Chapter 10

Intelligent Humanoid Robot

On successful completion of this course, students will be able to:

- Explain how the humanoid robot works.
- Develop vision-based humanoid robot.
- Explain object detection using keypoints and feature matching.

Introduction

Modern Humanoid Robot in uncontrolled environments needs to be based on vision and versatile. This paper propose a method for object measurement and ball tracking method using Kalman Filter for Humanoid Soccer, because the ability to accurately track a ball is one of the important features for processing high-definition image. A color-based object detection is used for detecting a ball while PID controller is used for controlling pan tilt camera system. We also modify the robot's controller CM-510 in order able to communicate efficiently using main controller.

Humanoid Robot

The humanoid robot is popular nowadays for the entertainment or contests such as RoboCup Humanoid League. The important features of humanoid soccer, such as accuracy, robustness, efficient determination and tracking of ball size and location; has proven to be a challenging subset of this task and the focus of much research. With the evolution of robotics hardware and subsequent advances in processor performance in recent years, the temporal and spatial complexity of feature extraction algorithms to solve this task has grown (Ha et al, 2011).

In the case of Humanoid soccer, vision systems are one of the main sources for environment interpretation. Many problems have to be solved before having a fully featured soccer player. First of all, the robot has to get information from the environment, mainly using the camera. It must detect the ball, goals, lines and the other robots. Having this information, the robot has to self-localize and decide the next action: move, kick, search another object, etc. The robot must perform all these tasks very fast in order to be reactive enough to be competitive in a soccer match. It makes no sense within this environment to have a good localization method if that takes several seconds to compute the robot position or to decide the next movement in few seconds based on the old perceptions (Martin et al, 2011). At the same time many other topics like human-machine interaction, robot cooperation and mission and behavior control give humanoid robot soccer a higher level of complexity like no any other robots (Blanes et al, 2011). So the high speed processor with efficient algorithms is needed for this issue.

One of the performance factors of a humanoid soccer is that it is highly dependent on its tracking ball and motion ability. The vision module collects information that will be the input for the reasoning module that involves the development of behaviour control. Complexity of humanoid soccer makes necessary playing with the development of complex behaviours, for example situations of coordination or differ rent role assignment during the match. There are many types of behaviour control, each with advantages and disadvantages: reactive control is the simplest way to make the robot play, but do not permit more elaborated strategies as explained for example in (Behnke, 2001). On the other side, behaviour-based control are more complex but more difficult to implement, and enables in general the possibility high-level behaviour control, useful for showing very good performances. Intelligent tracking algorithm for state estimation using Kalman filter has been successfully developed (Noh et al, 2007), and we want to implement that method for ball tracking for humanoid soccer robot.

We propose architecture of low cost humanoid soccer robot compared with the well known humanoid robots for education such as DarwIn-OP and NAO and test its ability for image processing to measure distance of the ball and track a ball using color-based object detection method, the robot will kick the ball after getting the nearest position between the robot and the ball. The Kalman filter is used here to estimate state variable of a ball that is excited by random disturbances and measurement noise. It has good results in practice due to optimality and structure and convenient form for online real time processing.

For future robotics, we will familiar with term of robot ethics. Robot ethics is a growing interdisciplinary research effort roughly in the intersection of applied ethics and robotics with the aim of understanding the ethical implications and consequences of robotic technology. Swarm robotics is a new approach to the coordination of multirobot systems which consist of large numbers of mostly simple physical robots. It is supposed that a desired collective behavior emerges from the interactions between the robots and interactions of robots with the environment. Swarm robotics systems are characterized by decentralized control, limited communication between robots, and use of local information and emergence of global behavior.

The Architecture of the Humanoid Robot

Humanoid soccer robots design based on the vision involves the need to obtain a mechanical structure with a human appearance, in order to operate into a human real world. Another important feature for modern humanoid robot is the ability to process tasks especially for computer vision. We propose an embedded system that able to handle high speed image processing, so we use main controller based on the ARM7 Processor. Webcam and servo controller are used to track a ball, and the output of the main controller will communicate with the CM510 controller to control the actuators and sensors of the robot as shown in figure. 10.1.

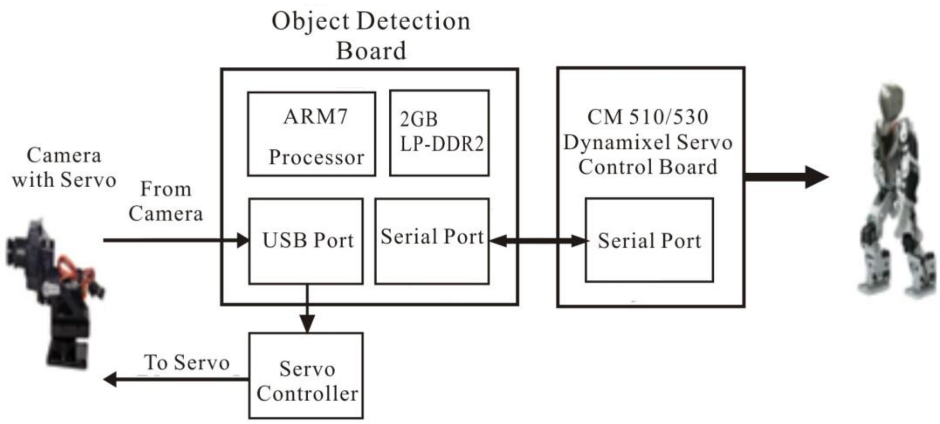

Figure 10.1 *Architecture of high speed system for Humanoid Soccer Robot.*

The main controller uses Odroid X2 that consist of Cortext-A9 1.7 GHz and sufficient memory and ports to be connected with other devices as shown in fig. 10.2. The specification of the Odroid X2:

- Exynos4412 Quad-core ARM Cortex-A9 1.7GHz.
- 2GByte Memory.
- 6 x High speed USB2.0 Host port.
- 10/100Mbps Ethernet with RJ-45 LAN Jack.

Figure 10.2 *Odroid X2 for processing the images from webcam (hardkernel.com, 2013).*

The Firmware of the robot to control the servos is modified from the original one named Robotis Firmware due to the limitation for sending a motion command by serial interface based on Peter Lanius works published in google code (Lanius, 2013). This firmware instead using RoboTask to program the robot controlling its movement but it directly program the AVR Microcontroller inside the CM-510 controller using C language. Using this alternative can reduce the size of the program from originally 170KB to 70KB in the memory. By this firmware, the robot can be connected directly to Ball Tracking System using USB Serial Interface to command its motion. Based on this framework, it opens an opportunity to built Real Time Operating System for the robot. The robot's control starts with initialization routines of CM-510 controller then move to Wait for Start Button state. In this state, it waits the button to be pressed to change the start_button_pressed variable from FALSE to TRUE then move to Dynamixel and Gyro Initialization which send broadcast ping to every Dynamixel servo connected to CM-510. When one or more servos do not respond of the ping then CM-510 will send a message mentioning the failure of a servo to serial terminal. Gyro Initialization does gyro calibration in the robot to get center reference and sends the value to serial terminal. Next state is Waiting Motion Command that waits the command through serial interface, from terminal or tracking module, then check if the command is valid or not. If it does not valid then the state will repeat to Wait Motion Command or continue to next Execute Motion Command state when the command is valid. Execute Motion Command executes a motion command to move a servos based on defined Look-Up-Table (LUT).

For example, when a command says WALKING then the state looks servo's values for WALKING stored in the LUT then sends it to Dynamixel servo through serial bus. When a motion is completed then it move to preceding state but if there is an emergency which is determined by pressing start button when the servos is moving compared to command input which does not receive stop command, then it move to Dynamixel Torque Disable to disable all the servo's torque to save from damage and move to Wait for Start Button state. The improved system to accept commands from the main controller is shown as the state machine in figure 10.3.

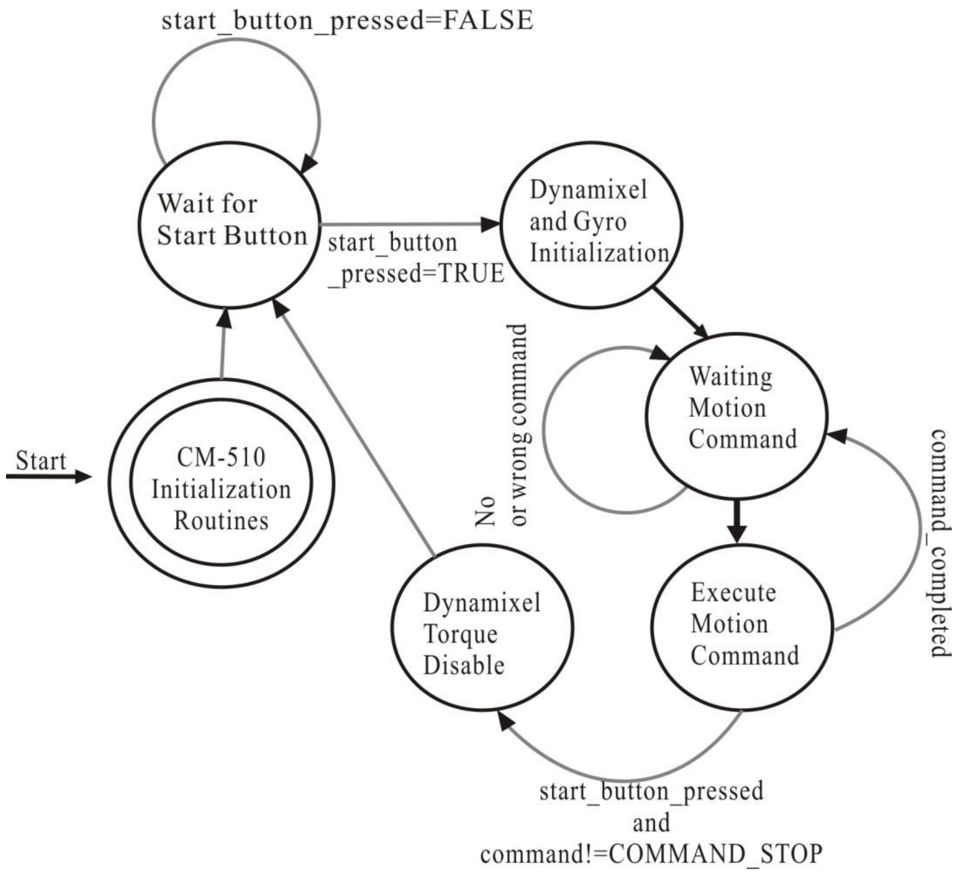
start_button_pressed=FALSE
Wait for Start Button
start_button _pressed=TRUE
Dynamixel and Gyro Initialization
No or wrong command
Waiting Motion Command
command_completed
Start
CM-510 Initialization Routines
Dynamixel Torque Disable
Execute Motion Command
start_button_pressed and command!=COMMAND_STOP

