# Event Control through Motion Detection

*Project submitted in partial fulfillment of the requirements for the degree of*

## Bachelor of Technology

*in*

## Computer Science and Engineering

*by*

**Vivek Bhatt**
**Ankur Samantara**
**Kamaljeet Singh**


**Department of Computer Science and Engineering**
**National Institute of Technology Rourkela**
**Rourkela, Odisha, 769 008, India**

**May 2011**

# Event Control through Motion Detection

*Project submitted in partial fulfillment of the requirements for the degree of*

## Bachelor of Technology

*in*

## Computer Science and Engineering

*by*

**Vivek Bhatt(107CS021)**
**Ankur Samantara(107CS026)**
**Kamaljeet Singh(107CS054)**

*Under the guidance of*

## Prof. Pankaj Kumar Sa


**Department of Computer Science and Engineering**
**National Institute of Technology Rourkela**
**Rourkela, Odisha, 769 008, India**

**May 2011**

Department of Computer Science and Engineering
**National Institute of Technology Rourkela**
Rourkela-769 008, India. www.nitrkl.ac.in

**Dr. Pankaj K Sa**
Assistant Professor

May 09, 2011

# Certificate

This is to certify that the work in the project entitled **Event Control through Motion Detection** by **Vivek Bhatt**, **Ankur Samantara**, and **Kamaljeet Singh** is a record of an original work carried out by them under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Bachelor of Technology* in *Computer Science and Engineering*. Neither this project nor any part of it has been submitted for any degree or academic award elsewhere.

***Pankaj K Sa***

# Acknowledgment

We are grateful to numerous local and global peers who have contributed towards shaping this project. At the outset, We would like to express our sincere thanks to Prof. Pankaj Kumar Sa for his advice during our project work. As our supervisor, he has constantly encouraged us to remain focused on achieving our goal. His observations and comments helped us to establish the overall direction of the research and to move forward with investigation in depth. He has helped us greatly and been a source of knowledge.

We are higly indebted to Prof. Ashok Kumar Turuk, Head-CSE, for his continuous encouragement and support, as he has always been eager to help. We are also thankful to all the professors of the department for their support.

We are thankful to all our friends. Our sincere thanks to everyone who has provided us with inspirational words, a welcome ear, new ideas, constructive criticism, and their invaluable time, We are truly indebted.

We must acknowledge the academic resources that we have acquired from NIT Rourkela. We would like to thank the administrative and technical staff members of the department who have been kind enough to advise and help in their respective roles.

Last, but not the least, we would like to dedicate this project to our families, for their love, patience, and understanding.

*Vivek Bhatt, Ankur Samantara and Kamaljeet Singh*

# Abstract

Computer Vision is the study of machines that extract information from an image and perform some processing on the captured images to extract necessary data to solve some task. As a scientific discipline, the study of computer vision is concerned with the theories behind artificial systems that extract information from images. The image data could in several different forms and formats, such as video sequences, views from multiple cameras, or multi-dimensional data acquired from a medical scanner. As a technological discipline, computer vision intends to apply its theories and models to the construction and design of computer vision systems

The role of computer vison in robots is providing detailed information about the environment. A robust vision system should be able to detect and identify objects reliably and provide an accurate representation of the environment to higher level processes. The vision system should also be highly efficient, allowing a resource limited agent to respond quickly to a changing environment. Each frame acquired by a digital camera must be processed in a small, usually fixed, amount of time. Algorithmic complexity is therefore constrained, introducing a tradeoff between processing time and the quality of the information acquired. In most robotic applications, the vision system is the main perception device and autonomous robots must be capable of using it in order to self-localize and locate the objects that they have to manipulate.

The objective of the project was to build a computer controlled bot which could collect and deposit balls rolling down a ramp with the help of overhead/onboard camera.The ojective was achieved with the use of Motion History Image(MHI) based image processing algortihms and microcontroller based controling of motors.

# Contents

# List of Figures

# Chapter 1

# Introduction

*Real Time Systems*

*Computer Vision*

*Literature Survey*

*Goal and Objectives*

*Problem Formulation*

*Hardware and Software Requirements*

# Chapter 1

# Introduction

## 1.1 Real Time Systems

In computer science, real-time computing (RTC), or reactive computing, is the study of hardware and software systems that are subject to a "real-time constraint"– *i.e.*, operational deadlines from event to system response. Real-time programs must execute within strict constraints on response time[1]. By contrast, a non-real-time system is one for which there is no deadline, even if fast response or high performance is desired or preferred. The needs of real-time software are often addressed in the context of real-time operating systems, and synchronous programming languages, which provide frameworks on which to build real-time application software.

A real-time system may be one where its application can be considered (within context) to be mission critical. The anti-lock brakes on a car are a simple example of a real-time computing system  the real-time constraint in this system is the time in which the brakes must be released to prevent the wheel from locking. Real-time computations can be said to have failed if they are not completed before their deadline, where their deadline is relative to an event. A real-time deadline must be met, regardless of system load.

## 1.2 Computer Vision

It is the study of machines that are able to extract information from an image that is necessary to solve some task. As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems. Examples of applications of computer vision include systems for:

1. Controlling processes (e.g., an industrial robot or an autonomous vehicle).
2. Detecting events (e.g., for visual surveillance or people counting).
3. Organizing information (e.g., for indexing databases of images and image sequences).
4. Modeling objects or environments (e.g., industrial inspection, medical image analysis or topographical modeling).
5. Interaction (e.g., as the input to a device for computer-human interaction).

