Project Report

Demonstration of Steepest Descent and Newton's Optimization Technique Using A Robot

WORKPLACE: INDIAN INSTITUTE OF TECHNOLOGY, MADRAS

Project Guide

Dr. Sridharakumar Narasimhan

Assistant Professor

Chemical Department

IIT Madras

Submitted by:

P S Vishnu

3rd year B.Tech,

Mechanical Engineering

NIT, Tirichirappalli

Index

Abstract

Several optimization techniques are available in order to select the best element from some set of available alternatives.

Steepest descent also known as **Gradient descent** is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point.

Newton's method is an iterative method for finding roots of equations. More generally, Newton's method is used to find critical points of differentiable functions, which are the zeros of the derivative function. Newton's method takes into consideration the second order effects of a function. So if a function which is second order is optimized by Newton's method, the method gives the critical point in a single iteration.

In this project the above optimization techniques namely, steepest descent and Newton's optimization algorithm is modeled using a robot trying to perform a specific task. The robot utilizes the above techniques in order to perform its task. This project encompasses the use of the above optimization techniques and also specifies intricate nuances in building a robot which require control systems in every step of the way.

This project is aimed at emphasizing the importance of control and optimization in every aspect of technology.

Problem Statement

The problem statement is in general to minimize a given function. A robot is placed in an arbitrary point and in an arbitrary orientation on an arena. The arena is a contour of an arbitrary function which has to be minimized. So the value of the function on any point on the arena is the relative brightness or darkness of that point on the arena.

In this particular project a contour of a paraboloid is chosen as the arena. The center of the contour which is an ellipse is the darkest corresponding to the minimum and the contour becomes brighter away from the center with a constant gradient. So the robot should traverse to the center of the ellipse (the minimum) and stop. A figure of the contour is shown below.

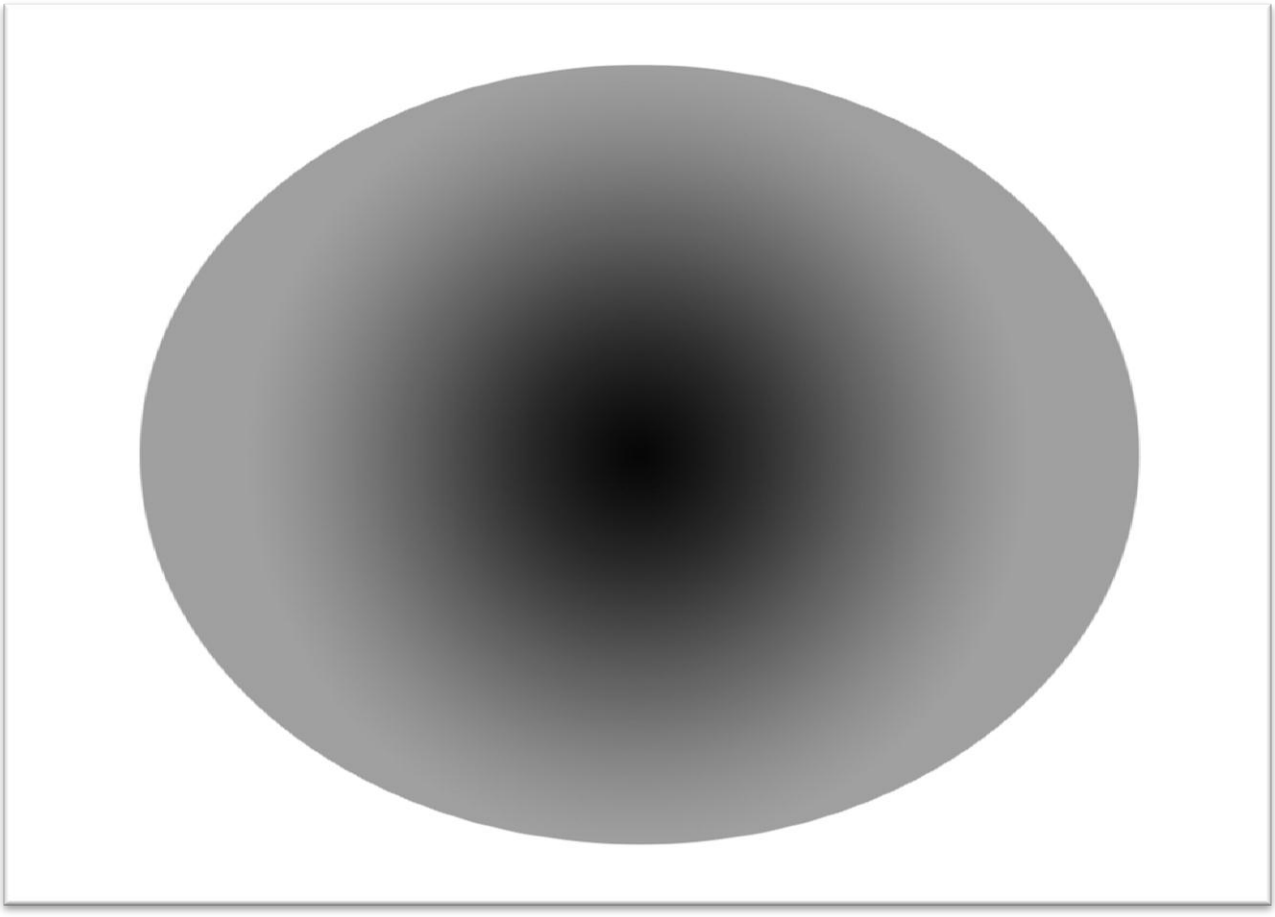

Fig1. The contour of the function the which the robot has to minimize

Introduction

This project can be divided into four major components.

1. Creating the arena
2. Building the robot
3. Selection of the software
4. Algorithms and logic

1) Creating The Arena

The arena as shown in figure1 is a contour of a paraboloid. The contour is an ellipse of varying intensity along its axes. The ellipse has its darkest point at the center indicating the minimum and becomes brighter towards the ends with a uniform gradient.

Since the arena has to be sufficiently large the contour was printed on an A1 sheet. Initially Matlab was used to make a contour but the problem was the resolution of the image. The image obtained on the A1 sheet appeared as bands of ellipses with the innermost band being the darkest and the outermost band being the brightest. It resulted in poor estimation of the descent direction in case of the steepest descent method and the Newton's method.

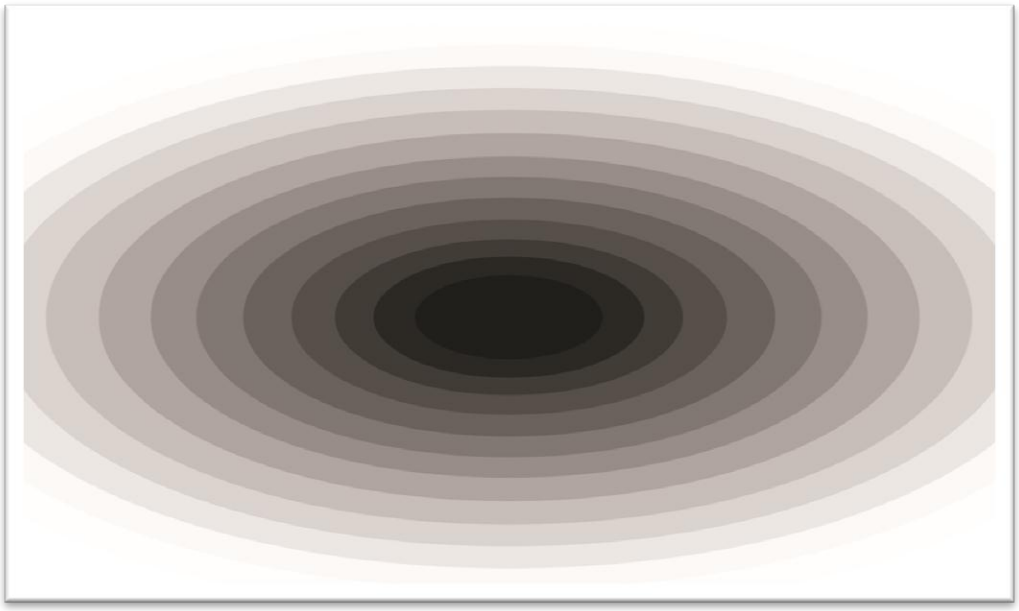

Fig2. The contour obtained by Matlab

So Photoshop was used to design the arena. A circle was drawn in Photoshop and a gradient was filled in it using a gradient tool and the circle was stretches along its side to get an ellipse with a proper gradient. The method produced a proper resolution and a uniform gradient in the printout. The better arena significantly improved the accuracy of the robot.

2) Building The Robot

The LEGO MINDSTORMS NXT 2.0 robotics kit was used for the construction of the robot. For this project the following electronic components were used other than the NXT Brick and the several plastic building components.

1. Three servo motors

Fig3. Lego mindstorm servo motor

2. A Lego Light Sensor

Fig4. Lego mindstorm light sensor

3. A Lego Color Sensor

Fig5. Lego mindstorm color sensor

The robot uses two servo motors connected to wheels to produce a differential drive for traversing that is moving forward or backward and for turning. There is a wheel on the rear side of the robot which is capable of rotating in all direction performing the function of an omnidirectional wheel.

There is also a servo motor on top of the robot which has a spindle. The spindle is connected to a Lego color sensor or a Lego light sensor allowing it to take readings along a small circle of radius 30mm in the steepest descent algorithm and 58mm in the Newton's algorithm.

Fig6. The completed robot side view

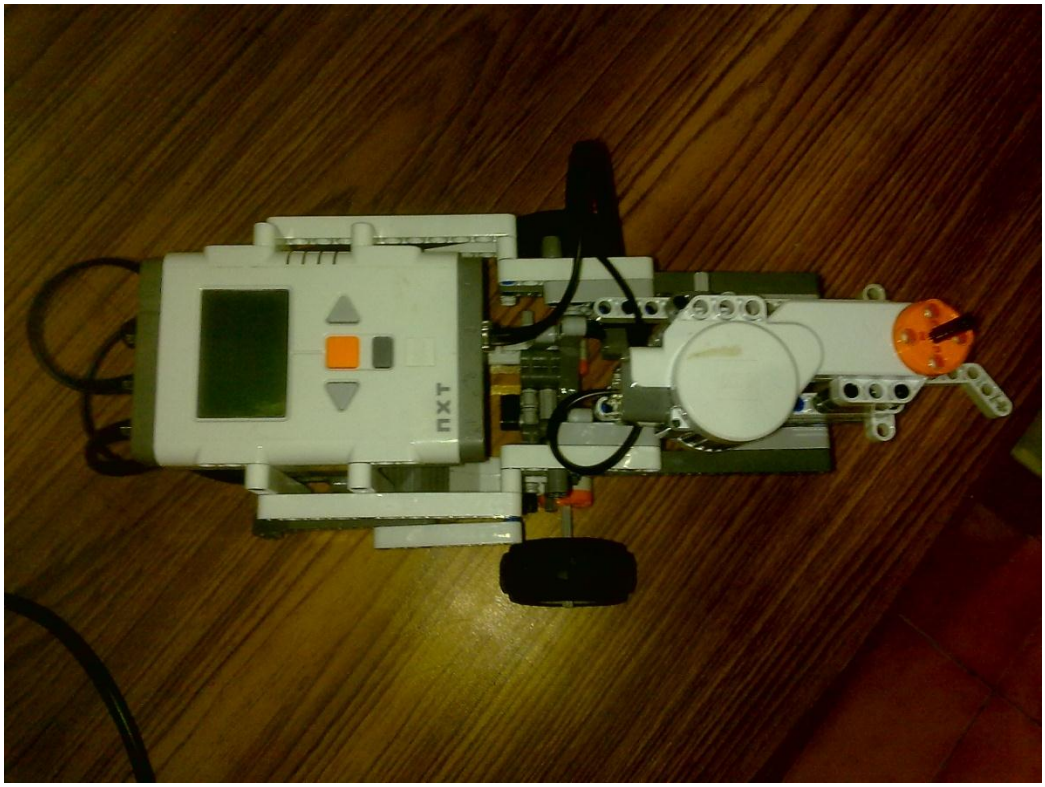

Fig7. The completed robot top view

3) Selection of the software

The software used for the programming the LEGO MINDSTORMS NXT brick is **Matlab**. The RWTH - Mindstorms NXT Toolbox for MATLAB is the toolbox used. This toolbox is developed to control LEGO MINDSTORMS NXT robots with MATLAB via a wireless Bluetooth connection or via USB.