Figure 10.3 *State machine of the robot's controller.*

Ball Distance Estimation and Tracking Algorithm

Computer vision is one of the most challenging applications in sensor systems since the signal is complex from spatial and logical point of view. An active camera tracking system for humanoid robot soccer tracks an object of interest (ball) automatically with a pan-tilt camera. We use OpenCV for converting to HSV (Hue Saturation-Value), extract Hue & Saturation and create a mask matching only the selected range of hue value (Szeliski, 2010).

To have a good estimation, the object must be in the centre of the image, i.e. it must be tracked. Once there, the distance and orientation are calculated, according to the neck's origin position, the current neck's servomotors position and the position of the camera in respect to the origin resulting of the design (Maggi, 2007). We considered method for distance estimation of the ball by centering the ball on the camera image, using the head tilt angle to estimate the distance to the ball.

Region growing algorithms are also used to locate the ball color blobs that have been identified by region growing and are useful and robust source for further image processing, as demonstrated by (Ghanai, 2009). The ball will be tracked based on the color and webcam will track to adjust the position of the ball to the center of the screen based on the Algorithm 1.

Algorithm 1: Ball tracking and Kick the ball

```
Get input image from the camera
Convert to HSV (Hue-Saturation-Value)
Extract Hue & Saturation
Create a mask matching only for the selected range of hue
Create a mask matching only for the selected saturation levels.
Find the position (moment) of the selected regions.
If ball detected then
Estimate distance of the ball
Object tracking using Kalman Filter
 centering the position of the ball
  Move robot to the ball
 If ball at the nearest position with the robot then
  Kick the ball
endif
```

```
endif
```

The estimated position($\hat{x}$,$\hat{y}$) from Kalman filter is used as an input to PID controller. We use a PID controller to calculate an error value as the difference between a measured (input) and a desired set point to control high speed HS-85 servos. The controller attempts to minimize the error by adjusting (an Output). The model of PID Controller is shown in figure 10. 4:

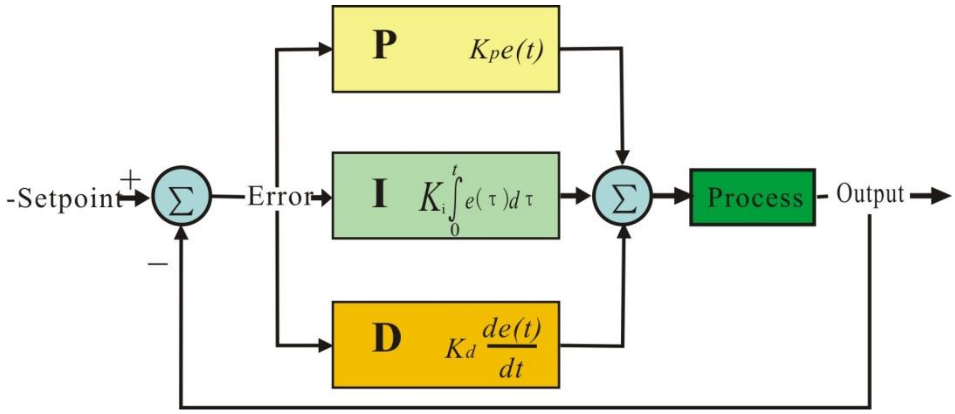

Figure 10.4 *General PID Controller.*

The output of a PID controller, equal to the control input to the system, in the time-domain is as follows:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de}{dt} \tag{10.1}$$

A Framework of Multiple Moving Obstacles Avoidance Strategy

Because we want a general model for humanoid service robot, we propose a framework for multiple moving obstacles avoidance strategy using stereo vision. A multiple moving obstacle avoidance strategy is an important framework to develop humanoid service robot in dynamic environment. There are two mains actors on multiple moving obstacles avoidance system; the Robot itself and the Range Finder. The Robot interacts with this system to detect customer and determine moving obstacles. Both processes to detect customer and determine moving obstacles include a process of face recognition as explained before. After the obstacles are determined, The Range Finder (camera and its system) will calculate and estimate the distance of those moving obstacles and estimate the direction of moving obstacles. Direction estimation of obstacles will be used to determine optimal maneuver of the Robot to avoid those obstacles. The framework is shown in figure 10.5:

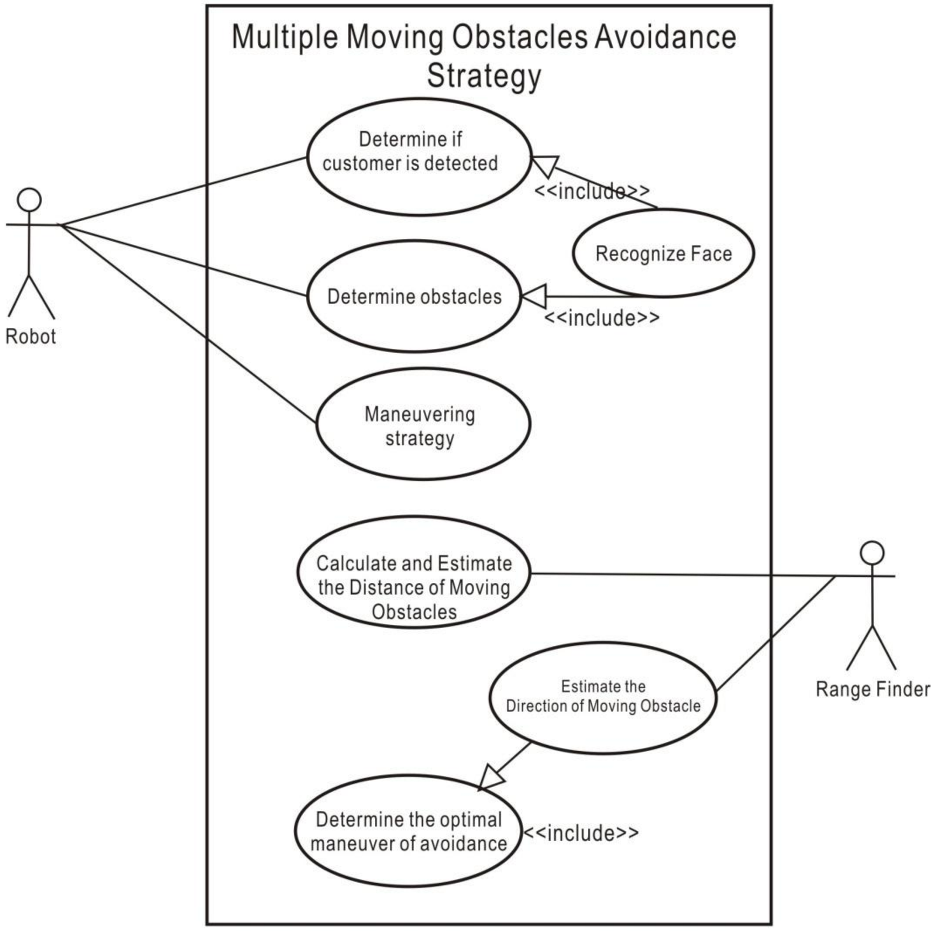
Multiple Moving Obstacles Avoidance Strategy
Determine if customer is detected
<<include>>
Recognize Face
Determine obstacles
<<include>>
Robot
Maneuvering strategy
Calculate and Estimate the Distance of Moving Obstacles
Estimate the Direction of Moving Obstacle
Range Finder
Determine the optimal maneuver of avoidance
<<include>>

Figure 10.5 *The use case diagram for our multiple moving obstacles avoidance strategy using stereo vision.*

Visual perception is the ability to interpret the information and surroundings from the effects of visible light reaching the eye. The resulting perception is also known as eyesight, sight, or vision. Visual-perception-based of service robot for customer identification is an interpretation process to direct a service robot to a destination of identified customer based on face recognition system and computer vision. After interpretation of images from camera done, then it is used as information for the robot and to decide actions based on the task given by a developer. The basic of visual–perception model for a humanoid service robot is shown in figure 10.6:

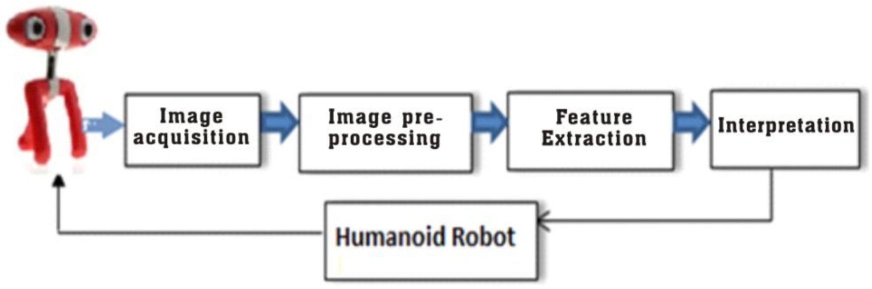

Figure 10.6 *Visual-perception model for vision-based humanoid robot. After interpretation process, the information used for navigating the robot or deciding actions for robot, such as direct a robot to customer's position.*

Experiments

Object detection and segmentation is the most important and challenging fundamental task of computer vision. It is a critical part in many applications such as image search, image auto-annotation and scene understanding. However it is still an open problem due to the complexity of object classes and images. The easiest way to detect and segment an object from an image is the color based methods. The colors in the object and the background should have a significant color difference in order to segment objects successfully using color based methods.