### 1.2.1 Role of CV in Robotics

Vision is essential for both humans and robots for providing detailed information and the reconstruction of the environment. It should be possible for a robust vision system to detect objects reliably and provide an accurate representation of the world for processing by the higher level processes. The vision system must necessarily be highly efficient, allowing a resource constrained agent to respond quickly and timely to a changing environment. Each frame acquired by a digital camera must be processed in a small, and fixed, amount of time. Algorithmic complexity is therefore a guiding factor, introducing a tradeoff between processing time and the quality of the information acquired. In most robotic systems, the

vision system is the main perception medium and autonomous robots must be capable of using it in order to self-localize and locate and identify the objects that they have to manipulate, and respond accordingly.

### 1.2.2 openCV(the tool)

The OpenCV Library is mainly aimed at real time computer vision. Some example areas would be Human-Computer Interaction (HCI); Object Identification, Segmentation, and Recognition; Face Recognition; Gesture Recognition; Motion Tracking, Ego Motion,and Motion Understanding; Structure From Motion (SFM); and Mobile Robotics.

The OpenCV Library is a collection of low-overhead, high-performance operations performed on images. The OpenCV implements a wide variety of tools for image interpretation. It is compatible with Intel Image Processing Library (IPL) that implements low-level operations on digital images. In spite of primitives such as binarization, filtering, image statistics, pyramids, OpenCV is mostly a high-level library implementing algorithms for calibration techniques (Camera Calibration), feature detection (Feature) and tracking (Optical Flow), shape analysis (Geometry, Contour Processing), motion analysis (Motion Templates, Estimators), 3D reconstruction (View Morphing), object segmentation and recognition (Histogram, Embedded Hidden Markov Models, Eigen Objects)[2].

## 1.3 Literature Survey

### 1.3.1 MHI

The MHI is a static image template where pixel intensity is a function of the motion history at that location, where brighter values correspond to more recent motion.

**Motion Representation**

The formal description of MHI suggests that motion can be represented using MHI, as the motion is most likely to occur from the faintest (least recent) to the

brightest (most recent) pixels, in an MHI template.
This representation can be shown as follows:
A frame capturing a foreground silhouette of the moving object is shown in figure 1.1 (left) . Acquiring a clear silhouette is achieved through application of some back-ground subtraction techniques. As the object moves, copying the most recent foreground silhouette as the highest values in the motion history image creates a layered history of the resulting motion; normally this highest value is just a floating point timestamp of time elapsing since the application was launched in milliseconds.
The result, which is called the Motion History Image (MHI) is shown in figure 1.1 (right). A pixel level or a time-delta threshold, as appropriate, is set such that pixel values in the MHI image that fall below that threshold are set to zero. The most recent motion has the highest value, earlier motions have decreasing values subject to a threshold below which the value is set to zero. Different stages of creating and processing motion templates are described below.

**A.Updating MHI Images**

Generally, floating point images are used because system time differences, that is, time elapsing since the application was launched, are read in milliseconds to be further converted into a floating point number which is the value of the most recent silhouette.
Then follows writing this current silhouette over the past silhouettes with subsequent thresholding away pixels that are too old (beyond a maximum mhiDuration) to create the MHI.

**B) Making Motion Gradient Image**

1. Start with the MHI image as shown in Figure 1.1(left).

2. Apply 3x3 Sobel operators X and Y to the image.

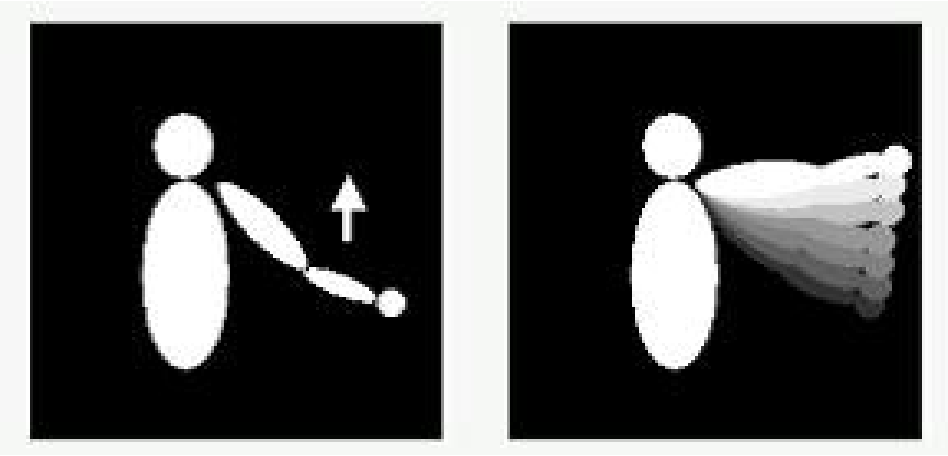

Figure 1.1: Motion History Image From Moving Silhouette

3. If the resulting response at a pixel location (X,Y) is Sx(x,y) to the Sobel operator X and Sy(x,y) to the operator Y, then the orientation of the gradient is calculated as:

$$A(x, y) = \arctan \frac{Sy((x, y)}{Sx(x, y))}$$

, and the magnitude of the gradient is:

$$M(x, y) = \sqrt{(}Sx2(x, y) + Sy2(x, y))$$

4. The equations are applied to the image yielding direction or angle of a flow image superimposed over the MHI image as shown in Figure 1.2.