The toolbox functions are based on the LEGO MINDSTORMS NXT Bluetooth Communication Protocol to control the intelligent NXT Brick via a wireless Bluetooth connection or via USB. Although a Bluetooth connection is not recommended for a real-time robot controlling in general (because of its high latency) this toolbox provides MATLAB functions to interact with a robot directly. Latencies via USB connections are much smaller and allow for more sophisticated applications.

The main advantage of this remote control concept enables you to combine robot applications with complex mathematical operations and visualizations within MATLAB. For further details about the toolbox visit

4) Algorithms and logic

Since the function is unknown to the robot we need to find out the gradient value in the case of steepest descent and the hessian and the jacobian in the case of Newton's method **Numerically**.

Two algorithms are used in this project to minimize the function and in this particular case to move to the darkest point or the minimum.

1. Steepest Descent
2. Newton's method

1) Steepest Descent

Steepest descent also known as Gradient descent is a **first-order optimization algorithm**. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

Gradient descent is based on the observation that if the real-valued function 'F(x)' is defined and differentiable in a neighborhood of a point 'a', then F(x) decreases fastest if one goes from 'a' in the direction of the negative gradient of F at 'a',- $\nabla F(a)$. It follows that, if

$$b = a - \gamma \nabla F(a)$$

for $\gamma > 0$ a small enough number, then . With this observation in mind, one starts with a guess for x_0 a local minimum of F, and considers the sequence x_0, x_1, x_2 such that

$$x_{n+1} = x_n - \gamma_n \nabla F(x_n),\ n \geq 0$$

We have

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \dots ,$$

so hopefully the sequence (x_n) converges to the desired local minimum. Note that the value of the step size γ is allowed to change at every iteration.

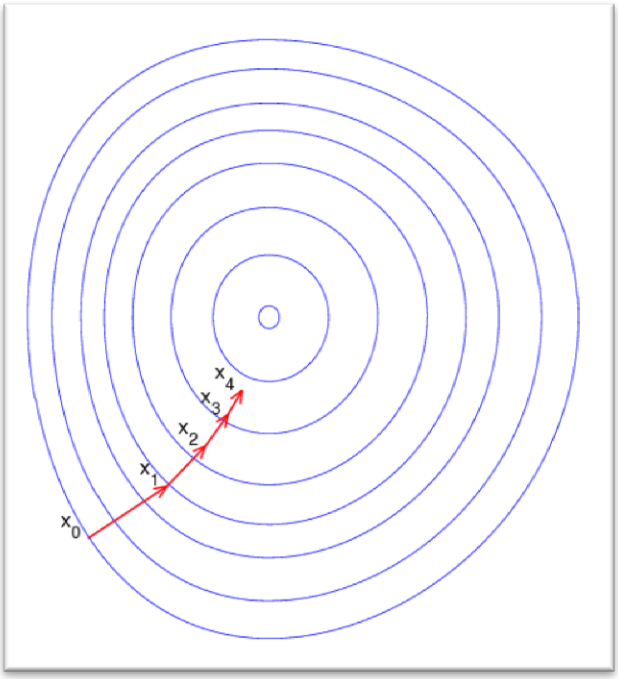

Fig8.Illustration of gradient descent

This process is illustrated in the above picture. Here F is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of F is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient descent leads us to the bottom of the bowl, that is, to the point where the value of the function F is minimal.

2) Newton's Method

In mathematics, Newton's method is an iterative method for finding roots of equations. More generally, Newton's method is used to find critical points of differentiable functions, which are the zeros of the derivative function.

Newton's Method attempts to construct a sequence x_n from an initial guess x_0 that converges towards x_* such that $f'(x_*) = 0$. This x_* is called a stationary point of $f(\cdot)$.

The second order Taylor expansion of $f(x)$ around x,

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$$

,

Attains its extremum when its derivative with respect to Δ*x* is equal to zero, i.e. when Δ*x* solves the linear equation:

$$f'(x) + f''(x)\Delta x = 0.$$

(Considering the right-hand side of the above equation as a quadratic in Δ*x*, with constant coefficients.)

Thus, provided that $f(x)$ is a twice-differentiable function well approximated by its second order Taylor expansion and the initial guess x_0 is chosen close enough to x_*, the sequence (x_n) defined by:

$$\Delta x = -\frac{f'(x_n)}{f''(x_n)}$$

$$\boxed{x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)},\ n = 0, 1, \ldots}$$

will converge towards a root of *f'*, i.e. x_* for which $f'(x_*) = 0$.

For a higher order function the above iterative scheme can be generalized to several dimensions by replacing the derivative with the gradient, $\nabla f(\mathbf{x})$, and the reciprocal of the second derivative with the inverse of the Hessian matrix, $Hf(\mathbf{x})$. One obtains the iterative scheme

$$\mathbf{x}_{n+1} = \mathbf{x}_n - [Hf(\mathbf{x}_n)]^{-1}\nabla f(\mathbf{x}_n),\ n \geq 0.$$

Usually Newton's method is modified to include a small step size γ > 0 instead of γ = 1

$$\boxed{\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma[Hf(\mathbf{x}_n)]^{-1}\nabla f(\mathbf{x}_n).}$$

Application Of The Algorithms In The Project

The robot is placed in an arbitrary position in an arbitrary point on the arena. The robot has a rotating sensor on its front which takes a circle of radius 30mm in the case of gradient descent and radius 58mm in the Newton's algorithm. The robot has to calculate the gradient in the case of the gradient descent and calculate the jacobian and the hessian in the Newton's algorithm at this point.

The robot initially takes the reading at the point at which it is placed. It moves through the distance of the radius in order to takes the readings over the circumference of a small circle around the point. Once the angle of descent is determined the robot moves in the direction of descent for a predetermined period of time. During its descent the robot simultaneously takes the sensor values. The sensor values versus the angle of rotation of the wheels are plotted in a graph and a best fit parabola is chosen. The robot then moves to the minimum of the parabola. This process is repeated until the condition for minimum is reached.

Description of the variables

x – It is the x coordinate with respect to the robot along the axis of the robot

y – It is the y coordinate with respect to the robot perpendicular to the axis of the robot

f(x,y) – It is a function describing the degree of darkness or brightness of the arena at any point. Its value is directly obtained from the sensor readings.

f_0 – The value of the center of the circle the point at which the values are determined

Θ – The angle read in the anticlockwise direction wrt the x-axis.

df – Small change in f

dx – Small change in x

dy – Small change in y

∂f/∂x – Gradient along x direction

∂f/∂y – Gradient along y direction

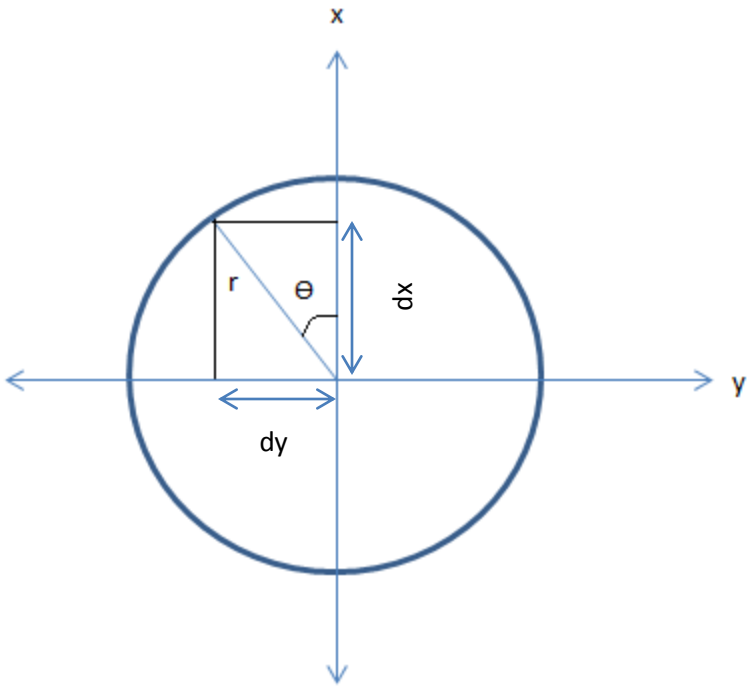

Fig9. Diagram explaining the variables and coordinates chosen

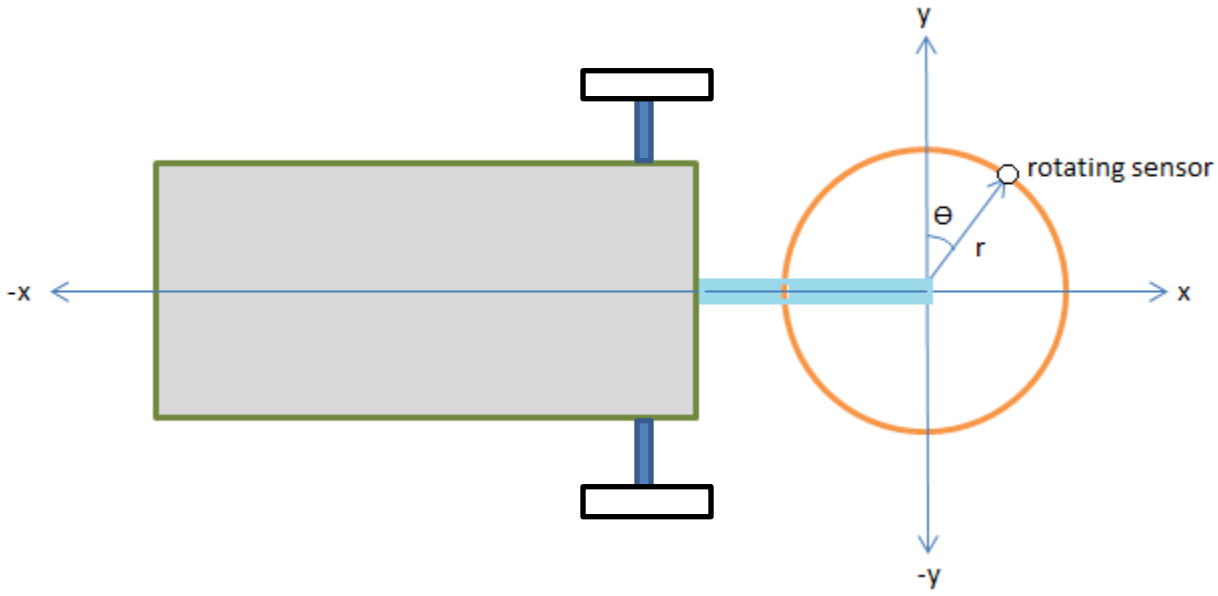

Fig10.Pictorial representation of the robot with its axes

Equations For Steepest Descent:

From figure 8, we find that

$dx = r*\cos\Theta$ ⟶ e 1

$dy = r*\sin\Theta$ ⟶ e2

f is a function of x and y and in steepest descent method we neglect the effects of second and higher order terms in the taylor's expansion.

$df(x,y) = (\partial f/\partial x)*dx + (\partial f/\partial y)*dy$ ⟶ e3

Substituting equations e1 and e2 in equation e3.

$df(x,y) =((\partial f/\partial x)*r*\cos\Theta)+ ((\partial f/\partial y)*r*\sin\Theta)$ ⟶ e4

In the equation e4 'r' is known as the radius the circle over which the readings are taken is fixed (30mm). Since the robot's motor has tachometer fixed in it we can find out the angle corresponding to each value of the intensity, therefore 'Θ' is known. Since we know the values of the function f at the center (f_0) and at the circumference at several points we can find out df at all these points given by

$$(df)_i = f_i - f_0$$

Where, f_i – It is the functional value at any point i

$(df)_I$ - It is the difference in the function at the point i

So the only unknowns are the gradients which need to find the angle of steepest descent

1) $\partial f/\partial x$
2) $\partial f/\partial y$

We have two unknowns and since we are taking continuous readings over a circle we over 200 equations to solve them. So the unknowns are over constrained and there are several redundant equations. Hence we use the method of least squares to find out the best suited values for the unknowns. This is done using Matlab.

