saBallTracking: A Comprehensive Description

By JingJing Xu

Contents

Abstract

This report is a comprehensive description of the ball tracking behaviour, saBallTracking, found in the Robocup 2003 rUNSWift code repository (version 2134). The function first calculates the position of the ball in a camera relative coordinate system, then transforms it into a coordinate system relative to the base of the neck of the robot, accounting for the original heading, tilt and turn. This results in a somewhat slow but rather accurate estimation of the heading and tilt of the top centre of the ball.

Introduction

saBallTracking is a behaviour found in the Robocup 2003 rUNSWift code repository (version 2134), to make a robot dog visually follow the ball by moving its head. Since the code itself lacks documentation, and previous theses lack details and references to the code, this document is an attempt to thoroughly explain the purpose of each significant line of code, the mathematics behind it, parameters used, and to assist readers' understanding with diagrams.

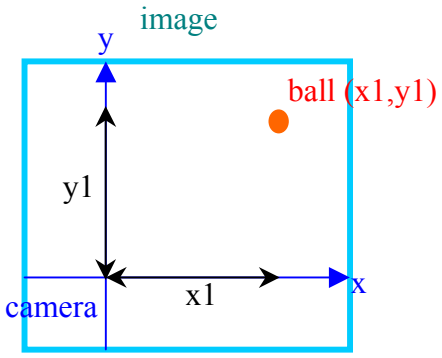
Jargons and parameters

- vision->vob[vobBall].**cf** (line 10) is the confidence factor of the visual object, ball, in the robot's vision module. It is a value from 0 to 1000 (where 0 = not confident and 1000 = very confident), and in this case, if the confidence factor is greater than 0, then the robot can see the ball.
- **Pan**ning is the act of rotating the head in the XZ plane (ie. looking left and right).
- **Heading** is the angle between an object and the z-axis in the XZ plane, where left of the axis is positive and right is negative.
- **panx** (line 12) is the parameter a programmer can control to change the heading (in degrees) of the robot's head, relative to the top of the neck of the robot (aka back of the head, pan pivot point), ie. to pan the head. The robot may not end up with the exact heading of panx because of calibration and offsets, but it will be close. **hPan** (lines 17-19, 33, 35) is the exact angle (in degrees) of the robot's current heading, but it can only be read and not written to.
- vBall.**imgHead** (line 24) is the heading of the centre of the ball relative to the camera.
- **Tilt**ing is the act of rotating the head in the YZ plane (ie. looking up and down)
- **Elevation** is the angle between an object and the z-axis in the YZ plane, where up is positive and down is negative.
- **tilty** (line 50) is the parameter a programmer can control to change the elevation (in degrees) of the robot's head, relative to the base of the neck of the robot, ie. to tilt the head. As with panx, tilty is not exact, and **hTilt** (lines 38-39) is the exact current elevation of the robot's head.
- vBall.**imgElev** (line 25) is the elevation of the centre of the ball relative to the camera.
- vision->vob[0].**misc** (line 25) is the elevation of the top of the ball relative to the camera.
- vBall.**d** is the distance between the camera and the ball which is parallel to the field.
- **turnCCW** is the parameter a programmer can control to make the robot turn counter clockwise turnCCW degrees per half-step.

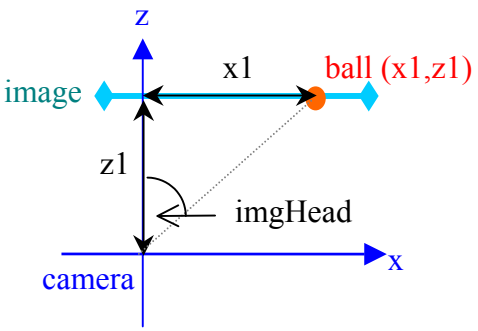
Coordinate system relative to the camera

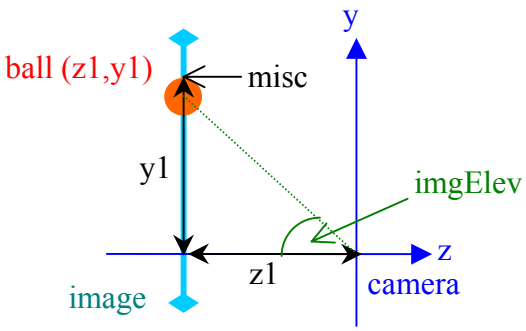
Let XYZ1 be the coordinate system where:

x-axis is perpendicular to the robot's field of vision
y-axis is the up vector
z-axis is parallel to the robot's field of vision (ie. the direction along the nose)
origin is the camera



Calculate the position (ie. the x,y,z coordinates) of the ball relative to the camera.