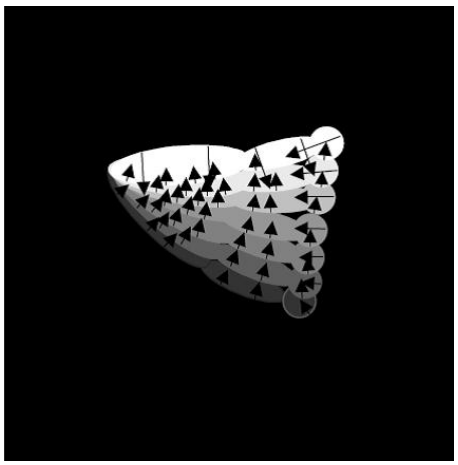

Figure 1.2: Direction of Flow Image

5. The boundary pixels of the MH region may give incorrect motion angles and magnitudes, as Figure 1.2 shows. Thresholding away magnitudes that are either too large or too small can be a remedy in this case. Figure 1.3 shows the ultimate results.[2][3][4]

Figure 1.3: Resulting Normal Motion Directions

### 1.3.2 Serial Communication

Serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels. Serial communication is used for all long-haul communication and most computer networks, where the cost of cable and synchronization difficulties make parallel communication impractical

### 1.3.3 RGB color Coding

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

To form a color with RGB, three colored light beams (one red, one green, and

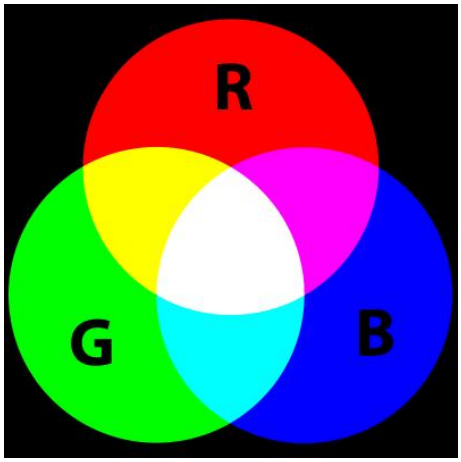


Figure 1.4: RGB Color Model

one blue) must be superimposed (for example by emission from a black screen, or by reflection from a white screen). Each of the three beams is called a component of that color, and each of them can have an arbitrary intensity, from fully off to fully on, in the mixture.

The RGB color model is additive in the sense that the three light beams are added together, and their light spectra add, wavelength for wavelength, to make the final color's spectrum.[6][7]

Zero intensity for each component gives the darkest color (no light, considered the black), and full intensity of each gives a white; the quality of this white depends on the nature of the primary light sources, but if they are properly balanced, the result is a neutral white matching the system's white point. When the intensities for all the components are the same, the result is a shade of gray, darker or lighter depending on the intensity. When the intensities are different, the result is a colorized hue, more or less saturated depending on the difference of the strongest and weakest of the intensities of the primary colors employed.

When one of the components has the strongest intensity, the color is a hue near this primary color (reddish, greenish, or bluish), and when two components have the same strongest intensity, then the color is a hue of a secondary color (a shade of cyan, magenta or yellow). A secondary color is formed by the sum of two primary colors of equal intensity: cyan is green+blue, magenta is red+blue, and yellow is red+green. Every secondary color is the complement of one primary color; when

a primary and its complementary secondary color are added together, the result is white: cyan complements red, magenta complements green, and yellow complements blue.[5] The RGB color model itself does not define what is meant by red, green, and blue colorimetrically, and so the results of mixing them are not specified as absolute, but relative to the primary colors. When the exact chromaticities of the red, green, and blue primaries are defined, the color model then becomes an absolute color space, such as sRGB

## 1.4 Goal and Objectives

### 1.4.1 Goal

Build a computer controlled robot which can collect and deposit balls rolling down a ramp with the help of overhead/onboard camera.

### 1.4.2 Objectives

1. Building Of Mechanical Robot capable of picking up balls and dropping them at specific locations
2. Designing a developing software capable of communicating between robot and computer
3. Designing and developing software for image processing and extracting desired information.

## 1.5 Problem Formulation

**Arena Specifications** The Arena consists of two parts.

### 1.5.1 Ramp

1. It consists of a rectangular ramp of inner dimension 2400 mm × 1000 mm, Coloured 'green' (the playing surface is green felt mat or carpet). The floor under the carpet is level, flat and hard.
2. At the top end are 5 launching pods from where the balls are launched.
3. Immediately following the pods is a 300 mm wide Pin Belt consisting of 6 rows of pins, to obstruct the path of the ball and provide randomness to their trajectories.
4. Then there is 2000 mm of completely unhindered ramp for the ball to roll down.
5. The sides of the ramp are lined with walls 60 mm high.
6. The boundary between the ramp and platform is marked with a 30mm thick white line.