The df values are taken as a column vector 'B' containing n elements where n is the number of readings taken. The unknowns that are the gradients are also taken as a column vector 'X' containing two elements. A n*2 matrix 'A' containing the 'n' 'dx' and 'dy' values is taken.

$$X = \begin{pmatrix} \partial f/\partial x \\ \partial f/\partial y \end{pmatrix}_{2*1}$$

$$B = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ . \\ . \end{pmatrix}_{n*1}$$

$$A = \begin{pmatrix} r*\cos\Theta_1 & r*\sin\Theta_1 \\ r*\cos\Theta_2 & r*\sin\Theta_2 \\ r*\cos\Theta_3 & r*\sin\Theta_3 \\ . & . \\ . & . \end{pmatrix}_{n*2}$$

The equation e4 can be rewritten in matrix form as

$$AX=B$$

$$X=A^{-1}B$$

But the equation is over constrained so we find out by the method of least squares. In Matlab it is given by

$$X=A\backslash B$$

So we find out the value of

1) $\partial f/\partial x$
2) $\partial f/\partial y$

The angle of steepest descent theta is given by the equation

$$\text{theta} = \tan^{-1}((\partial f/\partial x)/(\partial f/\partial y))$$

Equations For Newton's Method:

From figure 8, we find that

$dx = r*\cos\Theta$ ⟶ e 1

$dy = r*\sin\Theta$ ⟶ e2

Newton's method also takes into account the second order terms of the Taylor's series. The second order Taylor expansion of *f*(*x*) around *x*,

$$f(x+\Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$$

Since f is a function of x and y

$$df(x,y) = \begin{pmatrix} (\partial f/\partial x) & (\partial f/\partial y) \end{pmatrix} \begin{pmatrix} dx \\ dy \end{pmatrix} + (½) * \begin{pmatrix} dx & dy \end{pmatrix} * \begin{pmatrix} (\partial^2 f/\partial x^2) & (\partial^2 f/\partial x dy) \\ (\partial^2 f/\partial x dy) & (\partial^2 f/\partial y^2) \end{pmatrix} * \begin{pmatrix} dx \\ dy \end{pmatrix}$$

$$df(x,y) = ((\partial f/\partial x)*dx) + ((\partial f/\partial y)*dy) + (½)*(\partial^2 f/\partial x^2)*(dx)^2 + (\partial^2 f/\partial x dy)*dx*dy + (½)*(\partial^2 f/\partial y^2)*(dy)^2$$

⟶ e3

In the equation e3, we can substitute dx as r*cos Θ and dy as r*sin Θ according to the equations e1 and e2. So in the resulting equation 'r' is known as the radius the circle over which the readings are taken is fixed (30mm). Since the robot's motor has tachometer fixed in it we can find out the angle corresponding to each value of the intensity, therefore 'Θ' is known. Since we know the values of the function f at the center (f_0) and at the circumference at several points we can find out df at all these points given by

$$(df)_i = f_i - f_0$$

Where, f_i – It is the functional value at any point i

$(df)_I$ - It is the difference in the function at the point i

So the unknowns are the jacobian and the hessian which be used to find the minimum of the function using Newton's method.

1) $\partial f/\partial x$
2) $\partial f/\partial y$
3) $(\partial^2 f/\partial x^2)$
4) $(\partial^2 f/\partial x dy)$
5) $(\partial^2 f/\partial y^2)$

We have five unknowns and since we are taking continuous readings over a circle we over 200 equations to solve them. So the unknowns are over constrained and there are several redundant equations. Hence we use the method of least squares to find out the best suited values for the unknowns. This is done using Matlab.

The df values are taken as a column vector 'B' containing n elements where n is the number of readings taken. The unknowns that are the gradients are also taken as a column vector 'X' containing two elements. A n*5 matrix 'A' containing the 'n' 'dx', 'dy', '$(dx)^2/2$','dx*dy' and '$(dx)^2/2$' values is taken.

$$X = \begin{pmatrix} \partial f/\partial x \\ \partial f/\partial y \end{pmatrix}_{2*1} \qquad B = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ . \\ . \end{pmatrix}_{n*1}$$

$$A = \begin{pmatrix} dx_1 & dy_1 & ((dx_1)^2/2) & dx_1*dy_1 & ((dy_1)^2/2) \\ dx_2 & dy_2 & ((dx_2)^2/2) & dx_2*dy_2 & ((dy_2)^2/2) \\ dx_3 & dy_3 & ((dx_3)^2/2) & dx_3*dy_3 & ((dy_3)^2/2) \\ . & . & . & . & . \\ . & . & . & . & . \end{pmatrix}_{n*5}$$

The equation e3 can be rewritten in matrix form as

$$AX=B$$

$$X=A^{-1}B$$

But the equation is over constrained so we find out by the method of least squares. In Matlab it is given by

$$X=A\backslash B$$

So we find out the value of

1) $\partial f/\partial x$
2) $\partial f/\partial y$
3) $(\partial^2 f/\partial x^2)$
4) $(\partial^2 f/\partial x dy)$
5) $(\partial^2 f/\partial y^2)$

Hence the jacobian is given by

$$J = \begin{pmatrix} \partial f/\partial x & \partial f/\partial y \end{pmatrix} \begin{pmatrix} dx \\ dy \end{pmatrix}$$

and the hessian matrix is given by

$$H = \begin{pmatrix} (\partial^2 f/\partial x^2) & (\partial^2 f/\partial x dy) \\ (\partial^2 f/\partial x dy) & (\partial^2 f/\partial y^2) \end{pmatrix}$$

With these matrices the minimum point can be found out by the following equation given by Newton's method.

$$\begin{pmatrix} \Delta x & \Delta y \end{pmatrix} = - \begin{pmatrix} \partial f/\partial x & \partial f/\partial y \end{pmatrix} \begin{pmatrix} (\partial^2 f/\partial x^2) & (\partial^2 f/\partial x dy) \\ (\partial^2 f/\partial x dy) & (\partial^2 f/\partial y^2) \end{pmatrix}$$

Where $\begin{pmatrix} \Delta x & \Delta y \end{pmatrix}$ gives the distance to move along the x axis and the y axis to reach the minimum point.

We can make the robot to rotate through the required angle and then move forward by finding the angle at which the minimum point is present wrt the robot. The angle theta is given by

$$\text{theta} = \tan^{-1}(\Delta y / \Delta x)$$

The distance it has to move forward (step length) 'r' after it has rotated through angle theta is obtained the following equation

$$r = \sqrt{(\Delta x^2 + \Delta y^2)}$$

Though this method gives the step length directly it has been in this project the accuracy of the step length is very poor but the angle obtained by this method is very excellent.

Determination Of Step Length:

Once the angle of descent is estimated and the robot is oriented in the descent direction the robot moves forward through a particular distance called the step length which is determined by the following technique. The robot moves in the direction of descent for a predetermined period of time. During its descent the robot simultaneously takes the sensor values. Using the tachometers in the motors connected to the wheels of the robot the angle through which the motors have rotated at every instant is determined. The sensor values versus the angle of rotation of the wheels are plotted in a graph and a best fit parabola is chosen. Let the parabola be

$$f = ax^2+bx+c$$

where,

f- function indicating the darkness of a point got directly from the sensor
x – the tachometer value of the motors connected to the wheels in degrees
a,b,c – Constants

The best fit parabola can be one of the following types

1) Parabola with a proper minimum
2) Parabola with a proper maximum
3) Parabola which is almost a straight line with a negative slope
4) Parabola which is almost a straight line with a positive slope

1) Parabola with a proper minimum

In this case a proper minimum is present in the parabola. In the function

$$f = ax^2+bx+c$$

The value of 'a' is positive and is sufficiently large so that the parabola cannot be considered as a straight line. In this case the minimum of the parabola is found out by the following equation

$$df/dx = 0$$

A graph obtained during the test has been given below.

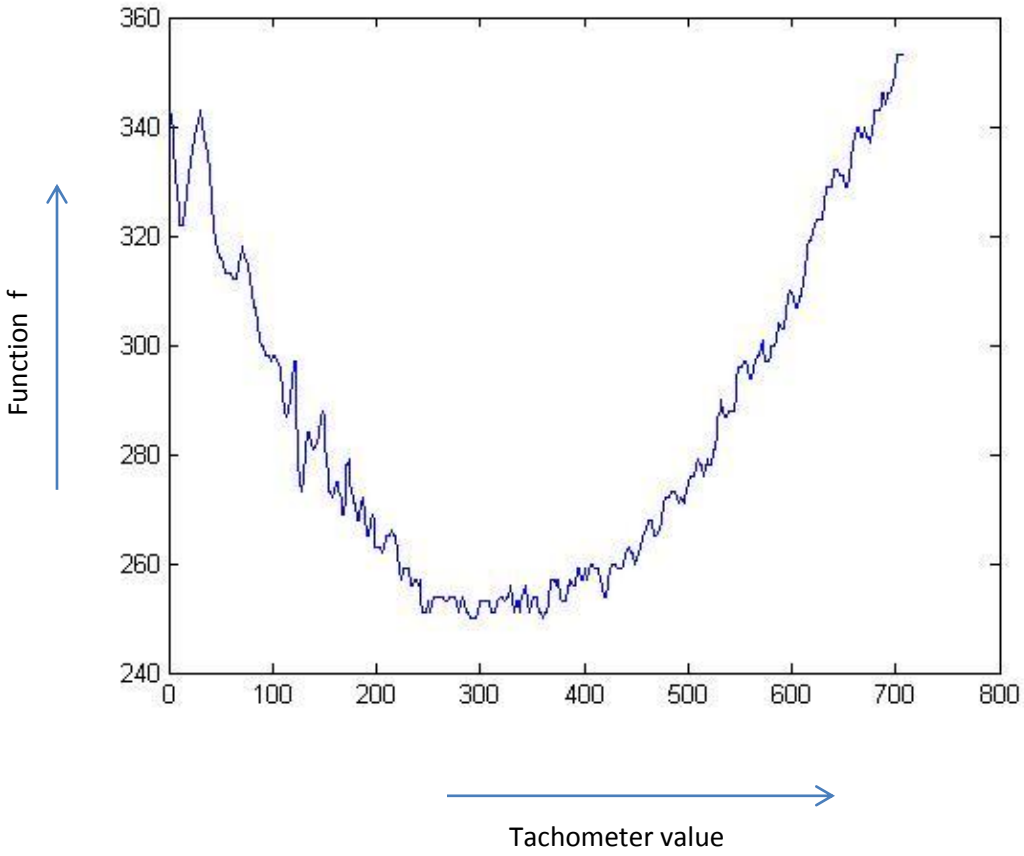

Fig11.Graph of intensity function f versus tachometer value x with minimum

Once the point of minimum is found the robot moves back to the point.

2) Parabola with a proper maximum

In this case a proper maximum is present in the parabola. In the function

$$f = ax^2+bx+c$$

The value of 'a' is positive and is sufficiently large so that the parabola cannot be considered as a straight line. A graph obtained during the test has been given below.

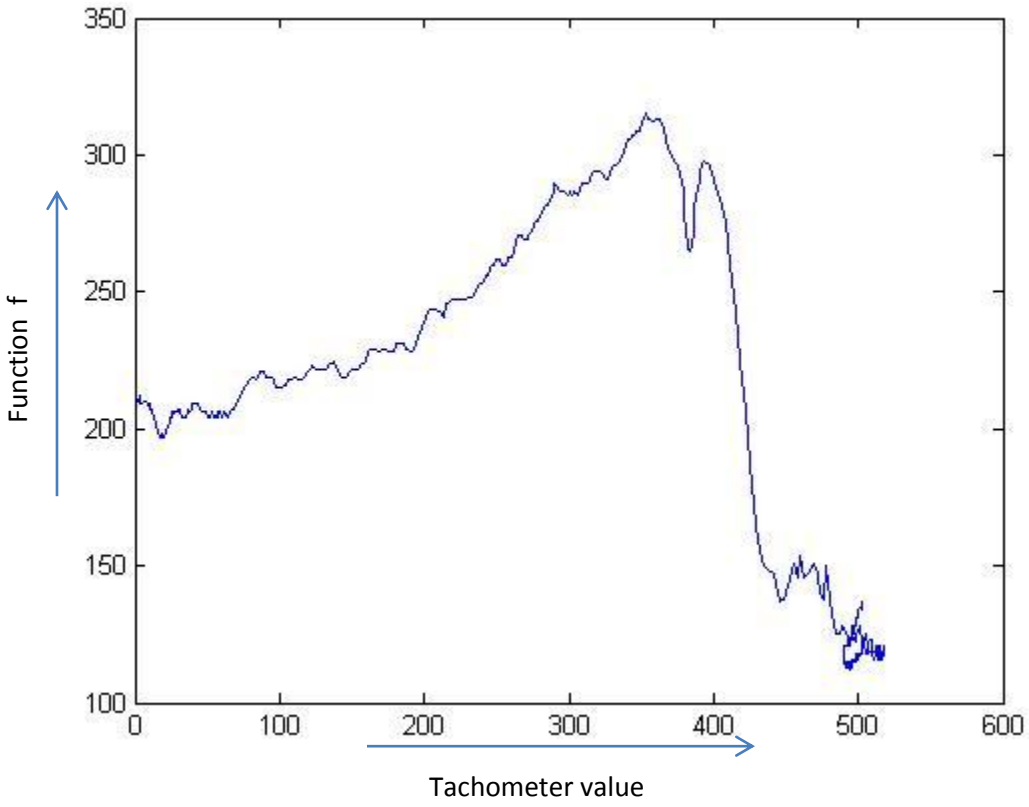

Fig12.Graph of intensity function f versus tachometer value x with maximum

In this case robot turns 180 degrees and moves to the same point at which the readings over the circle were taken.