We need a webcam to try detecting a ball using this program demo. Create an application using Visual C++ 2010 Express edition and OpenCV. Configure the properties and write a program below:

ColorBased.cpp:

```
//Demo Program of Color-Based Detection for a Ball
//Copyright Dr. Widodo Budiharto 2014

#include "stdafx.h"
#include <cv.h>
#include <highgui.h>

// threshold image HSV
```

```
  IplImage* GetThresholdedImage(IplImage* imgHSV){
  IplImage*
imgThresh=cvCreateImage(cvGetSize(imgHSV),IPL_DEPTH_8U, 1);
   cvInRangeS(imgHSV, cvScalar(170,160,60), cvScalar(180,256,256),
imgThresh);
    return imgThresh;
  }

  int main(){
    CvCapture* capture =0;
    capture = cvCaptureFromCAM(0);
    //set width and height
  cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH, 640 );
  cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT, 480);

    if(!capture){
      printf("Capture failure\n");
      return -1;
    }

    IplImage* frame=0;
    cvNamedWindow("Video");
    cvNamedWindow("Ball");

    //iterasi frame
    while(true){
      frame = cvQueryFrame(capture);
      if(!frame) break;
      frame=cvCloneImage(frame);
   //smooth the original image using Gaussian kernel
   cvSmooth(frame, frame, CV_GAUSSIAN,3,3);
  IplImage* imgHSV = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U,
3);
  //ubah formwat color dari  BGR ke HSV
  cvCvtColor(frame, imgHSV, CV_BGR2HSV);
  IplImage* imgThresh = GetThresholdedImage(imgHSV);
   //smooth the binary image using Gaussian kernel
   cvSmooth(imgThresh, imgThresh, CV_GAUSSIAN,3,3);
      cvShowImage("Ball", imgThresh);
      cvShowImage("Video", frame);

      //bersihkan  images
      cvReleaseImage(&imgHSV);
      cvReleaseImage(&imgThresh);
      cvReleaseImage(&frame);

      //tunggu 50sec
      int c = cvWaitKey(10);
      //If 'ESC' is pressed, break the loop
      if((char)c==27 ) break;
    }
    cvDestroyAllWindows() ;
    cvReleaseCapture(&capture);
    return 0;
  }
```

The approach proposed in this paper was implemented and tested on a humanoid Robot named Humanoid Robot Soccer Ver 2.0 based on Bioloid Premium Robot. By modify the robot's controller (CM-510) in order to accept serial command from the main controller, this system able to communicate efficiently.

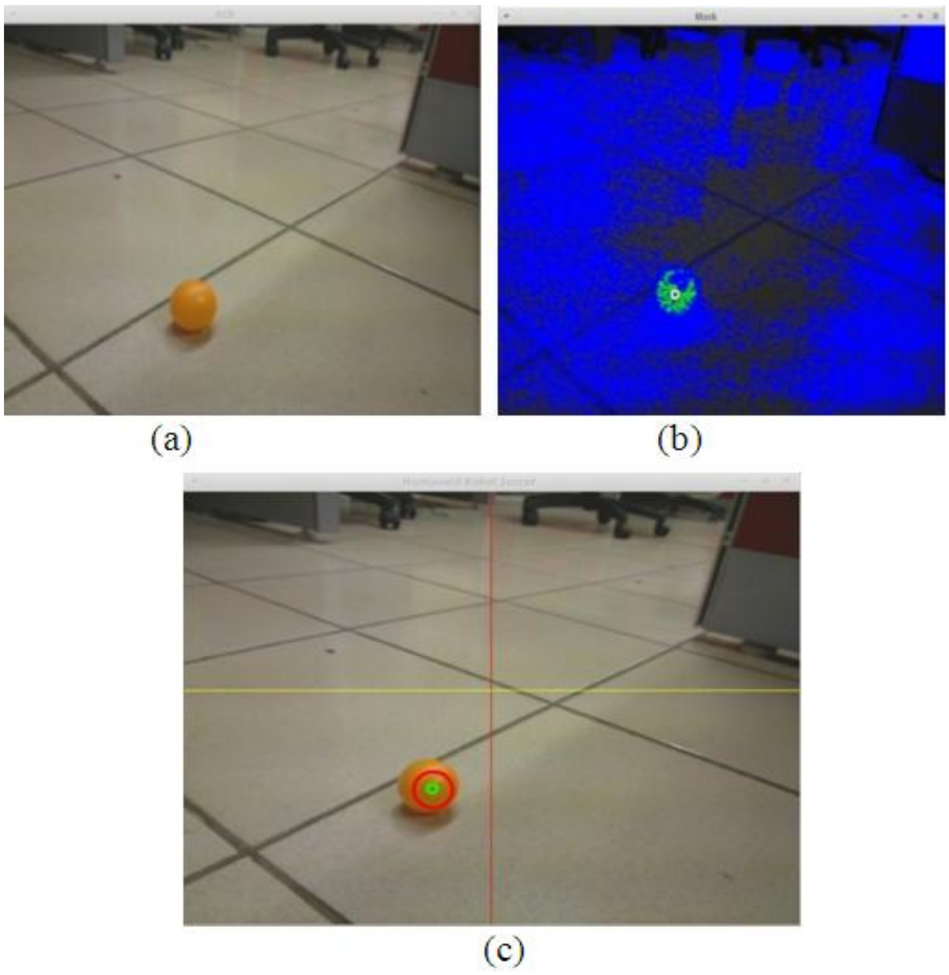

Figure 10.7 *The original image (a), the mask (a) and ball detected and tracked using Kalman Filters in the green circle (b).*