### 1.5.2 Collection Platform

1. At the base of the ramp is a flat collection zone of inner dimensions 300 mm × 1200 mm, coloured 'green' (the playing surface is green felt mat or carpet)
2. The platform is at a depth of 180 mm below the ramp.
3. At each end, there is a pocket of dimensions 300 mm × 60mm for collecting balls

### 1.5.3 Robot Specifications

1. Only one robot is to perform the entire task and no support is provided by any other means.
2. The robot must fit into a cube of 200 mm × 200 mm × 250 mm at all times. It may not expand at any point during its run beyond these dimensions , even for performing tasks like grabbing etc.

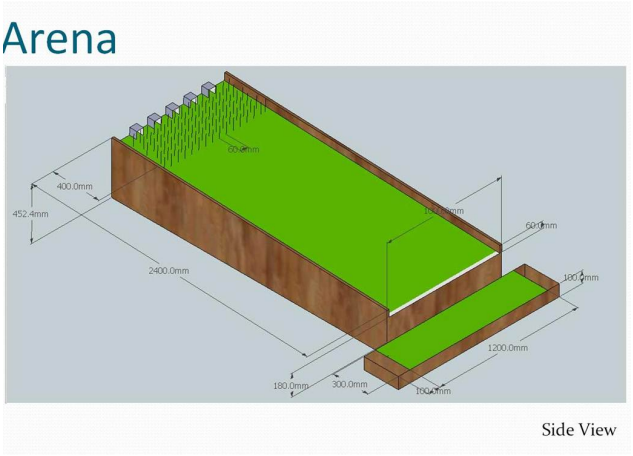


Figure 1.5: Arena

3. The uses an on board power supply. No external power supply is allowed. The max potential difference between any two points should not exceed 24V.

4. The robot can collect more than one ball at a time before depositing.

5. The robot is fully independent, with powering and motoring mechanisms self-contained. However, it can communicate with the computer using either wired or wireless data transfer.

6. The robotic equipment is fully autonomous. Human operators are not permitted to enter any information into the equipment during a run. The human operator should not directly control the motion of their robots with a joystick or by keyboard commands under any circumstances.

7. The robot cannot be constructed using readymade Lego kits or any readymade mechanism. But use of readymade gear assemblies and readymade wireless modules is allowed.

8. The robot has to work on the principle of image processing. Any sort of ambiguity is not allowed.

9. The robot is allowed to touch the boundary of the ramp freely for any kind of feedback.

## 1.6 Hardware and Software Requirements

### 1.6.1 Hardware Requirements:

1. **TSOP based obstacle detector / proximity sensing module** It is a

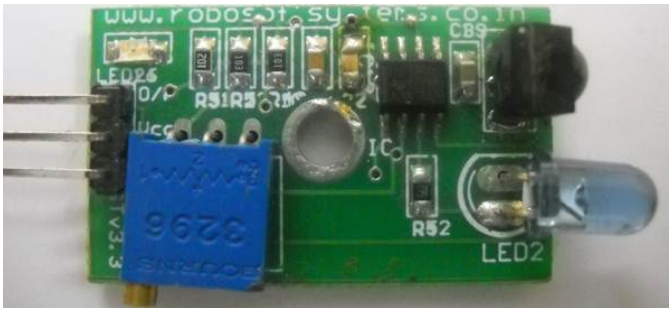


Figure 1.6: TSOP based obstacle detector

hardware module that gives a high value output (output 1) when an object is within its threshold distance and low output (output 0) when it is unable to detect the presence of an object within the threshold distance limit.

2. **Skid Steer Relay Motor Driver** It is a specially designed circuit for


Figure 1.7: Skid Steer Relay Motor Driver

controlling the motion of the motors in an efficient manner. It's specific features are:-

(a) 10 Amp. per motor capacity

(b) Provides automatic breaking (DC breaking) of motors in normal condition which makes it advantageous in competitions.

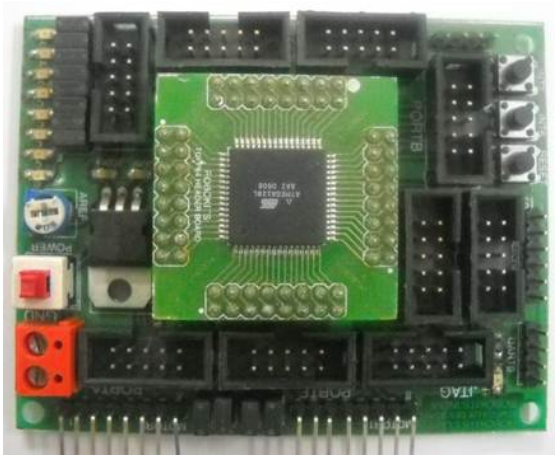

Figure 1.8: AtMega Rapid Robot Controller Board

3. **AtMega Rapid Robot Controller Board** A high speed microcontroller, Atmega 128 is used for controlling the robot and communicating with the computer. Its specific features are:-

   (a) On-board Regulator with filters and Operating voltage from 6V - 20 V

   (b) 8 LED's selectable though individual jumpers

   (c) 3 switches including reset

   (d) 2 switches on interrupt pins

   (e) Power on/off toggle switch

   (f) 16MHz crystal for maximum clock speed

   (g) AREF setting potentiometer

   (h) LED Power Indicator

   (i) All pins accessible through FRC 10 pin male connectors

   (j) 2 USART for serial communication

4. **PC-MCU Serial Link** This circuit module makes a virtual COM port on any PC when connected on a USB port, so it can be used for communication between PC software (Link Hyper Terminal) and microcontroller. Its specific features are :-

