SAPIENZA
UNIVERSITÀ DI ROMA

Dipartimento di Informatica e Sistemistica Antonio Ruberti

Master in Artificial Intelligence and Robotics

Report for the project in Robot Programming

Look at landmark: A strategy for switching the camera of Nao Robot

STUDENT:
Giovanni Murru

PROFESSOR:
Prof. Daniele Nardi

REFERENT:
Ricardo Dodds

SUMMER 2011

Contents

Abstract

Nao robot has two identical video cameras, one located between the eyes and another in the mouth position, strategically positioned in order to increase the field of view. Both of them can be used to identify objects such as goals and balls in the visual field, but bottom camera is necessary to see objects when they are near the foot of the robot, for example, for ball control.

The aim of this project is to implement a strategy that determines which camera to use, according to the distance of the landmark. The students may use both the Nao robot and the Webots simulator in order to test the algorithm. The development should be done mainly in the image processing module, which must be connected with the image server module of NaoQi. The basic idea is to take the landmark position (the position of the ball for example) given by the image processor and set the camera id to keep track of the object.

1 Introduction

Perception leads us to environment's awareness and understanding, by organizing and interpreting sensory information. Vision, which is critically important in human life, is used in robotics field to achieve intelligent actions such as identification and tracking of the objects.

Games, which have always covered a primary role in artificial intelligence research, often result in major findings. The world's most popular game is a very interesting and challenging subject for robotics and computer scientists.

In fact an international competition such as RoboCup, founded in 1997 with the goal to develop autonomous soccer robots, aims to promote research and education in the field of artificial intelligence. Aldebaran Robotics, one of the two Global Sponsors of this competition, is the creator of the Nao Robot, which was used in this project.

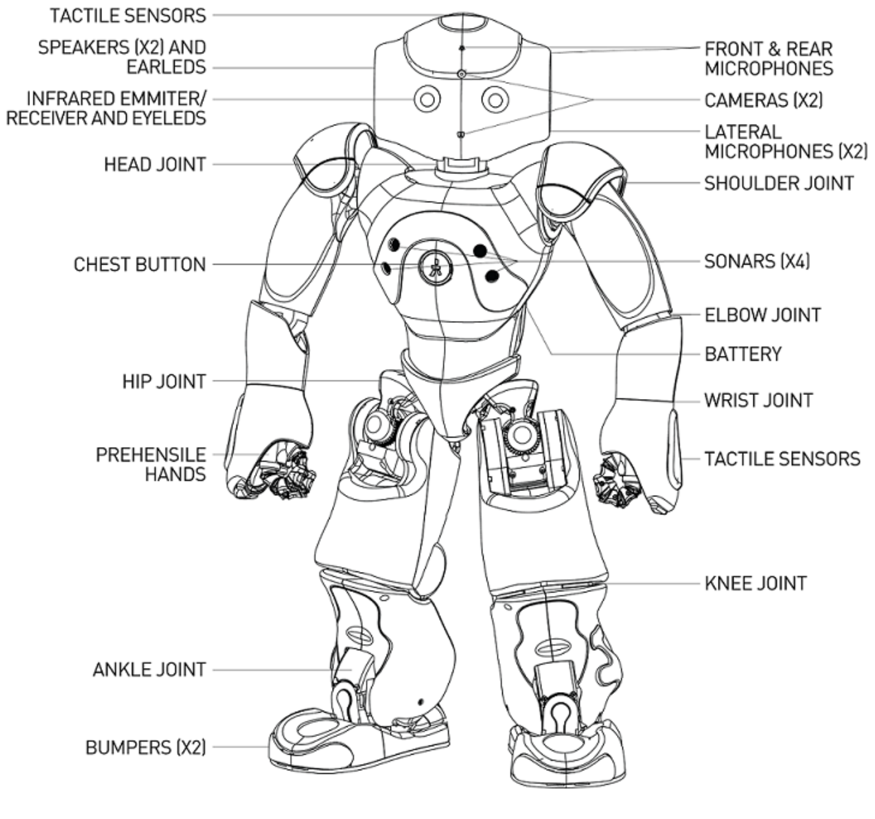

Figure 1.1: The Nao

1.1 The Nao Robot

Nao is a 5kg humanoid robot 58 cm tall with 13 degrees of freedom. Just two of them are located in the head: the pitch and the yaw.