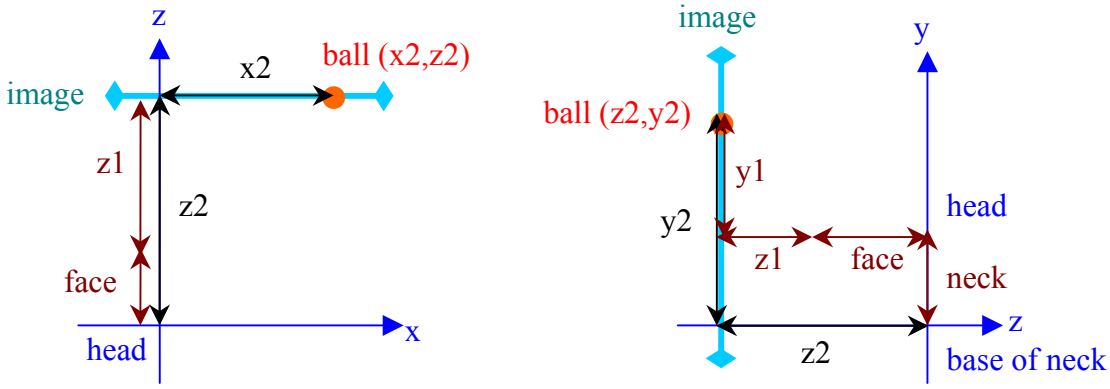

```
24 x1 = vBall.d * tan(radians(vBall.imgHead));
25 //y1 = vBall.d * tan(radians(vBall.imgElev));
26 y1 = vBall.d * tan (radians (PointToElevation (vision->vob[0].misc)));
27 z1 = vBall.d;
```

Coordinate system relative to the neck of the robot

Let XYZ2 be the coordinate system where:

axes are the same as above
origin is the base of the neck

Calculate the position of the ball relative to the base of the neck of the robot.




```
29 x2 = x1;
30 y2 = y1 + NECK_LENGTH;
31 z2 = z1 + FACE_LENGTH;
```

Account for original pan and tilt

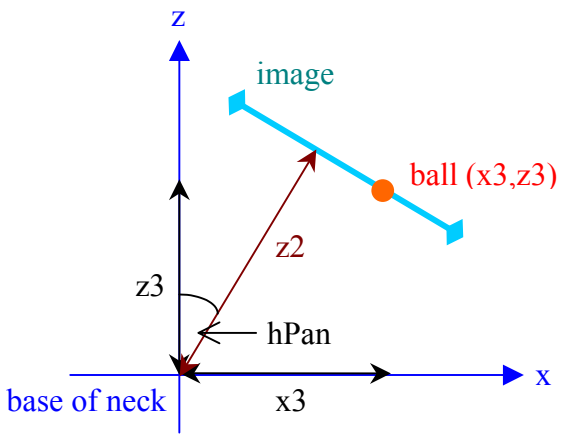
Let XYZ3 be the coordinate system where:

z-axis is the direction the robot's body is facing

origin is the base of the neck

Negate hPan because it is opposite to mathematical conventions (hence rotate by –hPan degrees instead)