   (a) Can be used with any controller UART/USART

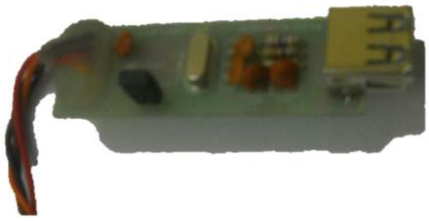

Figure 1.9: PC-MCU Serial Link

(b) Extra RTS and DTR signals

(c) External interface reduces board size and complexity

5. **High Torque DC geared Motor** High torque DC geared motor with

Figure 1.10: High Torque DC geared Motor

metal gearbox and off centered shaft is used for motion of the robot. A DC motor is also used for picking and depositing the balls. Specific features are—

(a) 450RPM 12V DC motors with Metal Gearbox

(b) 25000 RPM base motor

(c) 6mm shaft diameter

(d) Gearbox diameter 37 mm.

(e) Motor Diameter 28.5 mm

(f) Length 63 mm without shaft

(g) 300gm weight

(h) 20kgcm torque

(i) No-load current = 800 mA(Max), Load current = upto 9.5 A(Max)

6. **Lithium Polymer ( Li-Po) Rechargeable Battery 12.6V 2200mAH 20C** It is light weight and small size battery compared to any other battery of similar capacity. It is used to power the robot's on-board equipments. IMAX B5 5 Amp multipurpose battery charger is used for charging this battery.[8] Its specific features are—

   (a) Long life with full capacity for 1000 charge cycles

   (b) 3X Li-Po 4.2V 2200mAh cells (3S1P)

   (c) 192Grams Weight

   (d) Volume: 10cm × 3.3cm × 2cm

   (e) Discharge Current: 20 × 2200maH = 44Amp

   (f) Max Charging Current: 1A

7. **USB Atmega Programmer** A specifically designed circuit to burn the program on Atmega microcontroller. It has a USB based interface.

8. **Other supplementary materials** Some other hardware are required such as connecting wires, USB extension cables etc.

### 1.6.2 Software Requirements

1. Windows Operating System

2. Open CV

3. Dev Cpp IDE

4. AVR Studio 4

5. Extreme Electronics Burner

6. PC - MCU serial link driver

# Chapter 2

# Hardware Implementation

*Initializing Microcontroller*

*Data sending*

*Data receiving*

*Motor Movement*

*Robot control*

# Chapter 2

# Hardware Implementation

## 2.1 Initializing Microcontroller

The lower order 4 bits of PORTA are set to take input from the TSOP object detector sensor. The lower order 4 bits of PORTC are set to send output to control the movement of the motors. USART is initialized for the transmission of data between computer and the robot. The baud rate is set to 9600.

```
void USART1_Init()
{
        UBRR1H=0x00;
        UBRR1L=0x67;
        UCSR1B=0x98;
        UCSR1C=0x06;
        UCSR1A=0x00;
}
```

## 2.2 Data sending

For transmitting data we wait till the 1st bit on UCSR0A is not equal to 1. 1st bit on UCSR0A indicates whether output buffer is empty or not. Once output buffer is empty we can put 8 bit data in the UDR (USART data register) for transmission.

```
void USART0_Transmit(char data)
{
        /* Wait for data to be transmit */
while((UCSR0A & 0x20) != 0x20)
        {
        }
        /*Put the data into the buffer */
                UDR0=data;
}
```

## 2.3 Data receiving

For receiving data we wait till 4th bit on UCSR0A is not equal to 1. 4th bit on UCSR0A indicates whether input buffer is empty or not. Once output buffer is full we can take 8 bit data from the UDR (USART data register).

```
char USART0_Receive( void )
{
        /* Wait for data to be received */
        while ((UCSR0A & 0x80) != 0x80);
                /* Get and return received data from buffer */
                char data=UDR0;
        return data;
}
```

## 2.4 Motor Movement

Photo of Motor driver board Usage :

1. Connect the 6 wires to RC circuits and 4 wires to motors

2. Upon giving the battery/power supply input to circuit it will drive the motors as per the input from RC circuit.

1. Pin outs:

   (a) Output section (Screwed connector)

      i. Pin 1 : Motor 1 a

      ii. Pin 2 : Motor 1 b

      iii. Pin 3 : Motor 2 a

      iv. Pin 4 : Motor 2 b

   (b) Input Section (6 Pin male Header)

      i. Pin 1 : Forward

      ii. Pin 2 : Backward

      iii. Pin 3 : Left

      iv. Pin 4 : Right

   (c) Pin 5 : +ve to External Circuit

   (d) Pin 6 : -ve to External Circuit

2. Power Connector

   (a) Power Supply +

   (b) Power Supply -

## 2.5 Robot control

Data is received from computer for movement of the bot to specific location to collect specific color ball. On reaching the destination, the bot corrects its position based upon the correction data sent by the computer. After collecting the ball, bot goes to the targeted pit depending upon the color of the ball and deposits the ball there and comes back to the edge of the arena platform to wait for the next command.