3) Parabola which is almost a straight line with a negative slope

In this case the assumption that function f is a parabola returns poor results. In the function f

$$f = ax^2+bx+c$$

a is very close to zero that the approximation of a parabola is not valid. In this case an average slope of the function is determined by the following equation

$$\text{Average Slope} = (\sum 2ax_i+b)/n$$

Where,

∑ runs from 1 to n

x_i - Is the value of the tachometer at point i

a,b – Are the coefficients of the function f

A graph obtained during the test has been given below.

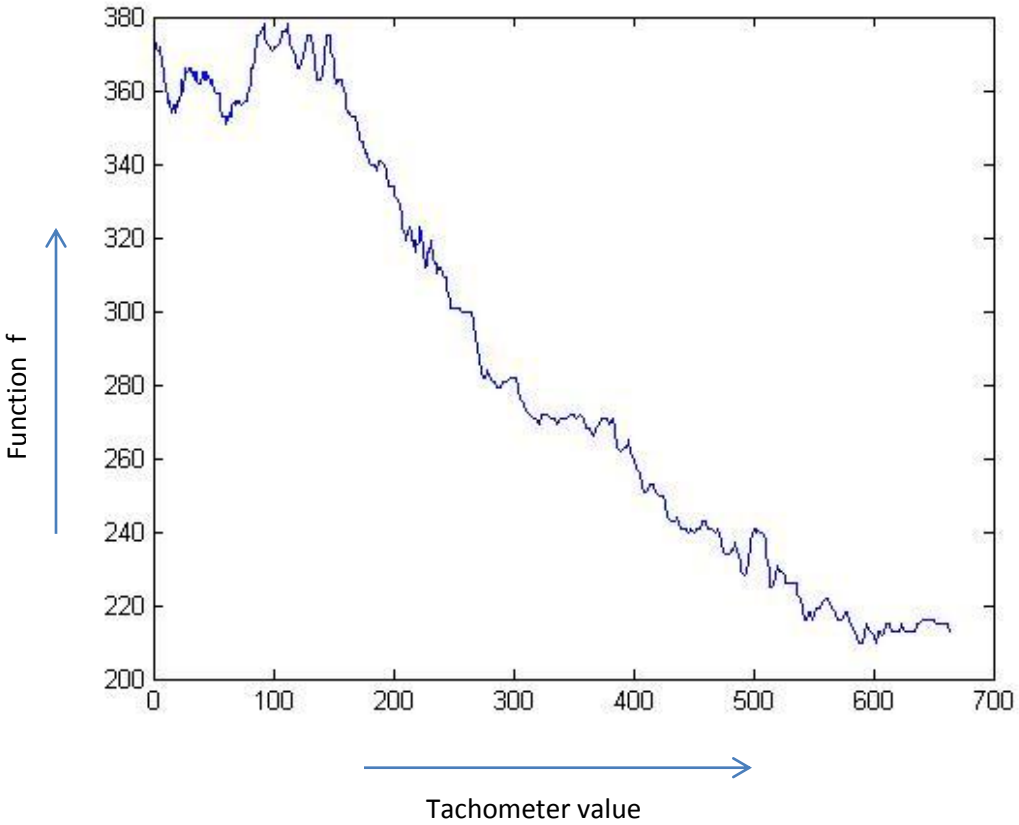

Fig13. Graph of intensity function f versus tachometer value which is a straight line with negative slope

If the slope obtained is negative the robot moves to the point with the minimum intensity.

4) Parabola which is almost a straight line with a positive slope

In this case the assumption that function f is a parabola returns poor results. In the function f

$$f = ax^2+bx+c$$

a is very close to zero that the approximation of a parabola is not valid. In this case an average slope of the function is determined by the method explained in the previous case.

If the slope obtained is positive the robot turns 180 degrees and moves to the same point at which the readings over the circle were taken. A graph obtained during the test has been given below.

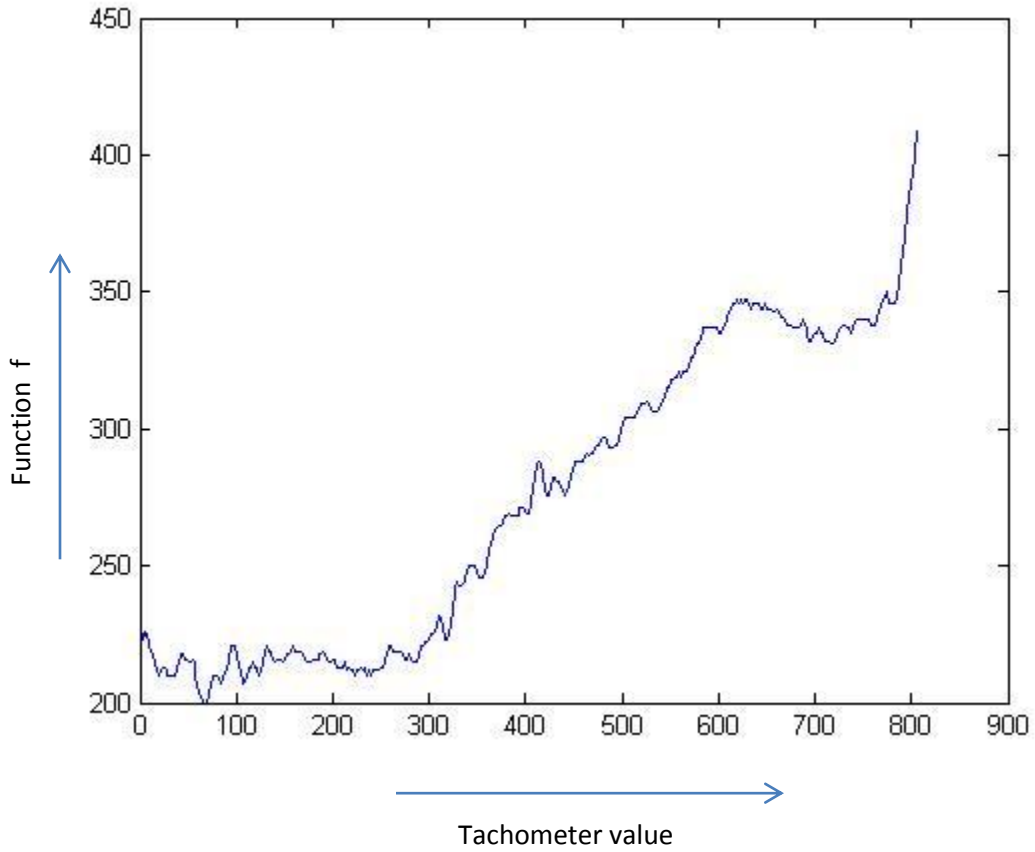

Fig14. Graph of intensity function f versus tachometer value
which is a straight line with positive slope

Program Codes

All the programs were built on **Matlab** using **RWTH - Mindstorms NXT Toolbox**. The following program codes are included in this project report.

1) Steep descent method using a nxc light sensor with an appropriate step length function
2) Steep descent method using a nxc light sensor with noise cancellation
3) Steep descent using a nxt color sensor using different colored light sources
4) Newton's Method with an appropriate step length function

1) Steep descent method using a nxc light sensor

In this program the robot takes readings over a small circle around the point at which the sensor of the robot was initially placed. After the calculation of the angle of steepest descent robot rotates about itself to the angle. Rotation consists of 3 steps

1) Moving the wheels of the robot forward to the point at which the sensor had taken the readings.
2) Turning the robot about itself to the required angle by giving equal and opposite powers to the wheels.
3) Moving the wheels of the robot back so that the sensor gets back to the point at which it had taken the readings.

This process is repeated three times to reduce the error in calculation of steepest descent. Then the function funcstep is called for the step length. The robot moves in the direction of descent for a predetermined period of time. During its descent the robot simultaneously takes the sensor values. And the minimum point is determined by the method explained in the above section "Determination Of Step Length" in page 18. Then the robot again calculates the value of steepest descent in the new point in this process continues till the condition for the minimum is reached.

The condition for the minimum consists of two parts

- The variance should be a below a particular value.
- The average value of the light sensor readings over the circle made by the rotating sensor should be below a particular value.

These values depend on the ambient light in the room, the nature of the arena and the construction of the robot as well. In the arena chosen for the project the darkness of the sheet near the center (the minimum) is almost a constant. Hence the variance in the values over the circle taken near the center should be small. The second condition says the average sensor value at the center should be very small which is quite obvious as the sheet is the darkest at the center.

The program consists of two parts, the main program and also a separate function **funstep()** for the step length of the robot whose coding is given below the coding of the main program.