Two identical cameras, with a standard VGA resolution, going at 30 fps, are mounted in Nao's head. The focus range is 30 cm to infinity. The vision field is 58 degrees on the diagonal and 34.80 degrees on the vertical.

Pitch and yaw are limited as shown in *Figure 1.2*. These constraints are important and should be taken into account during the definition of a possible strategy for the switch between the two cameras.

Two cameras are mounted on Nao's head, one in the forehead and the other in place of the mouth. The first one can be identified as the *Top Camera*, the last one as the *Bottom Camera*.

Besides the joints limitation another aspect considered during the elaboration of the switching strategy is the angular displacement between the two cameras. As a matter of fact the knowledge of this angle is essential for the computation of the camera matrix associated to each camera, which allows us to compute a correct information about object's z-depth in the world (e.g. the ball distance).

The radial displacement was also considered for aligning the head with the ball position after a switch made during ball tracking. In truth during the tracking of the ball, the switch between *Top Camera* and *Bottom Camera* is programmed to cause a 40 degrees upward rotation of head's pitch, while a downward rotation of the same angle is performed during the switch between *Bottom Camera* and *Top Camera*.

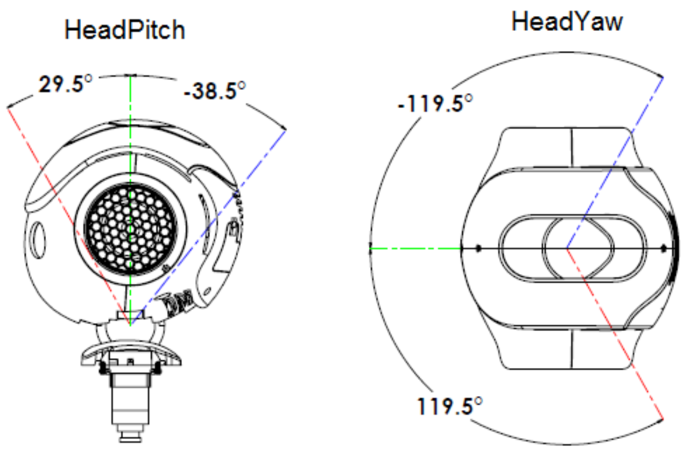

Figure 1.2: Pitch and yaw in Nao's head

2 Development

2.1 A brief introduction to the Open RDK framework

Open RDK is an open source modular software framework focused on rapid development of robotic applications. This framework with the additional modular extensions specifically developed for the Nao Robocup have been used to program the behavior of Nao robot.

The key point of Open RDK is its modularity. Typically a module creates its own properties during the initialization and it can read from or write to them only. The properties are stored in a shared memory and can be connected through a repository to other properties that belong to different modules. In this way Open RDK modules are able to share data such as sensor information.

Properties, which can be of input or output, are usually connected through a linking system, so that a write in a specific module's property is immediately seen by a read on the connected property.

Linking system is done through a XML file, which is generated using the **ragent2** console.

2.2 The modules and libraries involved

The main modules and libraries involved in the development are listed below with a brief description.

