, -1, 0.2
x=259.99 , y=100.00	1.24
x=489.99 , y=100.00	2.52
x=894.99 , y=300.00	7.96
Mark position	Incertitude at position -0.8,0,0.1
x=35.00 , y=200.00	1.77
x=40.00 , y=220.00	2.43
x=199.99 , y=540.00	13.86
Mark position	Incertitude at position -0.8,-1,0.1
x=369.99 , y= 120.00	1.45
x=339.99 , y= 160.00	2.70
x=399.99 , y= 600.00	11.05

Table 12: Sample of incertitude values given for each position

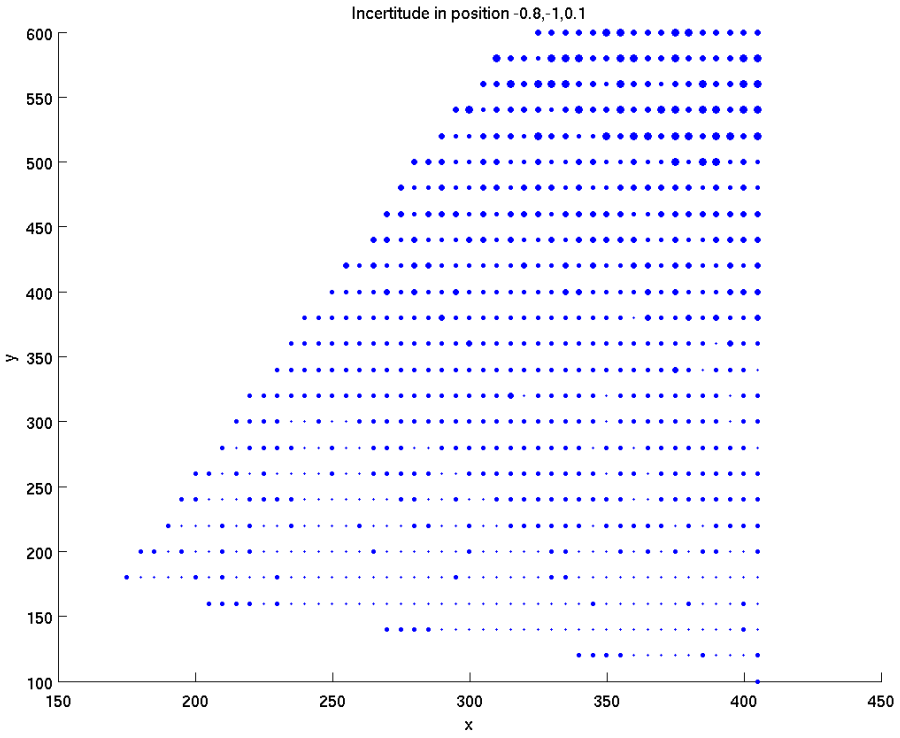


Figure 15: Incertitude in the following position : an angle of -0.8 rad on Tilt1, -1 on Pan rotation, 0.1 on Tilt2