Main Program:

```
%%                    Steepest Descent
%                    -------------------
%
%This is a program for a lego mindstorms nxt robot. This program is to
%explain the concept of steepest descent and modelling the optimization
%technique using a robot.The robot is placed in an arena having a contour
%of any function.The functional value at a point is taken as the brightness
%or darkness of the arena at the same point. Here the robot has to minimize
%the function so it has to travel to the darkest point. The bot initially
%takes values over a small region(circle of radius 3cm) and calculates the
%direction of steepest descent and the step size (the forward travel in
%descent direction) is decided by the function funcstep whose explanation
%is given in its documentation.
%%

while true
    COM_CloseNXT all;% Closing the previous connection
    close all;
    clear all;


    for rotn=1:3 %To check the angle 3 times before moving forward


        h = COM_OpenNXT();
        %Opens a new connection to an NXT on a USB port
        COM_SetDefaultNXT(h);
        % Setting h as the default handle for the functions below
        tp = 460;
        % Tachometer value which has to be given to one motor so that
        %the robot rotates by 90 degrees(Calibration for turning the robot)
        OpenLight(SENSOR_3,'ACTIVE'); % making the light sensor active
        flag_dir=0;% Flag to set clockwise or anticlockwise rotation


        pos=zeros(200,1);%gets position of rotating sensor at each point
        L = zeros(200,1);%gets value of rotating light sensor at the points
        b = zeros(200,1);%contains the change in intensity values
        A = zeros(200,2);%The change in positonal values of x and y
        % axis dx and dy
        x = zeros(2,1);% gradient values


        i=1;% Iteration variable
        L(i) = GetLight(Sensor_3);% Gettin the value of the sensor at
```

```
        %the middle of the circle

 %% Moving the distance of radius to take reading along the circle
        mfor = NXTMotor('AB','Power', 40);
        mfor.SpeedRegulation = false ;
        mfor.TachoLimit =65;
        mfor.ActionAtTachoLimit = 'Brake' ;
        mfor.SendToNXT();
        mfor.WaitFor();

%% Making the front sensor rotate by 360 degrees in anticlockwise

        mc=NXTMotor('C','Power',20);
        mc.SpeedRegulation = false ;
        mc.TachoLimit = 357;% Setting value 357 to avoid overshooting
        mc.ActionAtTachoLimit = 'Brake' ;
        mc.ResetPosition();
        mc.SendToNXT();


        OpenLight(SENSOR_3,'ACTIVE');% making the light sensor active

        while pos(i)<357
            i=i+1;
            L(i) = GetLight(SENSOR_3);% Getting sensor value at each point
            data1 = mc.ReadFromNXT();
            pos(i)= data1.Position;% Getting tachometer value at each point
        end
        mc.Stop('brake');
        StopMotor('all','off');
        %plot(pos,L);

 %% Making the wheels move to the position of the sensor so that the bot
 %  rotates about itself

        ms=NXTMotor('AB','Power',50);
        ms.SpeedRegulation = false ;
        ms.TachoLimit = 250;
        ms.ActionAtTachoLimit = 'Brake' ;
        ms.SendToNXT();
        ms.WaitFor();
        StopMotor('all','off');

%% Finding theta the direction of steepest descent

        for j = 2:i

            b(j-1) = L(j) - L(1); % Finding the difference in light sensor
            %values between the points on the circle and the centre
            A(j-1,1) = cosd(pos(j));%x coordinate value x = r*cos(theta)
            A(j-1,2) = sind(pos(j));%y coordinate value y = r*sin(theta)
        end

        x = A\b; %Finding the values of the gradients by the
```

```
        %least squares method
        xl=-x;
        xl(2)=xl(2)*sqrt(-1);
        xz=xl(1)+xl(2);
        theta = angle(xz); % Finding the angle of steepest descent
        theta=rad2deg(theta);

        if( theta < 0 )
            theta=abs(theta);
            flag_dir=1;% If theta is negative rotate clockwise
        end

%% Condition for stopping
        y=var(L(:));% Finding the variance of light intensity in the region
        tot=sum(L);
        sz=size(L);
        avg=tot/sz(1);
        %Finding the average of light intensity in the region

        if ( y<150) && avg<270 % If intensity is less and variance also
        % less the robot is at the minimum of the function so stop.

            NXT_PlayTone(300,200);pause(1);
            NXT_PlayTone(700,200);pause(1);
            NXT_PlayTone(1000,200);
            break;
        else


%% Making the robot rotate through the angle

            a = round((tp/90 * theta)/2); % Finding the tachometer value
            %required for taking the turn
            t = abs(a);

            if flag_dir==0 % Rotating anticlockwise
                NXC_MotorControl(0, 50, t, false, 'Brake', false);
                NXC_MotorControl(1, -50, t, false, 'Brake', false);
                pause(3)
                % One motor is rotated in clockwise sense and the other in
                % the anticlockwise sense so that the bot rotates about
                % itself

            elseif flag_dir==1 % Rotating anticlockwise
                NXC_MotorControl(0, -50, t, false, 'Brake', false);
                NXC_MotorControl(1, 50, t, false, 'Brake', false);
                pause(3)

            end

%% Bringing the sensor to the same point by moving backwards

            mbk=NXTMotor('AB','Power',-50);
            mbk.SpeedRegulation = false ;
```

```
            mbk.TachoLimit = 250;
            mbk.ActionAtTachoLimit = 'Brake' ;
            mbk.SendToNXT();
            mbk.WaitFor();
            StopMotor('all', 'off');
            portl= SENSOR_3;
        end

        if( y<150)&& avg<270
            NXT_PlayTone(300,200);pause(1);
            NXT_PlayTone(700,200);pause(1);
            NXT_PlayTone(1000,200);
            break;
        end
    end % End of rotations

%% Checking again

    if( y<150) && avg<270
        NXT_PlayTone(300,200);pause(1);
        NXT_PlayTone(700,200);pause(1);
        NXT_PlayTone(1000,200);
        break;
    end

%% Function for step length

funcstep();

%% Closing the sensor
    CloseSensor(SENSOR_3);

end
```

The Step Length Function:

```
%%                    STEP LENGTH FUNCTION
%                    ----------------------
%This is the function to find out the exact step length the robot should
%take to get to the minimum. Once the angle of steepest descent is found
%out and the robot is aligned in the position, we make the robot to move
%through some arbitrarily chosen period of time. When the robot moves it
%simultaneously gets the light sensor values . The values of the positon
%are taken from the tachometer . The values of the light sensor vs
%tachometer reaading are obtained . A best fit parabola is taken by the
%method of least squares and the robot moves to the minimum of the
%parabola. If the robots incurs a maximum it takes a U-turn and goes back
%to the original position.
%%                           Function

function [] = funcstep()
```

```
COM_CloseNXT all; % Closing the previous connection
close all;
clear all;

h = COM_OpenNXT(); % Opens a new connection to an NXT on a USB port
COM_SetDefaultNXT(h);
% Setting h as the default handle for the functions below

portl= SENSOR_3;
OpenLight(portl, 'ACTIVE');% making the light sensor active

A=zeros(400,3);% Array containing the coefficients of the quadratic
%function of intensity versus tachometer value
int_strt = zeros(1,400);%Gets the value of light sensor at each point
dist = zeros(1,400);%gets position of motors connected to the
%wheels at each point
n=400;%No of iterations for which the motor moves

%% Making the robot move forward

mab=NXTMotor('AB','Power', 30);
mab.SpeedRegulation = false ;
mab.TachoLimit = 0;
mab.ActionAtTachoLimit = 'Brake' ;
mab.ResetPosition();
mab.SendToNXT();

for it = 1:n
    int_strt(it)= GetLight(portl);% Getting sensor value at each point
    data=mab.ReadFromNXT();
    dist(it)= data.Position;% Getting tachometer value at each point

end

mab.Stop('brake');
mab.ResetPosition();

plot(dist,int_strt);% Plotting the graph of the intensity of light
%versus the tachometer positon at every point

%% Calculating the minimum by least squares method

for it=1:400
    A(it,1)=dist(it)^2;
    A(it,2)=dist(it);
    A(it,3)=1;
end
Y=zeros(3,1); % Y has the values of the coefficients of the quadratic
B=int_strt';
Y=A\B;

syms x ;
f = Y(1)*x^2+Y(2)*x+Y(3); % the quadratic function of the intensity value
```

```
%wrt tachometer value
f1 = diff(f);
f2 = diff(f1);
array=(2*Y(1)*dist)+Y(2);
total=sum(array);
length=size(dist);
f1d=total/length(2);%Average Slope
f2d=double(f2);%Second differentation value

%% Finding out the distance it has to move back.
%First we check whether the function has sufficient curvature by checking
%the coefficient of X^2 term of the quadratic. If its sufficiently large
%we can find out the minimum. We next check whether the function has a
%maximum or a minimum. If the function has a maximum we take a U-turn and
%come back to the original position . If the function has a minimum we find
% out the tachometer value at the minimum. If the coefficient of X^2 is too
% small we assume its a straight line and find the slope . If the slope
% is positive we go back to the starting point or if the slope is negative we
% we goto the point of minimum intensity.
%%


if abs(f2d)>=0.001% Checking whether curvature is large enough

    NXT_PlayTone(600,100);pause(1);
    NXT_PlayTone(440,100);

    if f2d>0 % Whether it is a minimum
        dist_go = solve(f1); % The minimum of the function
        dist_back=dist(n-1)-dist_go; % The distance the robot has to move
        % back is the distance the robot has moved forward subtracted by
        % the distance of the minimum

    elseif f2d < 0 % Whether is it a maximum
        tp1 = 460;
        back1=double(dist(n));

        % Going back to the original position
        mab_back1 = NXTMotor('AB','Power', -40);
        mab_back1.SpeedRegulation = false ;
        mab_back1.TachoLimit = back1;
        mab_back1.ActionAtTachoLimit = 'Brake' ;
        mab_back1.SendToNXT();
        mab_back1.WaitFor();

        %Making a U turn

        theta1=180;
        a1 = round((tp1/90 * theta1)/2);
        t1 = abs(a1);
        NXC_MotorControl(0, 40, t1, false, 'Brake', false);
        NXC_MotorControl(1, -40, t1, false, 'Brake', false);

        %Making the bot move back so that the sensor comes to the original
        %position
```

```
        mbk1=NXTMotor('AB','Power',-40);
        mbk1.SpeedRegulation = false ;
        mbk1.TachoLimit = 575;
        mbk1.ActionAtTachoLimit = 'Brake' ;
        mbk1.SendToNXT();
        mbk1.WaitFor();
        dist_back=1;


    end
else %If the function is a straight line

    NXT_PlayTone(440,100);pause(1);
    NXT_PlayTone(300,100);pause(1);
    NXT_PlayTone(500,100);

    if f1d>0 % If the slope is positive that is the function is increasing
    % the robot is made to take a U-turn

        tp1 = 460;
        back1=double(dist(n-1));
        mab_back1 = NXTMotor('AB','Power', -40);
        mab_back1.SpeedRegulation = false ;
        mab_back1.TachoLimit = back1;
        mab_back1.ActionAtTachoLimit = 'Brake' ;
        mab_back1.SendToNXT();
        mab_back1.WaitFor();

        theta1=180;
        a1 = round((tp1/90 * theta1)/2);
        t1 = abs(a1);

        NXC_MotorControl(0, 40, t1, false, 'Brake', false);
        NXC_MotorControl(1, -40, t1, false, 'Brake', false);
        mbk2=NXTMotor('AB','Power',-100);
        mbk2.SpeedRegulation = false ;
        mbk2.TachoLimit = 1;
        mbk2.ActionAtTachoLimit = 'Brake' ;
        mbk2.SendToNXT();
        mbk2.WaitFor();


        mbk1=NXTMotor('AB','Power',-40);
        mbk1.SpeedRegulation = false ;
        mbk1.TachoLimit = 575;
        mbk1.ActionAtTachoLimit = 'Brake' ;
        mbk1.SendToNXT();
        mbk1.WaitFor();
        dist_back=1;

    elseif f1d<=0 % If the slope is negative it goes to the minimum
        %intensity point
        int_min=min(int_strt);
        for iter3 = 1:n
```

```
            if int_strt(iter3)==int_min
                dist_go=dist(iter3);
                dist_back=dist(n-1)-dist_go+1;
            end
        end
    end
end
back=double(dist_back);
back=round(back);
if back>dist(400)%Precautionary statements to avoid the bot from going
    %beyond the initial point
    back=dist(400)+10;
end

if back==0%Precautionary statements to avoid the bot from going
    %beyond the initial point
    back=dist(400);
end
if back<0
    back=1;
end

%% Making the bot move back to appropriate point

mab_back = NXTMotor('AB','Power', -40);
mab_back.SpeedRegulation = false ;
mab_back.TachoLimit = back;
mab_back.ActionAtTachoLimit = 'Brake' ;
mab_back.SendToNXT();
mab_back.WaitFor();
end
```

OBSERVATIONS

- The robot moves in the descent direction in almost all occasions.
- If the angle of steepest descent is obtuse, the robot moves in the descent direction but the angle of descent is poorly estimated. For example, if the correct angle of descent was 160 degrees, the angle estimated maybe 125 degrees.
- When the angle of steepest descent is acute, the angle is estimated with acceptable accuracy.
- At some specific points of the arena the angle estimated are poor.
- The minimum is reached in most cases within four iterations.

CORRECTIONS TRIED

- Using an infra-red light source on the light sensor to reduce the effect of ambient light.
- A higher resolution of the arena that is a uniform gradient in the arena.
- Checking the angle of steepest descent at the same point more than once by orienting the robot at different angles to the same point before taking the step length.

RESULTS OBTAINED BY APPLYING THE CORRECTIONS

- The use infra-red light source reduced the errors by a small amount but it was not worth the extra effort
- A good resolution arena with a proper gradient error reduced the errors substantially and gave significantly better results.
- Checking the angle of steepest descent at the same point more than once gave a much better angle determination reduced the time taken to minimum significantly.

2) Steep descent method using a nxc light sensor with noise cancellation

This program is similar to the first program “Steep descent method using a nxc light sensor”. In this program, the robot takes readings over a small circle around the point at which the sensor of the robot was initially placed. The readings are first taken with the light in the light sensor in the active state (switched on) and then in the inactive state (switched off). The values obtained in the inactive state are subtracted from the values obtained in the active state in every point. This reduces the effect of ambient light disturbance.

After the calculation of the angle of steepest descent robot rotates about itself to the angle. This process is repeated three times to reduce the error in calculation of steepest descent. Then the function funcstep is called for the step length (moving in the descent direction) which is the same function used in the first program “Steep descent method using a nxc light sensor”, so it is not repeated in the report.