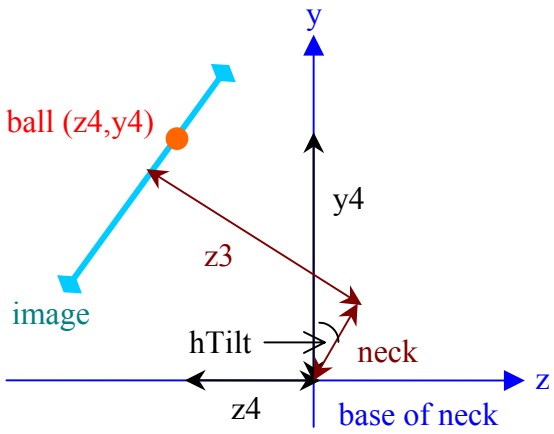
Rotate the position of the ball by -hPan degrees around the y-axis



$$\begin{pmatrix} x3 \\ y3 \\ z3 \end{pmatrix} = \begin{pmatrix} \cos(-hPan) & 0 & \sin(-hPan) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-hPan) & 0 & \cos(-hPan) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x2 \\ y2 \\ z2 \end{pmatrix} = \begin{pmatrix} x2 \times \cos(-hPan) - z2 \times \sin(-hPan) \\ y2 \\ x2 \times \sin(-hPan) + z2 \times \cos(-hPan) \end{pmatrix}$$

```
33 x3 = x2*cos(radians(-hPan))-z2*sin(radians(-hPan));
34 y3 = y2;
35 z3 = x2*sin(radians(-hPan))+z2*cos(radians(-hPan));
```

Rotate the position of the ball by hTilt degrees around the x-axis




```
37 x4 = x3;
38 y4 = z3*sin(radians(hTilt))+y3*cos(radians(hTilt));
39 z4 = z3*cos(radians(hTilt))-y3*sin(radians(hTilt));
```

Account for turn

Rotate the position of the ball by turnCCW/8 degrees around the y-axis in case the robot is turning

$$\begin{pmatrix} x4 \\ y4 \\ z4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(hTilt) & -\sin(hTilt) & 0 \\ 0 & \sin(hTilt) & \cos(hTilt) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x3 \\ y3 \\ z3 \end{pmatrix} = \begin{pmatrix} x3 \\ z3 \times \sin(hTilt) + y3 \times \cos(hTilt) \\ z3 \times \cos(hTilt) - y3 \times \sin(hTilt) \end{pmatrix}$$

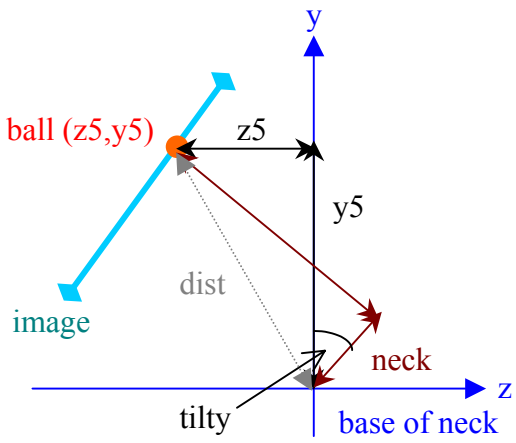
```
41 double turn = radians(turnCCW/8);
42 x5 = x4*cos(turn) - z4*sin(turn);
43 y5 = y4;
44 z5 = x4*sin(turn) + z4*cos(turn);
```

Calculate elevation of ball

Clip the z value in case it is shorter than the neck length.

```
46 if (z5 < NECK_LENGTH) z5 = NECK_LENGTH;
```

Calculate the elevation of the ball from its position relative to the base of the neck of the robot.



$$
\begin{aligned}
tilty &= \arctan\left(\frac{y5}{z5}\right) - \arcsin\left(\frac{neck}{dist}\right) \qquad \text{(For proof see Appendix B)} \\
&= \arccos\left(\frac{neck}{dist}\right) - \arcsin\left(\frac{z5}{dist}\right) \\
&= \arccos\left(\frac{neck}{dist}\right) - \arccos\left(\frac{y5}{dist}\right)
\end{aligned}
$$

```
48 headtype = ABS_H;
49 double dist = sqrt(z5*z5 + y5*y5);
50 tilty = degrees(atan(y5/z5)-asin(NECK_LENGTH/dist));
```