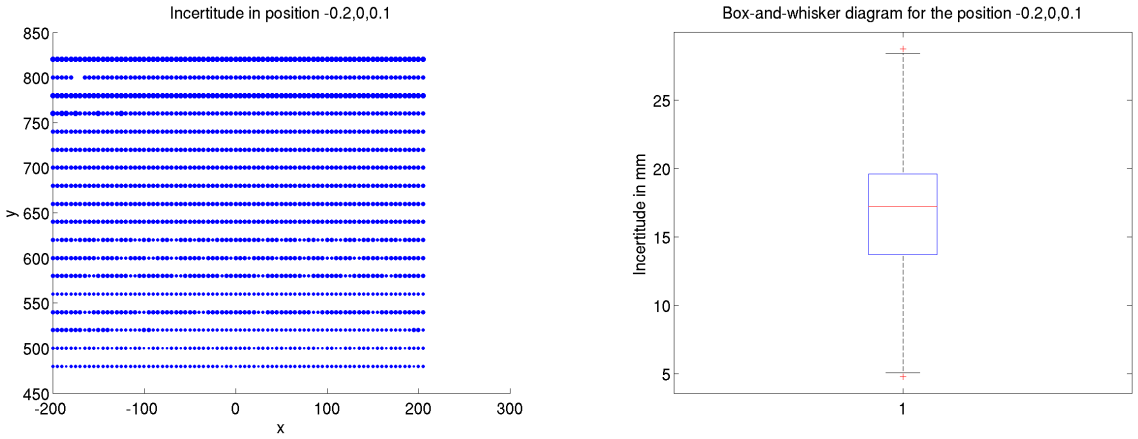


Table 13: Incertitude looking far away

6.3 Test with inverse kinematics

Finally after having done all the tests to be sure that the location of the mark using the camera was correct, the inverse kinematics - as presented in section 5 was added. The main problem using the inverse kinematics was that all positions of the mark that the camera could see were in fact not reachable by a leg. Reachability is determined by the fact that the value of the equation $|\frac{(x^2+y^2+z^2)-l_1^2-l_2^2}{2l_1l_2}|$ must be less than one.

In the position in which all the tests have been done the distance from the floor is 140.5 mm, so we calculate the circle $x^2 + y^2$ reachable knowing that z=140.5 and $l_1 = 70.1$, $l_2 = 76.9$:

$$\frac{(x^2+y^2+z^2)-l_1^2-l_2^2}{2l_1l_2} \leq 1$$
$$x^2 + y^2 \leq 2l_1l_2 + l_1^2 + l_2^2 - z^2$$

So the radius of the reachable circle is 43 mm and the nearest the camera can see is in fact about 140 mm. The solution would be to change the position but the risk is that P_0, P_1, P_2 would be inexact because of the assumption of the contact point between the paw and the floor: the touch sensor is round and if we change the position the contact point must not be too far away from the one supposed. The actual position was chosen because it is the same position as in the Model Information for ERS7[2] and so it was easy to check the position of the floor.

The solution was to assume that the leg was bigger than it was, and to fix the angle θ_3 to zero (in fact 30 degrees because of the shape of the leg). So by this artifice, Aibo falls down almost on the mark. It is not exact because when it falls down the angles of each servo calculated by the inverse kinematics do not consider the last move : the fall.

Table 14 shows Aibo looking for the mark, when it sees it : moving its leg, and falling on the mark and in the last picture we clearly see that the rear paw is no longer touching the floor.