When a ball is in front of the robot and has been detected, the robot tries to track the ball, and if the ball at the nearest position with the robot, robot will kick it as shown in figure 10.8.

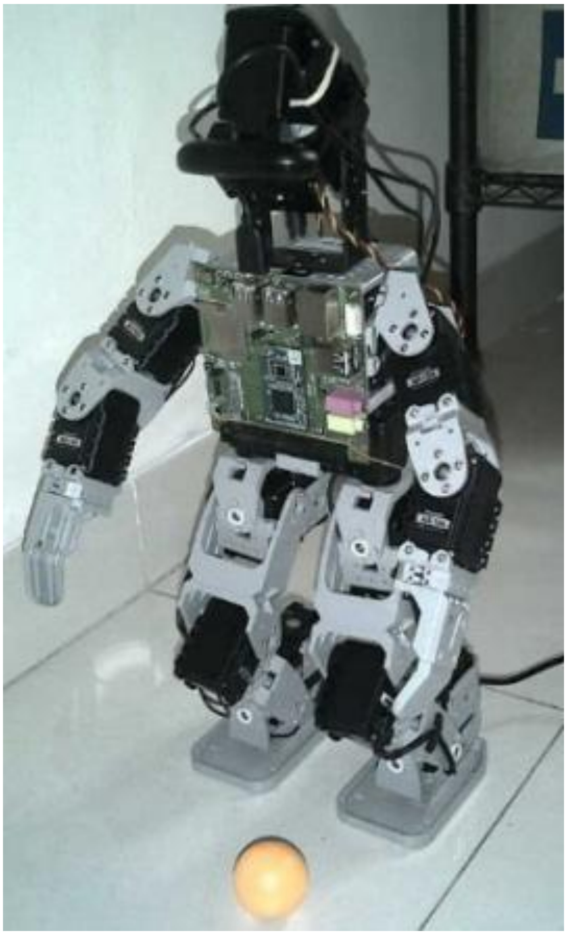

Figure 10.8 *The robot tracks and kicks a ball when at the correct position [13].*

Object Detection Using Keypoint and Feature Matching

The color-based object detector works well only for single-colored objects and can be fooled by different object of the same color, but color-based object detection is very fast. If you want to design the vision system for intelligent robot, you should obviously not rely on color for detecting object, because you don't know what the working environment of your robot will look like. So, we use Machine Learning and Object detection based on keypoints. In this method, the computer "learn" the characteristics of the whole object template and look for similar instances in other images. SIFT (Scale Invariant Feature Transform) is a famous algorithm for keypoint extraction and description) keypoints, and the matching descriptors between the two images.

Keypoint descriptors are also often called features. Object detection using SIFT is scale and rotation invariant. The algorithm will detect object that have the same appearance but a bigger or smaller size in the test image compared with the training images. Use the FlannBasedMatcher interface in order to perform a quick and efficient matching by using the FLANN (Fast Approximate Nearest Neighbor Search Library). SURF is a class for extracting Speeded Up Robust Features from an image. In short, SURF adds a lot of features to improve the speed in every step. OpenCV provides SURF functionalities just like SIFT. You initiate a SURF object with some optional conditions like 64/128-dim descriptors, Upright/Normal SURF, etc.

The features are invariant to image scaling, translation, and rotation, and partially in-variant to illumination changes and affine or 3D projection. Features are efficiently detected through a staged filtering approach that identifies stable points in scale space [14]. The first stage of keypoint detection is to identify locations and scales assigned under differing views of the same object. Detecting locations that are invariant to scale change of the image can be accomplished by searching for stable features across all possible scales, using a continuous function of scale known as scale space. It has been shown by Koenderink and Lindeberg that under a variety of reasonable assumptions the only possible scale-space kernel is the Gaussian function. Therefore, the scale space of an image is defined as a function, L(x, y, σ), that is produced from the convolution of a variable-scale Gaussian, G(x, y, σ), with an input image, I(x, y):

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)\,. \tag{10.2}$$

Where * is the convolution operation in *x* and *y*, and:

$$G(x,y,\sigma) = (1/2\pi\sigma^2)e^{-(x^2+y^2)/2\sigma^2}\,. \tag{10.3}$$

To efficiently detect stable keypoint locations in scale space, we use scale-space extrema in the difference-of-Gaussian function convolved with the image:

$$\begin{aligned} D(x,y,\sigma) &= [G(x,y,k\sigma) - G(x,y,\sigma)] * I(x,y) \\ &= L(x,y,k\sigma) - L((x,y,\sigma)) \end{aligned}\,. \tag{10.4}$$