Main Program:

```
%%              Steepest Descent With Noise Cancellation
%                         -------------------
%
%This is a program for a lego mindstorms nxt robot. This program is to
%explain the concept of steepest descent and modelling the optimization
%technique using a robot.The robot is placed in an arena having a contour
%of any function.The functional value at a point is taken as the brightness
%or darkness of the arena at the same point. Here the robot has to minimize
%the function so it has to travel to the darkest point.
%
%The bot initially takes values over a small region(circle of radius 3cm)
%with the light in the nxc sensor in the active state (that is light is
%switched on) and then the values over the same circle with the light of
%the sensor in the inactive state ( the light is switched off ). We find
%the difference between the light intensities obtained at each point with
%the sensor in the active state and in the inactive state. With these
%values calculate the direction of steepest descent numerically and
%the step size (the forward travel in descent direction) is decided by
%the function  funcstep whose explanation is given in its documentation.
%%


while true
    COM_CloseNXT all;% Closing the previous connection
    close all;
    clear all;
```

```
    for rotn=1:3 %To check the angle 3 times before moving forward

        h = COM_OpenNXT(); %Opens a new connection to an NXT on a USB port
        COM_SetDefaultNXT(h);
        % Setting h as the default handle for the functions below
        tp = 460;
        % Tachometer value which has to be given to one motor so that
        %the robot rotates by 90 degrees(Calibration for turning the robot)

        OpenLight(SENSOR_3,'ACTIVE'); % making the light sensor active
        flag_dir=0;% Flag to set clockwise or anticlockwise rotation

        pos=zeros(200,1);%gets position of rotating sensor at each point
        L = zeros(200,1);%gets value of rotating light sensor at the points
        b = zeros(200,1);%contains the change in intensity values
        A = zeros(200,2);%The change in positonal values of x and y
        % axis dx and dy
        x = zeros(2,1);% gradient values

        i=1;% Iteration variable
        L(i) = GetLight(SENSOR_3);% Gettin the value of the sensor at
        %the middle of the circle

        %% Moving the distance of radius to take reading along the circle

        mx = NXTMotor('AB','Power', 30);
        mx.SpeedRegulation = false ;
        mx.TachoLimit =52;
        mx.ActionAtTachoLimit = 'Brake' ;
        mx.SendToNXT();
        mx.WaitFor();

        %% Making the front sensor rotate by 360 degrees in anticlockwise
with
        %light in the sensor switched on(active state)

        mc=NXTMotor('C','Power',20);
        mc.SpeedRegulation = false ;
        mc.TachoLimit = 357;
        mc.ActionAtTachoLimit = 'Brake' ;
        mc.ResetPosition();
        mc.SendToNXT();
        OpenLight(SENSOR_3,'ACTIVE');% making the light sensor active

        while pos(i)<357
            i=i+1;
            L(i) = GetLight(SENSOR_3);% Getting sensor value at each point
            data1 = mc.ReadFromNXT();
            pos(i)= data1.Position;% Getting tachometer value at each point
        end

        Lx=L;%Storing the orignal values in another variable
        mc.Stop('brake');
        StopMotor('all','off');
```

```
        %% Taking the values of the same circle with the light in the sensor
        %switched off(inactive state)

        pos1=zeros(100,1);
        L1 = zeros(100,1);
        i1=1;


        mxB = NXTMotor('AB','Power', -30);% Moving back to centre to get
        %the sensor value in the inactive state at the centre
        mxB.SpeedRegulation = false ;
        mxB.TachoLimit =52;
        mxB.ActionAtTachoLimit = 'Brake' ;
        mxB.SendToNXT();
        mxB.WaitFor();
        OpenLight(SENSOR_3,'INACTIVE');
        pause(2);
        L1(i1) = GetLight(SENSOR_3);


        %% Moving the distance of radius to take reading along the circle

        mxF = NXTMotor('AB','Power', 30);
        mxF.SpeedRegulation = false ;
        mxF.TachoLimit =52;
        mxF.ActionAtTachoLimit = 'Brake' ;
        mxF.SendToNXT();
        mxF.WaitFor();


        %% Making the front sensor rotate by 360 degrees in anticlockwise
with
        %light in the sensor switched off(inactive state)

        mc1=NXTMotor('C','Power',20);
        mc1.SpeedRegulation = false ;
        mc1.TachoLimit = 357;
        mc1.ActionAtTachoLimit = 'Brake' ;
        mc1.ResetPosition();
        mc1.SendToNXT();
        OpenLight(SENSOR_3,'INACTIVE');% making the light sensor inactive
        while pos1(i1)<357
            i1=i1+1;
            L1(i1) = GetLight(SENSOR_3);
            data2 = mc1.ReadFromNXT();
            pos1(i1)= data2.Position  ;
        end
        mc1.Stop('brake');
        StopMotor('all','off');

        %% Subtracting the sensor values in the inactive state with values in
the
        % active state to cancel off the noise effects. Since the values are
```

```
        % continously taken we cannot be sure that the same position of the
motor
        % is present in the same iterations of the inactive and active
readings. So
        % we check within -3 to +3 of the iterations to check if the
positions are
        % the same and then  find the difference.

        for j1 = 2:i
            if j1>3 && (j1+3<i1)
                for j2 = (j1-3):(j1+3)
                    if pos(j1)==pos1(j2)
                        L(j1)=L(j1)-L1(j2);
                        break;
                    end
                end
            elseif j1<=3 % If the iterant is less than 3 we cant use
                %iterant-3
                for j2 = 2:(j1+3)
                    if pos(j1)==pos1(j2)
                        L(j1)=L(j1)-L1(j2);
                        break;
                    end
                end

            elseif (j1+3)>i1 % If the iterant+3 is greater than the size of
                %the array
                for j2 = (j1-3):i1
                    if pos(j1)==pos1(j2)
                        L(j1)=L(j1)-L1(j2);
                        break;
                    end
                end
            end
        end



        %% Making the wheels move to the position of the sensor so that the
bot
        %  rotates about itself

        %plot(pos,L);
        ms=NXTMotor('AB','Power',50);
        ms.SpeedRegulation = false ;
        ms.TachoLimit = 250;
        ms.ActionAtTachoLimit = 'Brake' ;
        ms.SendToNXT();
        ms.WaitFor();
        StopMotor('all','off');

        L(1)=L(1)-L1(1);% Finding the difference at the centre

        %% Finding theta the direction of steepest descent
```

```
        for j = 2:i
            b(j-1) = L(j) - L(1); % Finding the difference in light sensor
            %values between the points on the circle and the centre
            A(j-1,1) = cosd(pos(j));%x coordinate value x = r*cos(theta)
            A(j-1,2) = sind(pos(j));%y coordinate value y = r*sin(theta)
        end

        x = A\b;%Finding the values of the gradients by the
        %least squares method
        xl=-x;
        xl(2)=xl(2)*sqrt(-1);
        xz=xl(1)+xl(2);
        theta = angle(xz); % Finding the angle of steepest descent
        theta=rad2deg(theta);

        if( theta < 0 )
            theta=abs(theta);
            flag_dir=1;% If theta is negative rotate clockwise
        end


        %% Condition for stopping
        y=var(Lx(:));% Finding the variance of light intensity in the region
        tot=sum(Lx);
        sz=size(Lx);
        avg=tot/sz(1);
        %Finding the average of light intensity in the region

        if ( y<150) && avg<270 % If intensity is less and variance also
            % less the robot is at the minimum of the function so stop.

            NXT_PlayTone(300,200);pause(1);
            NXT_PlayTone(700,200);pause(1);
            NXT_PlayTone(1000,200);
            break;

        else
%% Making the robot rotate through the angle

            a = round((tp/90 * theta)/2);
            t = abs(a);

             if flag_dir==0 % Rotating anticlockwise
                NXC_MotorControl(0, 50, t, false, 'Brake', false);
                NXC_MotorControl(1, -50, t, false, 'Brake', false);
                pause(3)
                % One motor is rotated in clockwise sense and the other in
                % the anticlockwise sense so that the bot rotates about
                % itself

            elseif flag_dir==1 % Rotating anticlockwise
                NXC_MotorControl(0, -50, t, false, 'Brake', false);
                NXC_MotorControl(1, 50, t, false, 'Brake', false);
                pause(3)
```

```
            end

%% Bringing the sensor to the same point by moving backwards

            mbk=NXTMotor('AB','Power',-50);
            mbk.SpeedRegulation = false ;
            mbk.TachoLimit = 250;
            mbk.ActionAtTachoLimit = 'Brake' ;
            mbk.SendToNXT();
            mbk.WaitFor();
            StopMotor('all', 'off');
            portl= SENSOR_3;
        end


        if( y<150)&& avg<270
            NXT_PlayTone(300,200);pause(1);
            NXT_PlayTone(700,200);pause(1);
            NXT_PlayTone(1000,200);
            break;
        end
    end% End of rotations

%% Checking again

    if( y<150) && avg<270
        NXT_PlayTone(300,200);pause(1);
        NXT_PlayTone(700,200);pause(1);
        NXT_PlayTone(1000,200);
        break;
    end

%% Function for step length

funcstepir2();

%% Closing the sensor
    CloseSensor(SENSOR_3);

end
```

RESULT OBTAINED

Though this method should logically give better results, the method did not give any additional accuracy in the estimated values.

3) Steep descent using a nxt color sensor using different colored light sources

This program is similar to the first program “Steep descent method using a nxc light sensor” but in this program the robot used an nxt color sensor rather than the nxc light sensor. In this program, the robot takes readings over a small circle around the point at which the sensor of the robot was initially placed. The readings over the circle are taken three times with a different colored light source in each rotation first with red, then blue and green colored lights. The average of the three angles of steepest descent estimated was taken. And then the robot rotates about itself to the angle. This process is repeated three times to reduce the error in calculation of steepest descent.

The step length function is same as the previous program but the functions used for the sensor is slightly different but the logic and the algorithms used are the same. So the step length function program is not included in the report.