Future Improvements

Removals

```
11 lastTrackHead = vision->vob[vobBall].head;
```
This is the only place where `lastTrackHead` is used.

```
13 VisualObject &ball = vision->vob[vobBall];
```
`ball` is not referred to again.

```
17 if (abs(panx - hPan) > FOV) {
18 double factor = 1.0 + ABS(panx-hPan)/60;
19 panx = hPan+(panx - hPan)*factor;
20 }
```
When the ball rolls by too fast, the robot tends to lose sight of the ball instead of turning its head too fast. Therefore the differece between `panx` & `hPan` is rarely (if ever) greater than 10 when it *can* see the ball.

```
46 if (z5 < NECK_LENGTH) z5 = NECK_LENGTH;
```
Sanity checks are done elsewhere to prevent this from doing any harm.

```
51 old_tilt = tilty;
52 old_pan = panx;
53 } else {
54 tilty = old_tilt;
55 panx = old_pan;
```
These lines do absolutely nothing since `panx` and `tilty` are not reset each time saBallTracking is called.

Modifications

```
41 double turn = radians(turnCCW/8);
```
Currently this causes the robot to lose sight of the ball when turning at a large angle. Should change turnCCW/8 to a more accurate estimation of angle turned per camera frame (or angle turned per half step / number of camera frames per half-step).

Other Comments

```
25 //y1 = vBall.d * tan(radians(vBall.imgElev));
```
Tracks the centre of the ball, can cause the robot to jerk its head when the ball is too close to its face (fireball).

If one wants to calculate `panx` using the same way `tilty` is calculated, just add:
```
36 panx = atan(x3/z3);
```
Note: `panx` should be calculated relative to the head pivot point and not the base of the neck, hence use z3 instead of z4.

Conclusion

saBallTracking tracks the ball by converting the pan and tilt angles relative to each image the camera takes into and tilt angles relative to the base of the neck. First it calculates the position of the ball relative to the camera, then the base of the neck, then take in account of the pan and tilt angles it already faces, and finally converts this position back into angles.

Bibliography

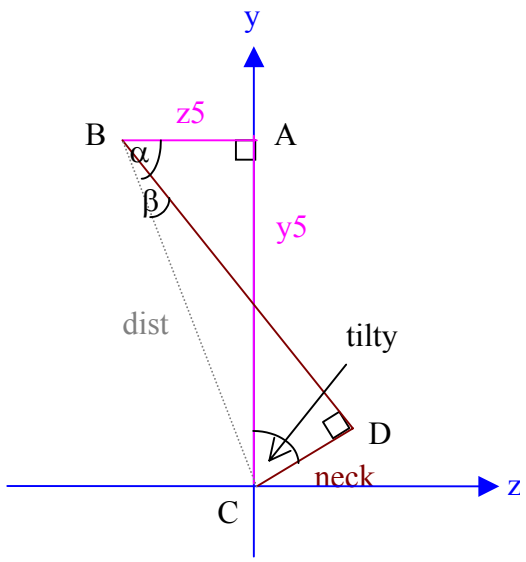
Z. Wang, J. Wong, et al. Ball Tracking, *rUNSWift RoboCup2002 Sony Legged League Team Thesis, UNSW,* 2002