RSensorsServer: It's the main library where we take low level decision like the selection of the camera identifier. *RSensorsServer*, which is compiled and built in the main library *libopenrdk.so*, directly calls the NAOqi framework, a basic set of functions such as the one used in the code below to change the camera parameter kCameraSelectedID.

```
try
{
fCamProxy->callVoid( "setParam", AL::kCameraSelectID, ...
                              ... imageData.selectedCameraID );
currentCamera = imageData.selectedCameraID;
}
catch (const AL::ALError& e){...}
```

NaoVisionUtils: It's a set of utilities for the Nao vision system incorporated in the perception libraries. One of the functionalities of NaoVisionUtils is the computation of camera matrix. We modified the class in order to compute the correct camera matrix for the top camera.

```
static const double xHeadTiltToCamera      = 0.0488;       //m
static const double zHeadTiltToCamera      = 0.02381;      //m
static const double headTiltToCameraTilt = 0.698132;     //rad

// added to manage the Top CameraMatrix
static const double xHeadTiltToTopCamera = 0.0539;       //m
static const double zHeadTiltToTopCamera = 0.0679;       //m
static const double headTiltToTopCameraTilt = 0.0;       //rad
.
.
.
if(topCamera)// TOP CAMERA is the current selected camera
{
        theRobotCameraMatrix.translate(xHeadTiltToTopCamera,
                                       0., zHeadTiltToTopCamera);

        theRobotCameraMatrix.rotateY(headTiltToTopCameraTilt
                                        + cameraTiltCorrection);
}
else
{
        theRobotCameraMatrix.translate(xHeadTiltToCamera,
                                        0., zHeadTiltToCamera);

        theRobotCameraMatrix.rotateY(headTiltToCameraTilt
                                        + cameraTiltCorrection);
}
```

We have done this considering the known measures shown in *Figure 2.1* and in *Table 2.1*. In particular the **x** head tilt and **z** head tilt for the top camera are respectively 53.9 cm and 67.9 cm, while as regard the bottom one the tilts are 48.8 cm in the **x** and 23.81 cm in the **z**. The tilt on the **y** is null for both the cameras, since they are assumed to be centered in the head. Adopting the roll, pitch, yaw notation for the angle representation, we see that while the bottom camera is placed at 40 degrees of pitch, the top one is at 0. This is also important for the computation of the camera matrix.

Table 2.1: Informations about position and orientation of the two cameras

Camera name	**X** [m]	**Y** [m]	**Z** [m]	**WX** [deg]	**WY** [deg]	**WZ** [deg]
Top Camera	0.0539	0.0	0.0679	0.0	0.0	0.0
Bottom Camera	0.0488	0.0	0.02381	0.0	40.0	0.0