Main Program:

```
%%       Steepest Descent With Different Colors For Sensing
%                     -------------------
%
%
%This is a program for a lego mindstorms nxt robot. This program is to
%explain the concept of steepest descent and modelling the optimization
%technique using a robot.The robot is placed in an arena having a contour
%of any function.The functional value at a point is taken as the brightness
%or darkness of the arena at the same point. Here the robot has to minimize
%the function so it has to travel to the darkest point.
%
%The bot initially takes values over a small region(circle of radius 3cm)
%with red , blue and green light emitted by the light source in the color
%sensor of the nxt robot and calculates the direction of steepest descent
%by taking the average of the angles obtained by using the different
%colors and the step size (the forward travel in descent direction)
%is decided by the function funcstep whose code is similar to the function
%using the nxc light sensor instead of the nxt color sensor.
%%                            Program




while true

    COM_CloseNXT all;% Closing the previous connection
    close all;
```

```
clear all;

color = {   'RED'; 'BLUE' ;'GREEN' };% One colour of the colour sensor
%for each rotation
y3=[0 0 0];% Variances for each colour in the small circle
theta3=[0 0 0];%Theta value for each colour in the small cirlce
avg3=[0 0 0 ];%Average intensity values for each colour in the circle



for rotna = 1:3 % Checks three times before moving forward

    for rotn=1:3 % One rotation for each colour

        h = COM_OpenNXT();
        %Opens a new connection to an NXT on a USB port
        COM_SetDefaultNXT(h);
        % Setting h as the default handle for the functions below
        tp = 460;
        OpenNXT2Color(Sensor_3,color(rotn));
        % making the colour sensor switch on with different colour in
        % each iteration

        flag_dir=0;% Flag to set clockwise or anticlockwise rotation
        pos=zeros(100,1);
        %gets position of rotating sensor at each point
        L = zeros(100,1);
        %gets value of rotating light sensor at the points
        b = zeros(100,1);
        %contains the change in intensity values
        A = zeros(100,2);%The change in positonal values of x and y
        % axis dx and dy
        x = zeros(2,1);% gradient values

        i=1;% Iteration variabl
        L(i) = GetNXT2Color(Sensor_3);% Gettin the value of the sensor
        %at the middle of the circle

%% Moving the distance of radius to take reading along the circle

        mx = NXTMotor('AB','Power', 30);
        mx.SpeedRegulation = false ;
        mx.TachoLimit =52;
        mx.ActionAtTachoLimit = 'Brake' ;
        mx.SendToNXT();
        mx.WaitFor();

%% Making the front sensor rotate by 360 degrees in anticlockwise

        mc=NXTMotor('C','Power',20);
        mc.SpeedRegulation = false ;
        mc.TachoLimit = 360;
        mc.ActionAtTachoLimit = 'Brake' ;
        mc.ResetPosition();
```

```
            mc.SendToNXT();

            while pos(i)<357
                i=i+1;
                L(i) = GetNXT2Color(SENSOR_3);
                % Getting sensor value at each point
                data1 = mc.ReadFromNXT();
                pos(i)= data1.Position  ;
                % Getting tachometer value at each point
            end
            mc.Stop('brake');
            StopMotor('all','off');

            %plot(pos,L);


%% Finding theta the direction of steepest descent

            for j = 2:i

                b(j-1) = L(j) - L(1);%Finding the difference in light
                %sensor values between the points on the circle and the
                %centre
                A(j-1,1) = cosd(pos(j));%x coordinate value x= r*cos(theta)
                A(j-1,2) = sind(pos(j));%y coordinate value y= r*sin(theta)
            end

            x = A\b;%Finding the values of the gradients by the
            %least squares method
            xl=-x;
            xl(2)=xl(2)*sqrt(-1);
            xz=xl(1)+xl(2);
            theta = angle(xz); % Finding the angle of steepest descent
            theta=rad2deg(theta);

            if( theta < 0 )
                theta=abs(theta);
                flag_dir=1;% If theta is negative rotate clockwise
            end
            theta3(rotn)=theta;% Storing thetas in array to take average

%% Condition for stopping

            y=var(L(:));
            % Finding the variance of light intensity in the region
            y3(rotn)=y;
            tot=sum(L);
            sz=size(L);
            avg=tot/sz(1);
            %Finding the average of light intensity in the region
            avg3(rotn)=avg;


            if ((y<80)&&(avg<270)&&(rotn==1))% If intensity is less and
```

```
                %variance also less the robot is at the minimum of the
function
                %so stop.

                NXT_PlayTone(300,200);pause(1);
                NXT_PlayTone(700,200);pause(1);
                NXT_PlayTone(1000,200);
                break;
            end

%% Coming back the distance of take readius to take the intensity reading
% at centre with different colour

            my = NXTMotor('AB','Power', -30);
            my.SpeedRegulation = false ;
            my.TachoLimit =52;
            my.ActionAtTachoLimit = 'Brake' ;
            my.SendToNXT();
            my.WaitFor();

        end

        theta_avg=sum(theta3)/3;
        % Taking average theta got by the three colours


%% Making the wheels move to the position of the sensor so that the bot
%  rotates about itself

        ms=NXTMotor('AB','Power',50);
        ms.SpeedRegulation = false ;
        ms.TachoLimit = 250;
        ms.ActionAtTachoLimit = 'Brake' ;
        ms.SendToNXT();
        ms.WaitFor();
        StopMotor('all','off');


        if ((y3(1)<80)&&(avg3(1)<270))% If intensity is less and
            %variance also less the robot is at the minimum of the function
            %so stop.

            NXT_PlayTone(300,200);pause(1);
            NXT_PlayTone(700,200);pause(1);
            NXT_PlayTone(1000,200);
            break;
        else
%% Making the robot rotate through the angle

            a = round((tp/90 * theta_avg)/2);% Finding the tachometer value
            %required for taking the turn
            t = abs(a);

            if flag_dir==0 % Rotating anticlockwise
```

```
                NXC_MotorControl(0, 40, t, false, 'Brake', false);
                NXC_MotorControl(1, -40, t, false, 'Brake', false);
                pause(3)
                % One motor is rotated in clockwise sense and the other in
                % the anticlockwise sense so that the bot rotates about
                % itself

            elseif flag_dir==1 % Rotating anticlockwise
                NXC_MotorControl(0, -40, t, false, 'Brake', false);
                NXC_MotorControl(1, 40, t, false, 'Brake', false);
                pause(3)

            end

%% Bringing the sensor to the same point by moving backwards

            mbk=NXTMotor('AB','Power',-50);
            mbk.SpeedRegulation = false ;
            mbk.TachoLimit = 250;
            mbk.ActionAtTachoLimit = 'Brake' ;
            mbk.SendToNXT();
            mbk.WaitFor();
            StopMotor('all', 'off');
            portl= SENSOR_3;
        end

        if ((y3(1)<80)&&(avg3(1)<270))% Checking again

            NXT_PlayTone(300,200);pause(1);
            NXT_PlayTone(700,200);pause(1);
            NXT_PlayTone(1000,200);
            break;
        end
    end

    if ((y3(1)<80)&&(avg3(1)<270))% Checking again

        NXT_PlayTone(300,200);pause(1);
        NXT_PlayTone(700,200);pause(1);
        NXT_PlayTone(1000,200);
        break;
    end


%% Function for step length

    funcstep2();

%% Closing the sensor

    CloseSensor(SENSOR_3);

end
```

RESULT OBTAINED

The result obtained by using the above method (program) were slightly better than the previous program. But the extra time it had taken to scan with three different colored light sources overshadowed the better estimation of the angles. So this method is not advised.

4) Newton's Method

In this program the robot takes readings over a small circle around the point at which the sensor of the robot was initially placed. The robot estimates the jacobian and the hessian matrices required to find the minimum point.

If the **hessian** is a **positive definite** that is if the eigen values of the hessian matrix are positive, then the estimated direction is the descent direction and its correct. But during the testing it was found out that at some positions on the arena the hessian estimated was not a positive definite and the estimated direction was the ascent direction. So the following technique was used to overcome the problem faced.

After the first estimation at a point, if the hessian is not a positive definite, the robot orients itself at 20 degrees to the left from its original position. The rotation is done in such a way that the sensor comes back to the same point (approximately). Again the hessian is estimated, if it turns out wrong the robot 20 degrees to the right from its original orientation (40 degrees in total) and the estimation is done again. The process is repeated with 40,60,80 degrees in either directions until a positive definite hessian is obtained.

Once the descent direction is correctly estimated (a positive definite hessian), the function funcstepn is called for the step length. The robot moves in the direction of descent for a predetermined period of time. During its descent the robot simultaneously takes the sensor values. And the minimum point is determined by the method explained in the above section "Determination Of Step Length" in page 18. Then the robot again calculates the value of steepest descent in the new point in this process continues till the condition for the minimum is reached.

In this program we use the Newton's method only for the determination of the angle at which the robot should move in order to reach the minimum and not the step length. Though the step length is easily estimated it was found that the estimated step length was inaccurate (theoretically it should be accurate) in the tests conducted.

Main Program:

```
%%                    Newton's Optimization method
%                    --------------------------------
%
%This is a program for a lego mindstorms nxt 2 robot. This program is to
%explain the concept of Newton's optimization method and modelling the
%optimization technique using a robot.The robot is placed in an arena
%having a contour of any function. The functional value at a point is taken
%as the brightness or darkness of the arena at the same point. Here the
%robot has to minimize the function so it has to travel to the darkest
%point. The bot initially takes values over a small region,
%circle of radius 58mm and calculates the jacobian and the hessian matrix
%to find out the minimum point. Due to several noise factors the robot is
%unable to find the minimum point accurately. So we use it find the angle
%at which the minimum occurs. Instead using the step size obtained by
Newton's method
%we decide the step size using the function funcstepn whose explanation
%is given in its documentation.


%%                    The Program


COM_CloseNXT all;% Closing the previous connection
close all;
clear all;
h = COM_OpenNXT(); %Opens a new connection to an NXT on a USB port
COM_SetDefaultNXT(h);
% Setting h as the default handle for the functions below
hna = zeros(5,1);% The matrix containing the unknowns determined

counter=0;% Counter which determines the angle through the robot will turn
%if the hessian matrix is not a positive definite
flag_forward=0;


%%

while true

    tp = 460;% The tachometer value given to a motor to make the bot turn
    %through 90 degrees (Calibration for turning the robot)
    OpenLight(SENSOR_3,'active'); % making the light sensor active

    flag_dir=0;% Flag to set clockwise or anticlockwise rotation in case of
    %positive definite hessian
    flag_dir1=0;% Flag to set clockwise or anticlockwise rotation in case of
    %hessian is not a positive definite
    pos=zeros(200,1);%gets position of rotating sensor at each point
    L = zeros(200,1);%gets value of rotating light sensor at the points
    b = zeros(200,1);%contains the change in intensity values
    A = zeros(200,5);%If the change in positonal values of x and y
```

```
        % axis are dx and dy the matrix contains
        % [dx        dy      ((dx^2)/2))     dx*dy     ((dy^2)/2)) ]
    hn = zeros(5,1);% jacobian and the hessian values

    x=zeros(1,2);%Minimum point coordinates

    %% Getting the value of the sensor at the middle of the circle

    i=1;% Iteration variable
    L(i) = GetLight(SENSOR_3);

    %% Moving the distance of the radius in order to take radings on the
    %circumference of the circle

    mx = NXTMotor('AB','Power', 40);
    mx.SpeedRegulation = false ;
    mx.TachoLimit =125;
    mx.ActionAtTachoLimit = 'Brake' ;
    mx.SendToNXT();
    mx.WaitFor();

    %% Making the front sensor rotate by 360 degrees in anticlockwise
    mc=NXTMotor('C','Power',20);
    mc.SpeedRegulation = false ;
    mc.TachoLimit = 357;% Setting value 357 to avoid overshooting
    mc.ActionAtTachoLimit = 'Brake' ;
    mc.ResetPosition();
    mc.SendToNXT();


    while pos(i)<357
        i=i+1;
        L(i) = GetLight(SENSOR_3);% Getting sensor value at each point
        data1 = mc.ReadFromNXT();
        pos(i)= data1.Position;% Getting tachometer value at each point
    end
    mc.Stop('brake');
    StopMotor('all','off');
    %plot(pos,L);

%% Making the wheels move to the position of the sensor so that the bot
 %     rotates about itself

    ms=NXTMotor('AB','Power',50);
    ms.SpeedRegulation = false ;
    ms.TachoLimit = 250;
    ms.ActionAtTachoLimit = 'Brake' ;
    ms.SendToNXT();
    ms.WaitFor();
    StopMotor('all','off');

%% Finding theta the direction of steepest descent

    for j = 2:i
```

```
        b(j-1) = L(j) - L(1); %% The difference in light sensor values
        %between the points on the circle and the centre
        A(j-1,1) = 58*cosd(pos(j)); %change in x value dx = r*cos(theta)
        A(j-1,2) = 58*sind(pos(j));%change in y value dy = r*sin(theta)
        A(j-1,3) = ((58*cosd(pos(j)))^2)/2;% (dx^2)/2
        A(j-1,4) = 58^2*sind(pos(j))*cos(pos(j));% dx*dy
        A(j-1,5) = ((58*sind(pos(j)))^2)/2;% (dy^2)/2
    end

    hn = A\b;% finding out the jacobian and the hessian matrix by least
    %squares method
    hna=[hna hn];
    jac =[ hn(1) hn(2)];% Jacobian matrix
    hess=([hn(3) hn(4);%Hessian matrix
        hn(4) hn(5)]);
    eigen=eig(hess);%Eigen values of the hessian matrix


    if(eigen(1)>0&&eigen(2)>0)
%% Checking whether the hessian is positive definite

        counter=0;% Every time hessian becomes positive the counter is
        %resetted to 0
        x=-jac/(hess);% The minimum point
        xl=x;
        xl(2)=xl(2)*sqrt(-1);
        xz=xl(1)+xl(2);
        r=abs(xz);%The step size determined by Newton's method

        theta = angle(xz);%Angle wrt to the bot at which the minimum is
        %present
        theta=rad2deg(theta);

        if( theta < 0 )
            theta=abs(theta);
            flag_dir=1; % If theta is negative rotate clockwise
        end
        flag_forward=0;


%% Condition for stopping

        y=var(L(:));% Finding the variance of light intensity in the region
        tot=sum(L);
        sz=size(L);
        avg=tot/sz(1);
        %Finding the average of light intensity in the region

        if (y<525) && avg < 270 % If intensity is less and variance also
        % less the robot is at the minimum of the function so stop.
            NXT_PlayTone(300,200);pause(1);
            NXT_PlayTone(700,200);pause(1);
            NXT_PlayTone(1000,200);
            break;
```

```
        else
%% Making the robot rotate through that angle
            pause(1)
            a = round((tp/90 * theta)/2); % Finding the tachometer value
            %required for taking the turn
            t = abs(a);

            if flag_dir==0 % Rotating anticlockwise
                NXC_MotorControl(0, 40, t, false, 'Brake', false);
                NXC_MotorControl(1, -40, t, false, 'Brake', false);
                pause(3)
                % One motor is rotated in clockwise sense and the other in
                % the anticlockwise sense so that the bot rotates about
                % itself

            elseif flag_dir==1 % Rotating anticlockwise
                NXC_MotorControl(0, -40, t, false, 'Brake', false);
                NXC_MotorControl(1, 40, t, false, 'Brake', false);
                pause(3)

            end
            StopMotor('all', 'off');

%% Bringing the sensor to the same point by moving backwards

            mbk=NXTMotor('AB','Power',-50);
            mbk.SpeedRegulation = false ;
            mbk.TachoLimit = 345;
            mbk.ActionAtTachoLimit = 'Brake' ;
            mbk.SendToNXT();
            mbk.WaitFor();
            StopMotor('all', 'off');
            portl= SENSOR_3;
        end


    else
 %% If the hessian matrix is not a positive definite. Since we have data
 %pertaining only to the region around the point we make the robot scan
 %the same point by being at different angles but the sensor being at the
 %same point. So we tilt the bot to angle of 20 degrees to the left and
 %find the hessian and if its still wrong we tilt it through 20 to the
 %right and find the hessian. We continue tilting to the left and right
 %through 40 degredes, 60 degrees and so on till we get a positive definite
 %hessian. Each time it enters this else statement the counter is
 %incremented so that the angle at which it turns is incremented. But at
 %any point of time only the same point is scanned.
%%
        if mod(counter,2)== 0
            thetahess=(counter+1)*20;
            flag_dir1=0;%To rotate left
        else
            thetahess=counter*20*2;% We have multiplied by two because it
            %has to come to original position then move to the right.
```

```
            flag_dir1=1;%To rotate right
        end

%% Condition for stopping
        y=var(L(:));
        tot=sum(L);
        sz=size(L);
        avg=tot/sz(1);
        if y<525 && avg < 270
        NXT_PlayTone(300,200);pause(1);
        NXT_PlayTone(700,200);pause(1);
        NXT_PlayTone(1000,200);
        break;
        end

%% Making the robot rotate through the angle

        counter=counter+1; %Increment the counter

        a2 = round((tp/90 * thetahess)/2);
        t2 = abs(a2);

        if flag_dir1==0% Rotating in the left direction
            NXC_MotorControl(0, 40, t2, false, 'Brake', false);
            NXC_MotorControl(1, -40, t2, false, 'Brake', false);
            pause(3)

        elseif flag_dir1==1% Rotating in the right direction
            NXC_MotorControl(0, -40, t2, false, 'Brake', false);
            NXC_MotorControl(1, 40, t2, false, 'Brake', false);
            pause(3)

        end

%% Bringing the sensor to the same point by moving backwards

        mbkz=NXTMotor('AB','Power',-50);
        mbkz.SpeedRegulation = false ;
        mbkz.TachoLimit = 350;
        mbkz.ActionAtTachoLimit = 'Brake' ;
        mbkz.SendToNXT();
        mbkz.WaitFor();
        StopMotor('all', 'off');
        flag_forward=1;
    end

%% If the hessian is positive call the function for step length

    if flag_forward==0
        funcstepn();
        StopMotor('all', 'off');
    end

%% This part of the code should be used if the robot has to move according
```

```
% to the value obtained by Newton's method. This code should be used
% instead of the above. But this step length is inaccurate.

    %if flag_forward==0
    %forward=round(r*2.1);
    %mab = NXTMotor('AB','Power', 30);
    %mab.SpeedRegulation = false ;
    %mab.TachoLimit = forward;
    %mab.ActionAtTachoLimit = 'Brake' ;
    %mab.SendToNXT();
    %mab.WaitFor();
    %end


%% Checking again
    if y<525 && avg < 270
        NXT_PlayTone(300,200);pause(1);
        NXT_PlayTone(700,200);pause(1);
        NXT_PlayTone(1000,200);
        break;
    end

    CloseSensor(SENSOR_3);

end
```