Appendix A – ballTracking.cc

```
01 #include "Behaviours.h"
02 #define FOV 10
03
04 // HACK
05 static double old_tilt = 0;
06 static double old_pan = 0;
07
08 void Behaviours::saBallTracking(bool trackWM) {
09
10 if(vision->vob[vobBall].cf > 0) {
11 lastTrackHead = vision->vob[vobBall].head;
12 panx = vision->vob[vobBall].head;
13 VisualObject &ball = vision->vob[vobBall];
14 //cout << "ball d:" << ball.d << " dist:" << ball.dist << " camdist:" <<
ball.cam_dist << " h:" << ball.h << endl;
15
16
17 if (abs(panx - hPan) > FOV) {
18 double factor = 1.0 + ABS(panx-hPan)/60;
19 panx = hPan+(panx - hPan)*factor;
20 }
21
22 double x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,x6,z6;
23
24 x1 = vBall.d * tan(radians(vBall.imgHead));
25 //y1 = vBall.d * tan(radians(vBall.imgElev));
26 y1 = vBall.d * tan (radians (PointToElevation (vision->vob[0].misc)));
27 z1 = vBall.d;
28
29 x2 = x1;
30 y2 = y1 + NECK_LENGTH;
31 z2 = z1 + FACE_LENGTH;
32
33 x3 = x2*cos(radians(-hPan))-z2*sin(radians(-hPan));
34 y3 = y2;
35 z3 = x2*sin(radians(-hPan))+z2*cos(radians(-hPan));
36
37 x4 = x3;
38 y4 = z3*sin(radians(hTilt))+y3*cos(radians(hTilt));
39 z4 = z3*cos(radians(hTilt))-y3*sin(radians(hTilt));
40
41 double turn = radians(turnCCW/8);
42 x5 = x4*cos(turn) - z4*sin(turn);
43 y5 = y4;
44 z5 = x4*sin(turn) + z4*cos(turn);
45
46 if (z5 < NECK_LENGTH) z5 = NECK_LENGTH;
47
48 headtype = ABS_H;
49 double dist = sqrt(z5*z5 + y5*y5);
50 tilty = degrees(atan(y5/z5)-asin(NECK_LENGTH/dist));
51 old_tilt = tilty;
52 old_pan = panx;
53 } else {
54 tilty = old_tilt;
55 panx = old_pan;
56 }
```

Appendix B – Proof for tilty

Proof 1



Let neck = CD, $tilty = \angle ACD$

y5 = AC, $\alpha = \angle ABC$

z5 = AB, $\beta = \angle CBD$

dist = BC

$$\tan\alpha = \frac{y5}{z5}$$

$$\alpha = \arctan\left(\frac{y5}{z5}\right)$$

$$\sin\beta = \frac{neck}{dist}$$

$$\beta = \arcsin\left(\frac{neck}{dist}\right)$$

$$\angle BAC = \angle BDC = 90°$$

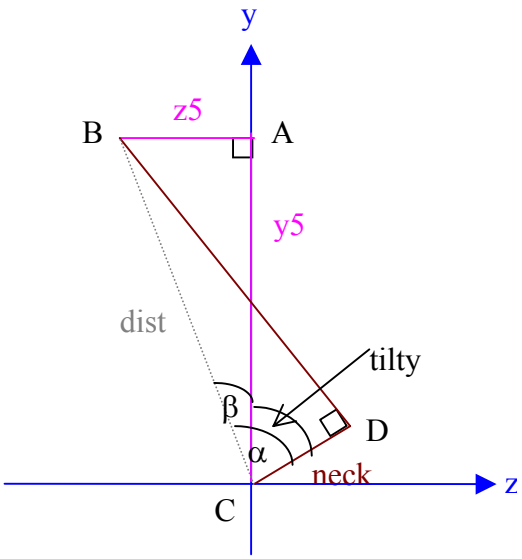
$\therefore ABCD$ forms a circle ($\angle BAC$ & $\angle BDC$ are angles on the circumference

$tilty = \angle ABD$ (angles on the circumference of circle $ABCD$

$$= \alpha - \beta$$

$$= \arctan\left(\frac{y5}{z5}\right) - \arcsin\left(\frac{neck}{dist}\right)$$

Proof 2



Let

neck	=	CD		*tilty*	$=\angle ACD$
y5	=	AC		α	$=\angle BCD$
z5	=	AB		β	$=\angle ACB$
dist	=	BC			

$$\cos\alpha = \frac{neck}{dist}$$

$$\alpha = \arccos\left(\frac{neck}{dist}\right)$$

$$\sin\beta = \frac{z5}{dist}$$

$$\beta = \arcsin\left(\frac{z5}{dist}\right)$$

$$\cos\beta = \frac{y5}{dist}$$

$$\beta = \arccos\left(\frac{y5}{dist}\right)$$

$$tilty \equiv \alpha - \beta$$

$$= \arccos\left(\frac{neck}{dist}\right) - \arcsin\left(\frac{z5}{dist}\right)$$

$$= \arccos\left(\frac{neck}{dist}\right) - \arccos\left(\frac{y5}{dist}\right)$$