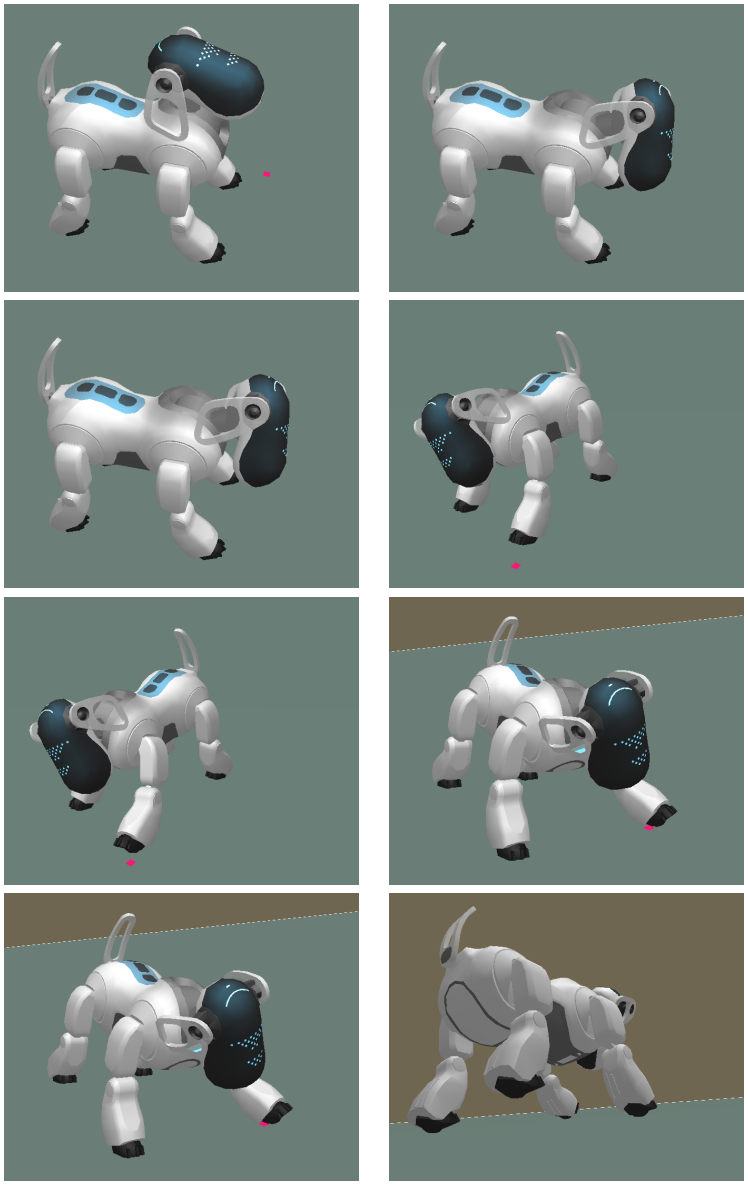


Table 14: Aibo simulation

7 Implementation on the real robot

The last step was to make it work on the real robot. To achieve this the controller has to be cross-compiled using openr and a project previously done by Lukas Hohl[9] - called RCServer which translates some functions written for simulation in webots into openr functions, camera remote control and translation between camera webots functions and openr functions were already done by Raphaël Haberer-Proust[8] but some little modifications had to be done to make it work (see Annex A for details).

The test on the real robot have been done in 3 steps (testing the camera, testing the program that works in simulation, debugging and improving knowing the results) :

First to be sure the mark was different from its environment we printed an 'r' when the function which checks if the pixel is red returns 1. We tested the environment without the mark and with the mark. Sometimes red pixels were detected when the mark was not present (so this has to be taken into account for the future tests).

Then we tried the same application as done in simulation : Aibo bends its head down, turns left and right and if it sees the red mark it advances its paw on the mark. First it was not moving at all because I was trying to put Aibo in a specific position : and the calculation of the mark could start only when we were sure that it was in this position : this could not work because the value returned by the servos were never the same as wanted and sometimes the difference was even up to 7 degrees on one leg. So the tests to check that it was in the "correct" position were removed. In fact there is no correct position but they were there to be sure that Aibo would not start calculating the position of the mark and then moves its leg while doing the calculations. The other problem was that Aibo moved its leg even with no mark. This was due to the fact that there is some noise in the image taken by the camera and there is sometimes a pixel detected as red depending on the light around the aibo. So to summarize the problems :

- Some noise in the picture is always there so to be sure that it really is the red mark and not a noise a minimal number of pixels must be counted
- The application depends too much on ambiant light.

Finally some modifications have been done on the controller. When Aibo seeks the mark if it sees a number of pixel less than 5 in the picture it does not consider it as the mark - to avoid considering the noise as being the mark. In fact another modification has been done to try to improve the result when Aibo sees something that it considers would be the mark, it continues turning its head to center the hypothetical mark in the center of the image : when it is centered or that it has reached some hardware bounds it will stop moving and then calculate the center of the mark.

Quantitative tests can be made but to obtain more interesting results the problem of Aibo "falling down on the mark" has to be solved first. Because we currently see not only the incertitude between the real position of the mark and the one calculated but also the incertitude of the inverse kinematics as explained in section 6.3. Anyway some videos have been done and can be visualised to see the results :

- http://birg.epfl.ch/webdav/site/birg/users/161024/public/aibo1.avi
- http://birg.epfl.ch/webdav/site/birg/users/161024/public/aibo2.avi

We can see that depending on which experiment Aibo moves its paw near the mark or too far away. We can compare these videos with the ones done with exactly the same controllor but in simulation :