The Step Length Function:

```
%%                      STEP LENGTH FUNCTION
%                      ----------------------
%This is the function to find out the exact step length the robot should
%take to get to the minimum. Once the angle of steepest descent is found
%out and the robot is aligned in the position, we make the robot to move
%through some arbitrarily chosen period of time. When the robot moves it
%simultaneously gets the light sensor values . The values of the positon
%are taken from the tachometer . The values of the light sensor vs
%tachometer reaading are obtained . A best fit parabola is taken by the
%method of least squares and the robot moves to the minimum of the
%parabola. If the robots incurs a maximum it takes a U-turn and goes back
%to the original position.
%%                            Function

function [] = funcstepn()

A=zeros(400,3);% Array containing the coefficients of the quadratic
%function of intensity versus tachometer value
flag_blfr=0;
portl= SENSOR_3;
OpenLight(portl, 'ACTIVE');% making the light sensor active
int_strt = zeros(1,400);%Gets the value of light sensor at each point
dist = zeros(1,400);%gets position of motors connected to the
```

```
%wheels at each point
n=400;%No of iterations for which the motor moves

%% Making the robot move forward

mab=NXTMotor('AB','Power', 30);
mab.SpeedRegulation = false ;
mab.TachoLimit = 0;
mab.ActionAtTachoLimit = 'Brake' ;
mab.ResetPosition();
mab.SendToNXT();

for it = 1:n
    int_strt(it)= GetLight(portl);% Getting sensor value at each point
    data=mab.ReadFromNXT();
    dist(it)= data.Position;% Getting tachometer value at each point

end

mab.Stop('brake');
mab.ResetPosition();
plot(dist,int_strt);% Plotting the graph of the intensity of light
%versus the tachometer positon at every point

%% Calculating the minimum by least squares method

for it=1:400
    A(it,1)=dist(it)^2;
    A(it,2)=dist(it);
    A(it,3)=1;
end
Y=zeros(3,1);% Y has the values of the coefficients of the quadratic
B=int_strt';
Y=A\B;

syms x ;
f = Y(1)*x^2+Y(2)*x+Y(3);% the quadratic function of the intensity value
%wrt tachometer value
f1 = diff(f);
f2 = diff(f1);
f1d=double(Y(2));%Slope
f2d=double(f2);%Second differentation value


%% Finding out the distance it has to move back.
%First we check whether the function has sufficient curvature by checking
%the coefficient of X^2 term of the quadratic. If its sufficiently large
%we can find out the minimum. We next check whether the function has a
%maximum or a minimum. If the function has a maximum we take a U-turn and
%come back to the original position . If the function has a minimum we find
% out the tachometer value at the minimum. If the coefficient of X^2 is too
% small we assume its a straight line and find the slope . If the slope
% is positive we go back to the starting point or if the slope is negative we
% we goto the point of minimum intensity.
%%
```

```
if abs(f2d)<=0.001% Checking whether curvature is large enough

    NXT_PlayTone(600,100);pause(1);
    NXT_PlayTone(440,100);

    if f2d>0% Whether it is a minimum
        dist_go = solve(f1);% The minimum of the function
        dist_back=dist(n)-dist_go;% The distance the robot has to move
        % back is the distance the robot has moved forward subtracted by
        % the distance of the minimum

    elseif f2d < 0 % Whether is it a maximum

        tp1 = 460;
        back1=double(dist(n));

        % Going back to the original position
        mab_back1 = NXTMotor('AB','Power', -40);
        mab_back1.SpeedRegulation = false ;
        mab_back1.TachoLimit = back1;
        mab_back1.ActionAtTachoLimit = 'Brake' ;
        mab_back1.SendToNXT();
        mab_back1.WaitFor();


        %Making a U turn

        theta1=180;
        a1 = round((tp1/90 * theta1)/2);
        t1 = abs(a1);
        NXC_MotorControl(0, 40, t1, false, 'Brake', false);
        NXC_MotorControl(1, -40, t1, false, 'Brake', false);

        %Making the bot move back so that the sensor comes to the original
        %position

        pause(2)
        mbk1=NXTMotor('AB','Power',-40);
        mbk1.SpeedRegulation = false ;
        mbk1.TachoLimit = 610;
        mbk1.ActionAtTachoLimit = 'Brake' ;
        mbk1.SendToNXT();
        mbk1.WaitFor();
        dist_back=1;

    end
else %If the function is a straight line

    NXT_PlayTone(440,100);pause(1);
    NXT_PlayTone(300,100);pause(1);
    NXT_PlayTone(500,100);

    if f1d>0% If the slope is positive that is the function is increasing
```

```
    % the robot is made to take a U-turn
        tp1 = 460;
        back1=double(dist(n));
        mab_back1 = NXTMotor('AB','Power', -40);
        mab_back1.SpeedRegulation = false ;
        mab_back1.TachoLimit = back1;
        mab_back1.ActionAtTachoLimit = 'Brake' ;
        mab_back1.SendToNXT();
        mab_back1.WaitFor();

        theta1=180;
        a1 = round((tp1/90 * theta1)/2);
        t1 = abs(a1);

        NXC_MotorControl(0, 40, t1, false, 'Brake', false);
        NXC_MotorControl(1, -40, t1, false, 'Brake', false);
        mbk2=NXTMotor('AB','Power',-100);
        mbk2.SpeedRegulation = false ;
        mbk2.TachoLimit = 1;
        mbk2.ActionAtTachoLimit = 'Brake' ;
        mbk2.SendToNXT();
        mbk2.WaitFor();

        pause(2)
        mbk1=NXTMotor('AB','Power',-40);
        mbk1.SpeedRegulation = false ;
        mbk1.TachoLimit = 610;
        mbk1.ActionAtTachoLimit = 'Brake' ;
        mbk1.SendToNXT();
        mbk1.WaitFor();
        dist_back=1;

    elseif f1d<=0 % If the slope is negative it goes to the minimum
        %intensity point
        int_min=min(int_strt);
        for iter3 = 1:n
            if int_strt(iter3)==int_min
                dist_go=dist(iter3);
                dist_back=dist(n)-dist_go+1;
            end
        end
    end
end
back=double(dist_back);
back=round(back);
if back>dist(400)%Precautionary statements to avoid the bot from going
    %beyond the initial point
    back=dist(400)+20;
end

if back==0%Precautionary statements to avoid the bot from going
    %beyond the initial point
    back=dist(400);
end
if back<0
    back=1;
```

```
end
if(flag_blfr==1)
    back=1;
end

%% Making the bot move back to appropriate point

mab_back = NXTMotor('AB','Power', -40);
mab_back.SpeedRegulation = false ;
mab_back.TachoLimit = back;
mab_back.ActionAtTachoLimit = 'Brake' ;
mab_back.SendToNXT();
mab_back.WaitFor();
end
```

OBSERVATIONS

- The **angle** of descent estimated by the Newton's Method were **highly accurate** varying by less than 10 to 15 degrees to the exact angle the robot has to turn to reach the minimum.
- When the arena designed using Matlab, figure2 was used the estimated values were very highly inaccurate.
- During the testing it was found out that at some positions on the arena the **hessian** estimated was **not a positive definite** and the estimated direction was the ascent direction.
- **Step length** estimated was in many occasions in occasions **incorrect**.
- If the hessian is determined correctly the minimum is reached mostly within two iterations.

CORRECTIONS TRIED

- The same arena was designed using Adobe Photoshop, figure1 with higher resolution and uniform gradient.
- After the first estimation at a point, if the hessian is not a positive definite, the robot orients itself at 20 degrees to the left from its original position. The rotation is done in such a way that the sensor comes back to the same point (approximately). Again the hessian is estimated, if it turns out wrong the robot 20 degrees to the right from its original orientation (40 degrees in total) and the estimation is done again. The process is repeated with 40, 60, 80 degrees in either directions until a positive definite hessian is obtained.
- The function used to find out the step length in the gradient descent method was used instead of the value obtained by Newton's method.

RESULTS OBTAINED BY APPLYING THE CORRECTIONS

- Using the arena with a better resolution gave significantly better solutions.
- The technique of rotating left and right to the same point worked and in most cases the hessian was estimated correctly before the robot had even rotated through an angle of 60 degrees.
- Utilizing the function used to find out the step length in the gradient descent method avoided the errors due to the inaccurate step length values obtained in Newton's method.

Data Collected

The sensor readings at different points of the arena are given in a separate file "Data.xls" along with this report.

Conclusion

The project has explained two important optimization techniques namely, steepest descent and Newton's optimization algorithm. The two algorithms are modeled using a robot trying to perform a specific task. This project delivers a visual presentation of the above algorithms for better understanding and also emphasizes the intricate nuances that arise when applying the algorithms in a practical situation. The project has presented various unconventional solutions for the numerous problems faced. It has also signified the importance of control systems in every phase of the project.

Thus the project has revealed that control systems and optimization plays a vital role even in such a small scaled project. And hence they would play a pivotal role when designing technologies of larger magnitudes.

References

- Optimization of Chemical Processes by Thomas F.Edgar
- Wikipedia