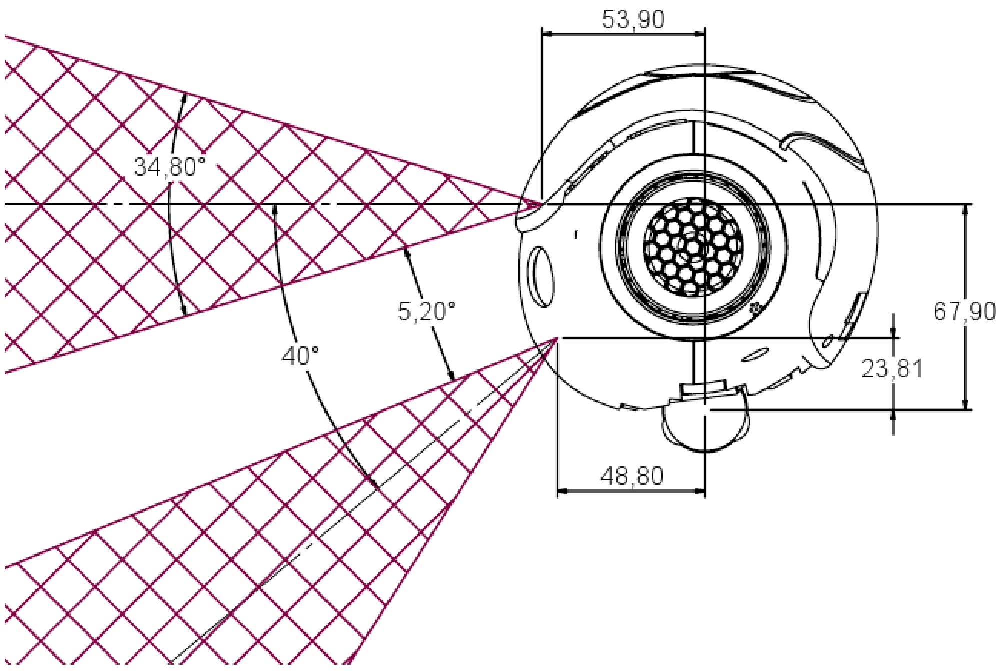

Figure 2.1: Cameras' position and radial displacement

PerceptionModule: Perception module is a module that includes a set of functions to manage information coming from the environment, such as the data processed by the two cameras.

Perception includes two submodules:

- AllSensors
- ImageProcessor

AllSensors is demanded to analyze and explicate all the sensors data, including the vision flow. AllSensors is linked to the ImageProcessor, whose goal is to process the image data and generate useful information such as ball position or distance. The ImageProcessor manages also the ball recognition process.

PnpExecutorModule This module is responsible for applying PNP strategies such as the tracking of the ball or its search.

2.3 The SwitchCamera module

The switching of the camera is a complex action that has to take into account many factors. In this project we had to decide a strategy for the switching based on the ball distance from the robot. The computation of the ball distance is delegated to BallSpecialist located in the imageProcessingLibrary, which belongs to the perception libraries.

Hence the SwitchCamera module is linked to the PerceptionModule to retrieve informations as the ball's distance from the robot. Ball distance is chosen to be

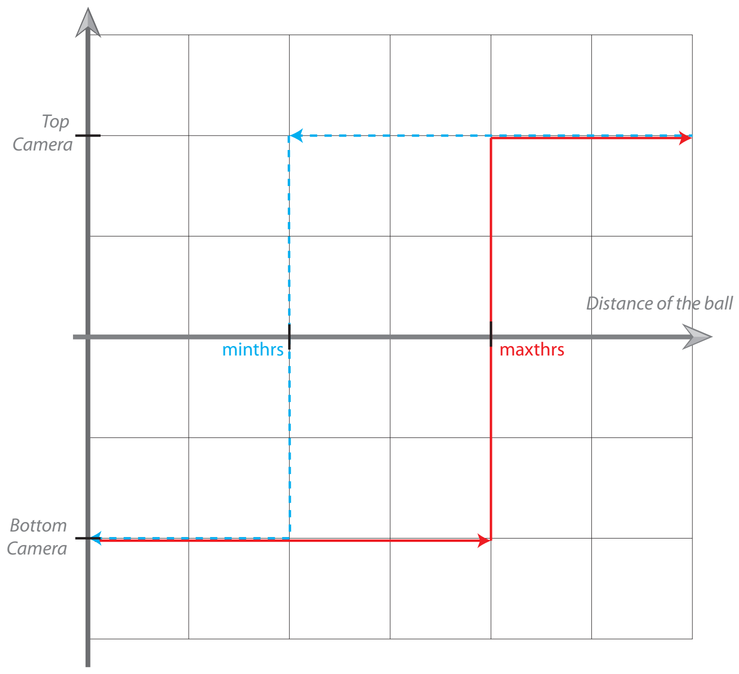

Figure 2.2: Isteresi method

the main criteria for the switch. In particular 2 thresholds are used for deciding the correct moment for performing the switch.

maxthrs: The maximum threshold is a critical distance limit used during the switch from *Bottom Camera* to *Top Camera*.

minthrs: The minimum threshold is a critical distance limit used during the switch from *Top Camera* to *Bottom Camera*.

When the *bottom camera* is selected we want to decide when to switch to the *top camera*. The idea is to perform this when the ball is far enough (i.e. the distance of the ball overcomes **maxthrs**) and some other constraints related to the joint movements are satisfied.