- http://birg.epfl.ch/webdav/site/birg/users/161024/public/aibo_sim1.mpeg
- http://birg.epfl.ch/webdav/site/birg/users/161024/public/aibo_sim2.mpeg
- http://birg.epfl.ch/webdav/site/birg/users/161024/public/aibo_sim3.mpeg

We can see that in simulation results are better than in reality (but more the mark is far away and more the problem with the inverse kinematic is important). To achieve tests on the real robot without this problem we could sit Aibo and make it touch marks in front of it putting the mark on a plane that we could determine using the distance sensors (for example). The tests that we can see in the video have been done one after the other on a black floor. Some other tests have been done on a white surface and with a redder mark. We can see that the difference between the paw and the mark can be up to 3 centimeters which is quite big in comparaison with the results in simulation. In fact some of the tests with mark placed approximately in the same position never gave exactly the same movement. We think that the principle reason for the difference between simulation and the real world is the "falling down on the mark" which is more important in the real world but we can suppose some other reasons :

- The shape of the mark will be more uncertain than in simulation.
- The ruggedness of the floor : does Aibo slip on the floor?
- When Aibo falls on the mark is the angle of the articulation modified by the acceleration of the fall? The resistance of the floor and the moments of the resulting forces could they change the position of the servo?
- Are the values of the servos exact ? Or could it be possible that Aibo has an incertitude on the real position of its articulations hence a bad calculation of the floor?
- There may be other reasons related to the real world.

To test the vision part we can draw on the floor the position of the legs and then place Aibo on these marks and look at the values calculated of the mark returned by telnet. The problem with these tests are that they are really imprecise and cannot validate or unvalidate our results. They are imprecise because we have to estimate the middle of the paws and to place Aibo on the mark done for its paws : this is too experimental to give good results. We also estimate the distance to the referential tilt1 which depends in fact if we have correctly placed Aibo on the marks. Five mesures have been done they have an imprecision of 1 to 2.5 centimeters and are anyway better than the point on which Aibo falls. Some more precise and reliable tests could be done if we could precisely compare the values returned by the application with the values of the mark placed in the real world.

To improve the application and to make it less dependent on its environment a change can be done in the color system : at present the color detection is done using rgb but it could be replaced by another color system. For example the system color YCrCb could be better because there are only two chroma components for three in the rgb system. In fact Aibo already returns values in YCrCb which are converted in rgb by the controller of RCServer, so the only thing to do is to implement a function that returns directly these values instead of converting them in rgb. Fig 7 shows how light can influence the color of the mark, the picture has been taken with the remote control of RCServer and two spotlights in front of Aibo. Another idea is to make Aibo independent of the light, and even independent of the color of the mark, would be a phase of initialization : we could show to Aibo the color that it has to look for and then it could know exactly what are the values it is looking for in the second phase.

Table 15 shows the robot Aibo in different steps : seeking the mark, and then after having found it : moving its paw near the mark, we can also see the rear leg as on the simulation being lifted up. These images have been taken from the videos of the real Aibo.