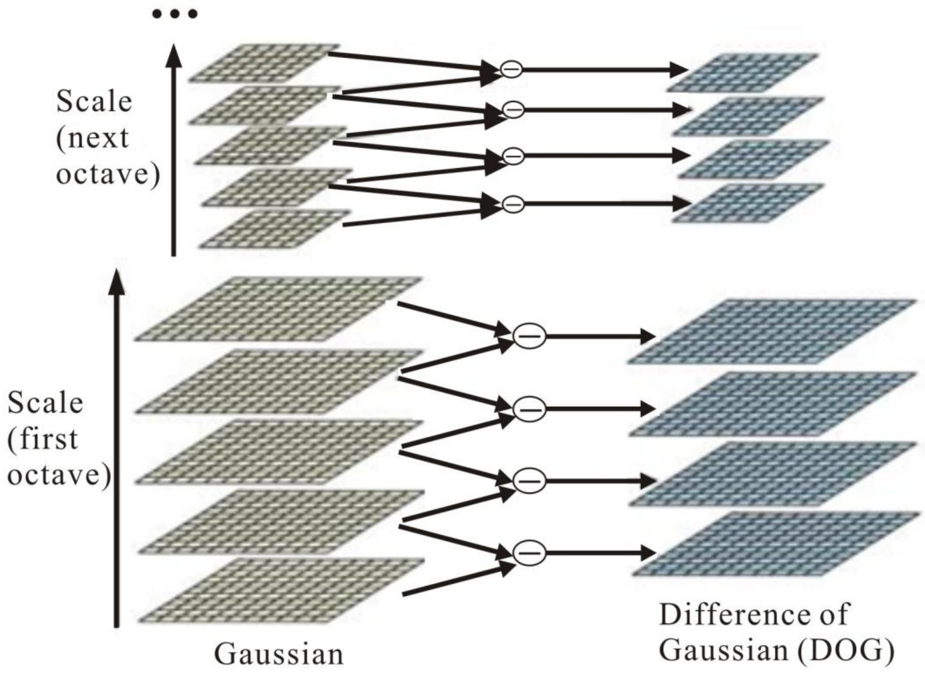

Figure 10.9 *Gaussian scale-space pyramid create an interval in the difference-of-Gaussian pyramid.*

Laplacian of Gaussian acts as a blob detector which detects blobs in various sizes due to change in σ. In short, σ acts as a scaling parameter. We can find the local maxima across the scale and space which gives us a list of (x,y,σ) values which means there is a potential keypoint at (x,y) at σ scale. But this LoG is a little costly, so SIFT algorithm uses Difference of Gaussians which is an approximation of LoG. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and kσ. This process is done for different octaves of the image in Gaussian Pyramid. It is represented in below image. Once this DoG are found, images are searched for local extrema over scale and space. In order to detect the local maxima and minima of G(x, y, σ), each sample point is compared to its eight neighbors in the current image and nine neighbors in the scale above and below:

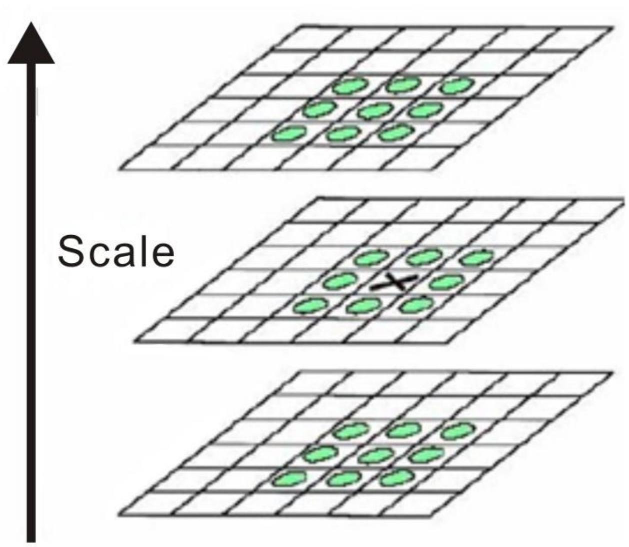

Figure 10.10 *Maxima and minima detection in the difference-of-Gaussian image.*

In 2004, D.Lowe, form University of British Columbia, [14] proposed how to extracting keypoints and computing descriptors using the Scale Invariant Feature Transform (SIFT). Keypoints are detected using scale-space extrema in difference-of-Gaussian function D and efficient to compute. Here is a sample program using a template file and webcam for SIFT keypoint detector using FLANN.