On the contrary when the *top camera* is selected the switch to *bottom camera* is done using a different threshold for ball distance. As a matter of fact it is performed whenever the ball distance goes below **minthrs**.

The choice of an hysteresis method as the one illustrated in *Figure 2.2* is necessary to avoid a continuous switching between the two cameras.

During the ball tracking the head is moved in order to keep the ball at the center of the current acquired frame. Once the camera is switched the robot's head has to be moved in order to re-center the ball with the frame. As means to find the ball after the switch, the minimum required pitch rotation is equal to the 40 degrees angle displacement between top and bottom camera.

2.4 Linking

In order to perform the switch, the necessary head motions and the ball perception, SwitchCamera module is linked with three other modules:

- Perception
- Executor
- Motion

```
#cmds/headCommand=
#cmds/switchCamera=
...
...
cmds/doColorTable=@/perception/ip/cmds/doColorTable
cmds/forceReloadCameraParams=@/perception/as/cmds/forceReload
cmds/headCommandFromPNP=@/executor/cmds/headCommand
cmds/reloadParams=@/perception/as/cmds/reloadCameraParams
enabled=Yes
in/ballCoordinateInImage=@/perception/ip/out/ballInImage
in/ballPercept=@/perception/ip/out/ballPercept
in/ballSeen=@/perception/ip/out/ballSeen
in/cameraSelect=@/perception/as/params/cameraSelect
params/maxThreshold=0.90
params/minThreshold=0.80
```

In the listing we shown a piece of the XML configuration file used to run the agent on the Nao. As we can notice there are various links. Let's briefly explain them.

cmds/doColorTable is linked to the image processor contained in the perception module. It is used to do the color table during the startup. This is used to automatically do the color table during the startup of the agent.

cmds/forceReloadCameraParams is linked to allsensors, contained in the perception module. Its aim is to force the reloading of the camera parameters in case ball is lost. In practice whenever the robot change its status from ball-seen to not-ball-seen the command is executed one time.

cmds/reloadParams is linked to allsensors which is built into perception module. In order to reload the camera parameters this command is used after the selection of the camera id during the switch of the camera.

in/cameraSelect is the parameter referred to the ID of the current selected camera. It is linked to allsensors in the perception module. It is used during the switch of the camera.

cmds/switchCamera is a command internal to the module, which serves to switch the camera manually (e.g. if the user want to manually switch between cameras).

cmds/headCommandFromPNP is linked to the head commands coming from PNP executor and is used to capture these commands and redirect them to the Motion module. In practice this is necessary because during the switch between the cameras, SwitchCamera module needs to move the head to re-center the ball with the line-of-sight. In order to avoid unexpected behavior caused by possible simultaneous head commands coming from PNP and SwitchCamera modules, the policy of head control is delegated to the SwitchCamera module, which decides whenever is appropriate to pass control to PNP or not.

cmds/headCommand is linked to Motion module. In this property the module writes the commands necessary to move the head, that can be generated by the module itself or the pnp executor.

in/ballCoordinateInImage is linked to the image processor built into the perception module. It's a input property since we used it to retrieve the ball coordinates relative to the current acquired frame.

in/ballSeen is a boolean value linked to the image processor and indicates if the ball is seen or not. However this value may be affected by noise, since it is updated every frame. Hence we adopted a policy which considered the ball as detected only if it has been seen in the last second of time.

in/ballPercept is linked to the perception module to retrieve information about the actual ball's position in space, relative to Nao robot's feet. In practice it incorporates a RPolarPointCov data structure which contains rho and theta polar coordinates, that are generated from the image processor and refer respectively to distance and orientation of the ball, relative to a coordinate system attached at the basis of the robot. In other words:

- **rho** is the ball's distance from the robot's feet.
- **theta** is the angle of the ball from the direction of the feet.

params/maxThreshold is a maximum distance limit threshold used when the robot is using the *bottom camera*. If the limit is overcome, Nao switch its camera to the top one.

params/minThreshold is a minimum distance limit threshold used when the robot is using the *top camera*. If the limit is overcome, Nao switch its camera to the bottom one.