Figure 16: Photo of the mark taken by Aibo

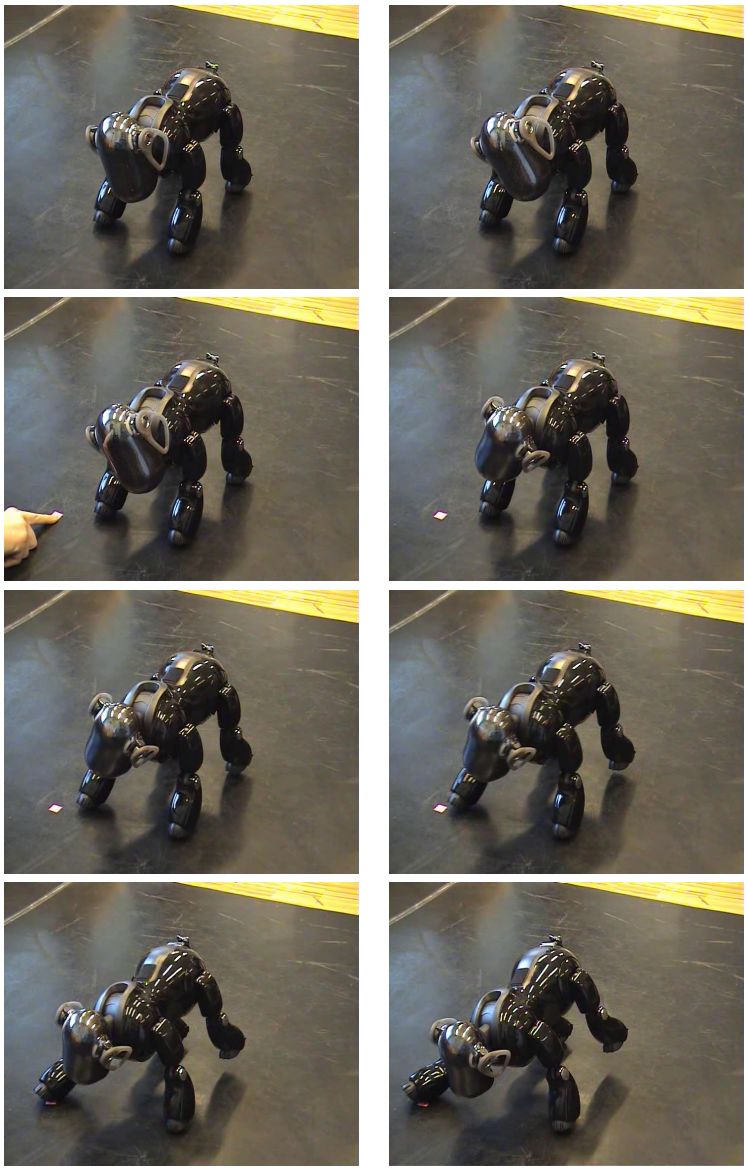


Table 15: Real Aibo moving

8 Conclusion

In this project a solution for the problem of localization of the mark has been implemented using the color of the mark and the position of each articulation of Aibo. The solution uses transformations of referentials, trigonometry, the measures of the real Aibo, geometric transformation and the characteristics of the camera. The camera is considered as being a pinhole camera model, we use the focal length, the field of view and the fact that the mark is on the floor to find the 3D position of the mark.

After having developed the theory the solution has been implemented and tested in simulation. Some systematic tests in a few positions of the head have been done to validate the results. The solution developed is independent of the position of the head. However the position of the articulations of Aibo's legs must be the same for right and left legs and the distance between the paws is fixed (see section 4.2.1) so there is 4 degrees of freedom in the placement of Aibo's body. Another limitation is that Aibo must have a contact point with the floor on each of its paws, and not too far from the contact point given in the Model Information for Ers7 [2], I mean that Aibo must not touch the floor with its knee in place of its paw (for example). All systematic tests have been done in one position of Aibo's body but the solution developed should be applicable in other positions of the articulations (for more degrees of freedom direct kinematics could be used). Finally the solution has been adapted and tested on the real robot.

The solution chosen is efficient but only in certain conditions of the environment : light, absence of any other red object near Aibo during the experiment. In simulation results are quite good but to validate these results with the real robot some more precise tests have to be done. However, the supervisor developed to see the imprecision proves that this solution can be used to localize a mark using only the camera. The imprecisions in localizing the mark are essentially introduced by the limitations of the camera and the different calculations that have been done to obtain the position.

To continue the project the localization can be used for a mark further away, and displacement of Aibo can be added, the calculation has to be done again as it walks towards the mark. The tolerance of the color to the light could be compensented by adjustment before starting the experiment; one could adjust the white balance of the camera for example. To make the solution even more independent of the light and the colors, one can imagine that the mark could be with a very specific drawing that could be recognized in any position, hence we would be sure that it is not an element of the environment.

References

[1] Richard Hartley and Andrew Zisserman, Multiple View Geometry in Computer Vision Second Edition, Cambridge University Press, 2003

[2] Sony, OPEN-R SDK Model Information for ERS-7,Sony Corporation,2004

[3] Webots, http://www.cyberbotics.com, Cyberbotics Ltd.