FLANN.cpp:

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
#include <stdio.h>
#include <cv.h>
#include <highgui.h>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/features2d.hpp"
using namespace std;
using namespace cv;
```

```
int main()
{
  int i=0;
  CvRect in_box,output_box;
  Mat train=imread("template3.jpg"), train_g;
  cvtColor(train,train_g,CV_BGR2GRAY);
  //detect SIFT keypoints
  vector<KeyPoint> train_kp;
  Mat train_desc;
  SiftFeatureDetector featureDetector;
  featureDetector.detect(train_g,train_kp);
  SiftDescriptorExtractor featureExtractor;
  featureExtractor.compute(train_g, train_kp, train_desc);
  //FLANN based descriptor matcher object
  FlannBasedMatcher matcher;
  vector<Mat> train_desc_collection (1,train_desc);
  matcher.add(train_desc_collection);
  matcher.train();
  //VideoCapture object
  VideoCapture cap(0);
  unsigned int frame_count=0;
  while (char(waitKey(1)) !='q') {
      double to=getTickCount();
      Mat test, test_g;
      cap>>test;
      if (test.empty())
          continue;
      cvtColor(test,test_g,CV_BGR2GRAY);
      //detect SIFT keypoint and extract descriptors in the
      test image
      vector<KeyPoint> test_kp;
      Mat test_desc;
      featureDetector.detect(test_g, test_kp);
      featureExtractor.compute(test_g,test_kp,test_desc);
      //match train and test descriptors, getting 2 nearest
      neighbors for all test descriptors
      vector<vector<DMatch> > matches;
```

```
        matcher.knnMatch(test_desc,matches,10);
        //filter for good matches according to Lowe's algorithm
            vector<DMatch> good_matches;
            Mat img_show;
            vector<KeyPoint> keypoints_1;
            vector<KeyPoint> keypoints_2;
            for ( i=0;i<matches.size();i++) {
            if (matches[i][0].distance <
            0.5*matches[i][1].distance)
            {
                good_matches.push_back(matches[i][0]);
            }
        }
  //-- Localize the object
  std::vector<Point2f> obj;
  std::vector<Point2f> scene;

  drawMatches(test,test_kp, train, train_kp, good_matches,
  img_show,Scalar::all(-1), Scalar::all(-
  1),vector<char>(),DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
    Point2f point1;
    int average_X=0;int average_Y=0;
    if (good_matches.size() >= 4){
        for( int i = 0; i < good_matches.size(); i++ )
        {
   //-- Get the keypoints from the good matches
   obj.push_back( train_kp[ good_matches[i].trainIdx ].pt );
   scene.push_back(test_kp[ good_matches[i].queryIdx ].pt );
   point1=test_kp[ good_matches[i].queryIdx ].pt;
   average_X+=point1.x; //get the coordinate of x
        }
        average_X=(average_X)/good_matches.size();
        printf("pointx: %d  pointy: %d \n",point1.x, point1.y);
  cv::rectangle(img_show , cvPoint( average_X-55, point1.y-50) ,
  cvPoint( average_X+50, point1.y+50)  , Scalar( 255, 0, 255),
  1 );
  }
  imshow("Matches", img_show);
  cout<<"Frame rate="<<getTickFrequency()/(getTickCount()-
  t0)<<endl;
 }
 return 0;
 }
```

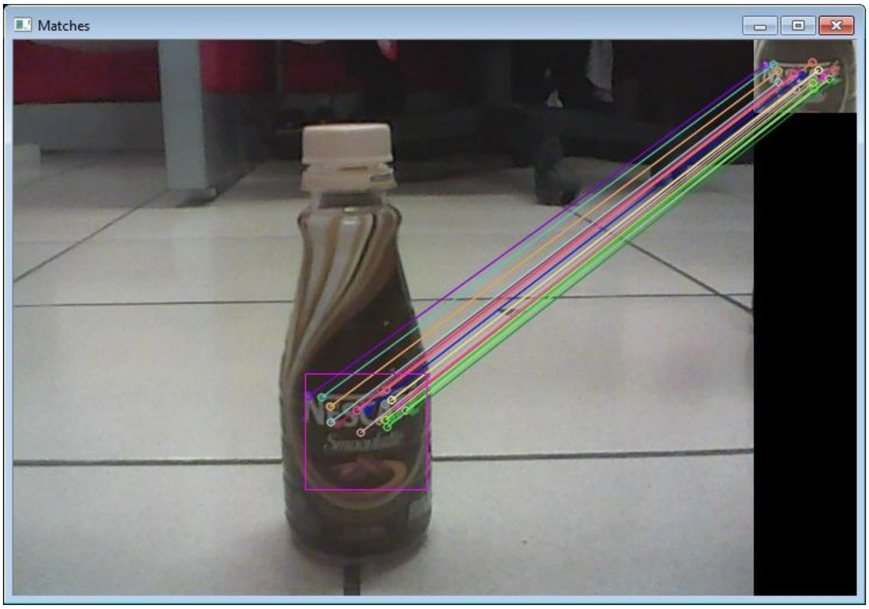

Figure 10.11 *Robust Object detector using FLANN based matcher, rectangle line used to get center position of the object.*

References

[1] Adrian Kaehler & Garry Bradksy, Learning OpenCV: Computer Vision in C++ with the OpenCV Library, O'Reilly PUblisher, 2014.

[2] Samarth Brahmbatt, Practical OpenCV, Technology in Action Publisher, 2013.

[3] Daniel bagio et al., Mastering OpenCV with Practical Computer Vision Project, Packt Publisher, 2012.