Figure 2.1: Robot Top View

Figure 2.2: Robot Side View

Figure 2.3: Robot Front View

# Chapter 3

# Software Implementation

**Ball Detection**

**Updating Motion History Image**

**Calculating motion**

**Communicating with Robot**

# Chapter 3

# Software Implementation

**Steps of IP Algorithm**

1. Ball Detection
2. Updating MHI
3. Calculating Motion
4. Communicating with the Robot

## 3.1 Ball Detection

The first step in solving the given problem statement involves the detecton o f the ball as it rolls down the ramp. This also involves the identification tof the colour of the rolling ball.

The following sub-steps have to be followed to perform this step

1. Capture of image
2. Separation of the background and the foreground:
3. Detection of colour

### 3.1.1 Capture of image:

Image is captpured from hte image feed from the overhead camera.

```
In opencv caputure is a Image variable
      capture = cvCaptureFromFile( argv[1] );
```

### 3.1.2 Separation of the background and the foreground

Here first all the pixels having RGB values for colours other than GREEN, which is the background are selected. These pixel groups give us the presence of an object which is ball in this case.

### 3.1.3 Detection of colour

Different Color filters are applied to the image obtained by seperating the foreground and the background (the input frame) and results were stored in different Image varaibles

**Filter for RED colour**

```
input image = imageIn
output image = imageR

for(int i=1;i<imageIn->height;i++)
  {
        for(int j=0;j<imageIn->width;j++)
        {
            s1=cvGet2D(imageIn,i,j);
            R=s1.val[0];
           G=s1.val[1];
            B=s1.val[2];

            if (R>=100&&G<100&&B<100)
            {
```

```
                    s2.val[0]=255;
                s2.val[1]=255;
                    s2.val[2]=255;
                }
                else
                {
                        s2.val[0]=0;
                        s2.val[1]=0;
                        s2.val[2]=0;
                }
            cvSet2D(imageR,i,j,s2);
            }
        }
```

The above code snippet identifies the red coloured balls rolling down the ramp. Similarly filter for other colours can be applied using approprite RGB values and other colours can be identified.

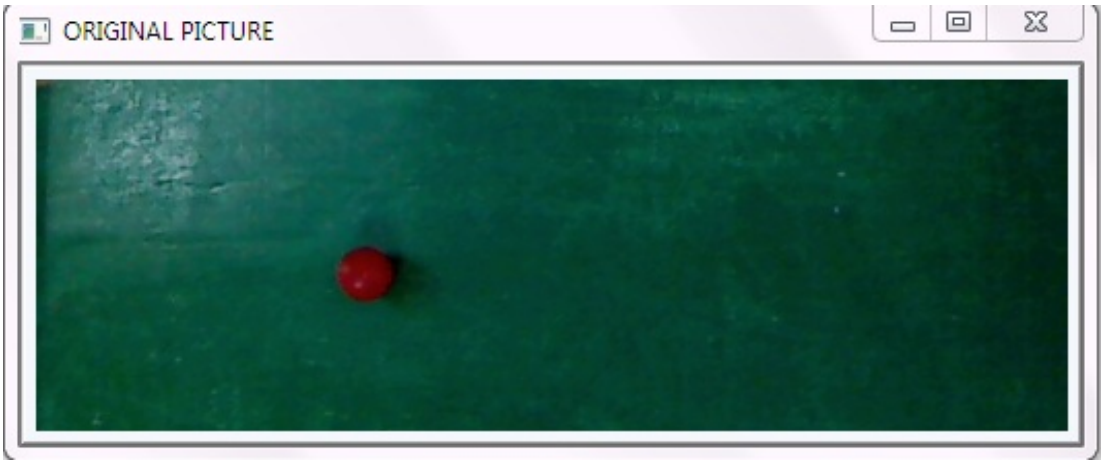


Figure 3.1: Original Image

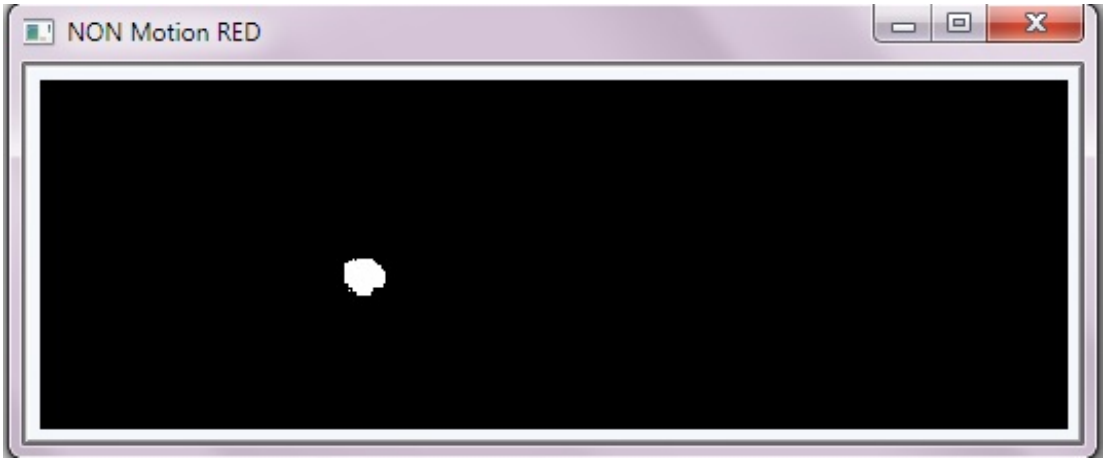


Figure 3.2: Image after Red Color Filter

## 3.2 Updating Motion History Image

1. Convert frame to grayscale
2. Get difference between frames
3. Threshold the image
4. Update MHI

Once the ball is located and its colour has been identified, the next step is to create and update a Motion History Image (MHI). The following are the sub-steps involves:

### 3.2.1 Convert frame to grayscale

To begin with the input image , which is a filtered image and consists of only one colour pixels is taken and it is converted into grayscale. This steps leads to an image which has all background pixels as black pixels and all the pixels where ball is present as white pixels. This allows to easily find the difference between two frames.

```
    cvCvtColor( img, buf[last], CV_BGR2GRAY )
where RGB[A]->Gray: Y<-0.299*R + 0.587*G + 0.114*B
```

### 3.2.2 Get difference between frames

Next find the difference between successive frames taking the grayscale images aas the input

```
cvAbsDiff( const CvArr* src1, const CvArr* src2, CvArr* dst );

  where, src1=First Source Array
         src2=Second Source Array
         dst=Destination Array
```

### 3.2.3 Threshold the image

Threshold values for several parameters are specified that govern, by what minimum values several parameters of two successive images should differ to be considered as a differences between two images.

```
cvThreshold( silh, silh, 30, 1, CV_THRESH_BINARY);

 where, silh=Source Array
        silh=Destination Array
        30 =Threshold Value
```

### 3.2.4 Update MHI

Finally the Motion History image is updated using silhuotte image over the MHI image.The function UpdateMotionHistory updates the motion history image with a silhouette, assigning the current timestamp value to those mhi pixels that have corresponding non-zero silhouette pixels. The function also clears mhi pixels older than timestamp  mhiDuration if the corresponding silhouette values are 0.

```
cvUpdateMotionHistory( silh, mhi, timestamp, MHI_DURATION );
```

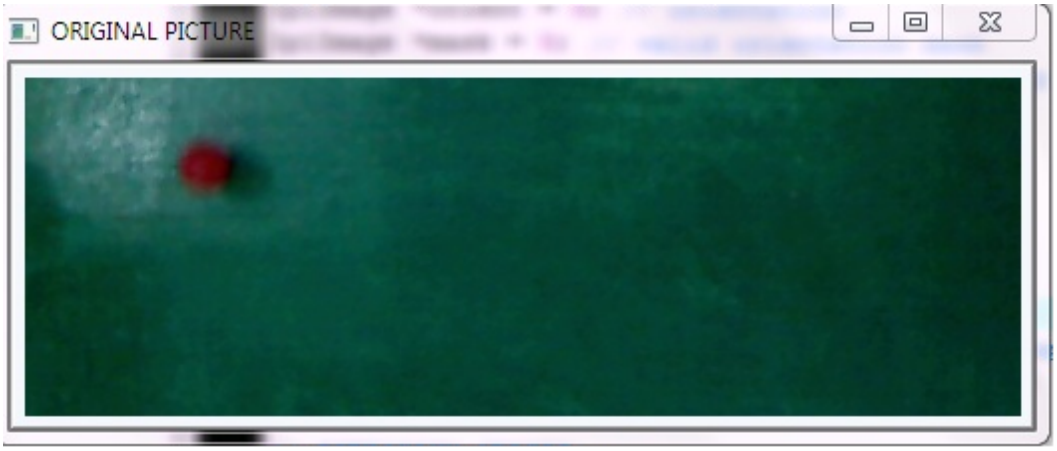


Figure 3.3: Original Image


Figure 3.4: Image showing movement and its direction

## 3.3 Calculating motion

1. Calculate Motion Gradient
2. Segment Motion
3. Removing Noise
4. Calculation Motion Centers and motion angle

### 3.3.1 Calculate Motion Gradient

The function finds minimum (m(x,y)) and maximum (M(x,y)) MHI values over each pixel (x,y) neighborhood and assumes the gradient is valid only if

$$"MIN\_TIME\_DELTA <= M(x,y) - m(x,y) \leq MAX\_TIME\_DELTA".$$

```
  cvCalcMotionGradient( mhi, mask, orient, MAX_TIME_DELTA,
MIN_TIME_DELTA, 3 );
where ,
mhi = Motion history image.
mask =Mask image, marks pixels where motion gradient data
is correct.
orient=Motion gradient orientation image; contains angles
from 0 to ~360.
3 = Aperture size of derivative operators used by the function
```

The function cvCalcMotionGradient calculates the derivatives dx and dy of MHI and then calculates gradient orientation as:

$$orientation(x,y) = \arctan\frac{dy}{dx}$$

### 3.3.2 Segment Motion

Identifying the several sequence of motion components

```
seq = cvSegmentMotion( mhi, segmask, storage, timestamp, MAX_TIME_DELTA );
where
  segMask = Image where the mask found should be stored, single-channel,
 32-bit floating-point.
storage = Memory storage that will contain a sequence of motion connected
 components.
timestamp = Current time in milliseconds or other units.
segthresh = Segmentation threshold.
```

### 3.3.3 Removing Noise

Noise in Motion segments is removed on the basis of:

1. Size of Motion segment
2. Percentage motion in Motion segment

#### Size of Motion segment

Very small motion sizes can also be neglected as the grayscale image of the same object by the same digital camera differs by a few isolated pixels , so these can be attributee to this feature.