The execution of the camera switch is managed in a function called by an handler listening on the property **cmds/headCommandFromPNP**. In this way the function is called every time the property is modified from the pnp module. Some parameters are initialized as the module start.

```
maxPitch =    0.5149;
minPitch =  - 0.6720;

deltaBT = 0.6981;
bottomCamPitchThrs = maxPitch - deltaBT;
topCamPitchThrs = minPitch + deltaBT;
```

maxPitch is the maximum pitch allowed for Nao's head expressed in radiants.

minPitch is the minimum pitch allowed for Nao's head expressed in radiants.

bottomCamPitchThrs is the maximum pitch necessary to perform the switch when the bottom camera is selected. In fact if the pitch overtake this limit the switch during the tracking of the ball is not allowed, because this would probably result in loss of the ball.

topCamPitchThrs is on the contrary the minimum pitch necessary to perform the switch when the top camera is selected, for the same reasons explained above.

However the switch of the camera may be performed setting the property value of **cmds/switchCamera** to true, and in this particular case the thresholds are not taken into account.

2.5 The algorithm

The algorithm executed in the function can be summarized in 3 steps:

1. Evaluate if the ball is seen by the robot.
2. IF the ball is seen:
   - A. Get **rho**'s current value, which is the ball distance with respect to the robot's feet.
   - B. IF the selected camera is the **bottom** one AND **rho** is greater than **maxThreshold** AND the head's current pitch is sufficiently small (i.e. it is smaller than **bottomCamPitchThrs**), THEN switch to the **top camera** AND move Nao's head adding 40 degrees to the current pitch AND forget that ball is seen.
   - C. ELSE IF the selected camera is the **top** one AND **rho** is less than **minThreshold** AND the head's current pitch is sufficiently high (i.e. it is greater than **topCamPitchThrs**), THEN switch to the **bottom camera** AND move Nao's head subtracting 40 degrees to the current pitch AND forget ball is seen.
   - D. ELSE track the ball using the commands of the head coming from PNP executor.
3. ELSE (if the ball is not seen)
   - A. Only IF the state is just switched from ball-seen to ball-not-seen, try to reload camera parameters. This is done because sometime it happens that robot can't see the ball because of an incorrect initialization of the camera.
   - B. Search the ball using PNP executor plan.

3 Conclusion

We developed a strategy to determine which camera has to be use according to the distance of the ball. The module has been integrated with the agent typically used during the Robocup, implementing a communication with PNP executor, perception and motion modules.

The agent has been tested several times on Nao robot physically and it appears to work as expected. When the agent runs it starts to search the ball with the bottom camera. When the ball is found *rho* distance is retrieved and if the ball is too far the camera is switched to the top one, and since typically the ball is found immediately the agent continues to track it using the top camera. In case the ball is near the robot uses the bottom camera to keep track of the ball. Nonetheless tracking or searching of the ball is performed using the PNP plan. Whenever the distance reaches one of the critical thresholds the camera is switched.

Because of a need in controlling head's motion during the switch, the module should be stand alone or at most integrated in the pnp executor module.

References

[1] Russel, S. and Norvig, P. *Artificial Intelligence: A modern approach*, Third Edition, 2010

[2] Hartley, R.I. and Zisserman, A. *Multiple View Geometry in Computer Vision*, Second Edition, 2004

[3] Aldebaran Robotics, *NAOqi Red Documentation*

[4] Wikipedia, *http://www.wikipedia.org/*

[5] OpenRDK project, *http://openrdk.sourceforge.net/*

[6] LabRoCoCo wiki, *http://labrococo.dis.uniroma1.it/dokuwiki/doku.php*