[4] Wikipedia, http://en.wikipedia.org/wiki/Line-plane_intersection

[5] Mark W. Spong, Robot Modeling and Control

[6] Alessandro Rovetto, Francesco Scandelli thesis: Semi-Autonomous Navigation of a Legged Robot using Monocular Vision, 2005, http://www.sais.se/mthprize/2005/rovetto_scandelli2005.pdf

[7] Uwe Düffert Diploma Thesis, Quadruped Walking modeling and Optimization of Robot Movements, 2003, http://uwe-dueffert.de/publication/dueffert04_diploma.pdf

[8] Raphaël Haberer-Proust, Semester project, 2005-2006, Remote control of AIBO camera from Webots, http://birg.epfl.ch/page59430.html

[9] Lukas Hohl summer, Semester project, Aibo Simulation in Webots and Controller Transfer to Aibo Robot, http://birg.epfl.ch/Jahia/site/birg/op/edit/pid/43234

[10] Sony, OPEN-R SDK Programmer's Guide,Sony Corporation,2004

Annex A

Suppose we have an Aibo model Ers7 what should be done to program it using webots?

1. Download webots from http://www.cyberbotics.com/products/webots/download_linux.html , create an environment variable called WEBOTS_HOME pointing on the directory where webots is installed.
2. Download openr from http://www.cs.cmu.edu/ tekkotsu/openr-install.html , unfortunately Sony has closed its download and support page for openr - so this page is not the official one. Follow the instructions on the web page. Do not forget to create an environment variable called OPENRSDK_ROOT if your installation is not in /usr/local/OPEN_R_SDK.
3. Download source files of the project called RCServer from http://www.cyberbotics.com/cdrom/common/devel/webots-openr-0.3.1.tar.bz2 , then follow the instructions inside the readme: copy the folder src from the package to the webots installation folder to have the following architecture : /webots/src/lib/openr/, also copy the folder transfer to the installation folder of webots. To compile RCServer go to /webots/src/lib/openr/rcserver/ directory and type make. After the compilation the folder transfer/openr/OPEN-R/MW/OBJS contains the executables that are needed to copy onto the memory stick in the folder open-r/mw/objs. This is needed only the first time and if modifications of the controller of RCServer have been done.
4. Check if the memory stick contains the architecture described on page 24 of the Programmer's Guide[10]

After having developed a controller that works under simulation of webots we want to compile it for Aibo :

- Take the makefile called Makefile.openr given with webots in the folder projects/robots/aibo/controllers/ers210 copy it to the folder which contains our controller.
- Compile it using the command make -f Makefile.openr, a folder named OPENR is created, take the CONTROLL.BIN inside the OPEN-R/MW/OBJS built and copy it onto the memory stick in the folder open-r/mw/objs
- Insert the memory stick into Aibo

To make the wireless lan work adapt the file wlanconf.txt which is inside the memory stick in the open-r/system/conf folder. To use telnet : type telnet lslaibo1 59000 (replace lslaibo1 with ip and 59000 with the port). To use distance control from the webots simulation, a double click on the robot Aibo opens a window and then go under the tab "Configuration" enter ip of Aibo and connect.

Here are the modifications done to make the camera work :

In the folder /webots/src/lib/openr/rcserver/RCServer in the file robot_data.h under model Ers7 substitute the line :

"PRM:/r1/c1/c2/c3/il-FbkImageSensor:F1", // COLOR CAMERA

by :

"PRM:/r1/c1/c2/c3/i1-FbkImageSensor:F1", // COLOR CAMERA

In /webots/src/lib/openr/rcserver/Controller in the folder stub.cfg (in this file the inputs and outputs of each openr object are described) the following line has to be added :

Service : "Controller.Image.OFbkImageVectorData.O", null, NotifyImage()

Verify that the file config.cfg onto the memory stick in the folder open-r/mw/conf contains the 2 following lines:

OVirtualRobotComm.FbkImageSensor.OFbkImageVectorData.S RCServer.Image.OFbkImageVectorData.O

OVirtualRobotComm.FbkImageSensor.OFbkImageVectorData.S Controller.Image.OFbkImageVectorData.O