#### Percentage motion in Motion segment

All differences below a certain percenatge is neglected as that can be associated with the noise in the signal. Only the differences above a certain value are considered to be be rolling balls.

### 3.3.4 Calculation Motion Centers and motion angle

This step involves detection of the direction and angle of motion using the motion History image.

#### angle

```
    angle = cvCalcGlobalOrientation( orient, mask, mhi, timestamp,
MHI_DURATION);
         angle = 360.0 - angle;
// adjust for images with top-left origin
         count = cvNorm( silh, 0, CV_L1, 0 );
// calculate number of points within silhouette ROI
```

**center**

```
center = cvPoint( (comp_rect.x + comp_rect.width/2),
        (comp_rect.y + comp_rect.height/2) );
```

## 3.4 Communicating with Robot

Once the ball and its colour has been detected and the angle and direction of motion are established, these values are to be communicated to the ROBOT using SERIAL communication.
Following steps are involved in setting up the serial communication :

1. Setting up of channel
2. Sending data
3. Recieving data

### 3.4.1 Setting up of channel

Here we create a handle on pcCommPort (Com4) with read/write exclusive access and default security attributes. Its attributes are then set to BaudRate = 9600, ByteSize =8, Parity = NOPARITY, StopBits = ONESTOPBIT

```
HANDLE hCom = CreateFile( pcCommPort, GENERIC_READ |GENERIC_WRITE,
0,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL)
```

### 3.4.2 Sending data

```
void SerialPutc(HANDLE *hCom, char txchar)
{
BOOL bWriteRC;
static DWORD iBytesWritten;
```

```
bWriteRC = WriteFile(*hCom, &txchar, 1,&iBytesWritten,NULL);
if(!bWriteRC)
{printf("error");}
return;
}
```

### 3.4.3 Recieving Data

```
char SerialGetc(HANDLE *hCom)
{
char rxchar;
BOOL bReadRC;
static DWORD iBytesRead;
bReadRC = ReadFile(*hCom, &rxchar, 1, &iBytesRead, NULL);
if(!bReadRC)
{ printf("error");}
return rxchar;
}
```

# Chapter 4

## Conclusions

# Chapter 4

# Conclusions

An image processing system was designed as per the problem statement which successfully made runs and achieved the objective. The system successfully collected balls rolling down the ramp and deposited them in the desired locations.

Future work may include randomization of problem statement to get results under more varying circumstances and a more realistic approach. Also better image processing can be done to avoid lighting condition dependency and better shadow removal techniques can be developed.

# Bibliography

[1]Ben-Ari, M., *Principles of Concurrent and Distributed Programming*,Prentice Hall, 1990. ISBN 0-13-711821-X. Ch16, Page 164.

[2] Intel OpenCV Reference Manual 1999-2001, `http://developer.intel.com`.

[3]J. Davis and Bobick , *The Representation and Recognition of Action Using Temporal Templates*,MIT Media Lab Technical Report 402, 1997.

[4] G. Bradski and J. Davis , *Motion Segmentation and Pose Recognition with Motion History Gradients*,IEEE WACV'00, 2000.

[5]Rafeel C. Gonzalez and Richard E. Woods, *Digital Image Processing, third edition*,ISBN 978-81-317-2695-2 ,Prentice Hall PTR, 2008.

[6]Charles A. Poynton , *Digital Video and HDTV: Algorithms and Interfaces*,Morgan Kaufmann. ISBN 1558607927, 2003.

[7]Nicholas Boughen, *Lightwave 3d 7.5 Lighting*, Wordware Publishing, Inc. ISBN 1556223544, 2003.

[8]Roboshop, `www.robokits.co.in`.

[9]R. Fisher, K Dawson-Howe, A. Fitzgibbon, C. Robertson, E. Trucco , *Dictionary of Computer Vision and Image Processing*,ISBN 0-470-01526-8,John Wiley,2005.

[10]Online Free Encyclopedia. `http://en.wikipedia.org`.

[11]Adam Osborne, *An Introduction to Microcomputers Volume 1: Basic Concepts*,Osborne-McGraw Hill Berkeley California USA, 1980 ISBN 0-931988-34-9 pp. 116-126.

[12]AVR Atmega128 datasheet , `http://www.atmel.com/atmel/acrobat/doc2467.pdf`.

[13]Pedram Azad, Tilo Gockel, Rdiger Dillmann, *Computer Vision - Principles and Practice*,Elektor International Media BV. ISBN 0905705718, 2008.

[14]Wilhelm Burger and Mark J. Burge, *Wilhelm Burger and Mark J. Burge*,Springer.

ISBN 1846283795 and ISBN 3540309403,2007.

[15]Tim Morris, *Computer Vision and Image Processing*, Palgrave Macmillan. ISBN 0-333-99451-5, 2004.

[16]Gady Agam, *Introduction to programming with OpenCV*, `http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html`, 2006.