NORTH WESTERN UNIVERSITY, KHULNA

***12/20/2023***

***LAB Report***



***Course Title: Computer Graphics and Pattern Recognition Sessional*  
Course Code : *CSE-4302***

**S**

**U**

**B**

**M**

**I**

**T**

**T**

**E**

**D**

tTof

***Name: M. Raihan***

***Assistant Professor***

***Department of Computer Science and Engineering***

***North Western University, Khulna.***

***Name: Sajib Bhattacharjee***

***ID: 20201070010***

***Name: Chandan Sourav Mallick***

***ID: 20201065010***

***Name: Saifur Rahman***

***ID: 20201165010***

tByf

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| SN | Topic | Page no. |
| 1. | Member-1 (Sajib Bhattacharjee, ID-20201070010) |  |
|  | 1. Lab-1 |  |
|  | 1.1 Adaptive Decision Boundary |  |
|  | 1. Lab-2 |  |
|  | 2.1 Single Linkage Algorithm |  |
|  | 1. Lab-3 |  |
|  | 3.1 DDA Line drawing |  |
|  | 3.2 Bresenham Line Drawing |  |
|  | 3.3 Bresenham Circle Drawing |  |
|  | 3.4 Mid-Point Circle Drawing |  |
| 2. | Member-2 (Chandan Sourav Mallick, ID-20201065010) |  |
|  | 1. Lab-1 |  |
|  | 1.1 Adaptive Decision Boundary |  |
|  | 1. Lab-2 |  |
|  | 2.1 Single Linkage Algorithm |  |
|  | 1. Lab-3 |  |
|  | 3.1 DDA Line drawing |  |
|  | 3.2 Bresenham Line Drawing |  |
|  | 3.3 Bresenham Circle Drawing |  |
|  | 3.4 Mid-Point Circle Drawing |  |
| 3. | Member-3 (Name: Saifur Rahman, ID-20201165010) |  |
|  | 1. Lab-1 |  |
|  | 1.1 Adaptive Decision Boundary |  |
|  | 1. Lab-2 |  |
|  | 2.1 Single Linkage Algorithm |  |
|  | 1. Lab-3 |  |
|  | 3.1 DDA Line drawing |  |
|  | 3.2 Bresenham Line Drawing |  |
|  | 3.3 Bresenham Circle Drawing |  |
|  | 3.4 Mid-Point Circle Drawing |  |
| 4. | Research Paper |  |
| 5. | Presentation Slides |  |

NORTH WESTERN UNIVERSITY, KHULNA



Course Title:Computer Graphics and Pattern Recognition Sessional

Course Code: CSE-4302

Lab Report

|  |  |
| --- | --- |
| **Submitted by:**  Name: Sajib Bhattacharjee  Id: 20201070010  Department of Computer Science and Engineering  North Western University, Khulna | **Submitted to:**  Name: M. Raihan  Assistant Professor  Department of Computer Science and Engineering  North Western University, Khulna. |

**Submission Date :**  **Teacher’s Signature**

*NORTH WESTERN UNIVERSITY, KHULNA*

**

*Course Title :**Computer Graphics and Pattern Recognition Sessional*

*Course Code: CSE-4302*

*Lab – 1*

|  |  |
| --- | --- |
| ***Submitted by:***  *Name: Sajib Bhattacjarjee*  *Id: 20201070010*  *Department of Computer Science and Engineering*  *North Western University, Khulna* | ***Submitted to:***  *Name: M. Raihan*  *Assistant Professor*  *Department of Computer Science and Engineering*  *North Western University, Khulna.* |

***Submission Date :*** ***Teacher’s Signature***

**1.1** **Algorithm Name: Adaptive Decision Boundary**

Code:

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

struct FeatureVector {

double feature1, feature2;

FeatureVector(double f1, double f2) : feature1(f1), feature2(f2) {}

};

double euclideanDistance(const FeatureVector& vec1, const FeatureVector& vec2) {

return sqrt(pow(vec1.feature1 - vec2.feature1, 2) + pow(vec1.feature2 - vec2.feature2, 2));

}

class AdaptiveDecisionBoundary {

public:

AdaptiveDecisionBoundary(const vector<FeatureVector>& featureVectors) : featureVectors(featureVectors) {}

void trainModel() {

initializeClusters();

while (clusters.size() > 1) {

int minCluster1, minCluster2;

findClosestClusters(minCluster1, minCluster2);

mergeClusters(minCluster1, minCluster2);

} }

void testModel(const FeatureVector& testVector) {

int predictedCluster = predictCluster(testVector);

cout << "Predicted Cluster: " << predictedCluster << endl;

}

private:

vector<FeatureVector> featureVectors;

vector<vector<int>> clusters;

void initializeClusters() {

clusters.clear();

for (size\_t i = 0; i < featureVectors.size(); ++i) {

clusters.push\_back({static\_cast<int>(i)});

}}

double calculateDistance(int cluster1, int cluster2) {

double minDistance = numeric\_limits<double>::infinity();

for (int index1 : clusters[cluster1]) {

for (int index2 : clusters[cluster2]) {

double distance = euclideanDistance(featureVectors[index1], featureVectors[index2]);

if (distance < minDistance) {

minDistance = distance;

} } }

return minDistance;

}

void findClosestClusters(int& minCluster1, int& minCluster2) {

double minDistance = numeric\_limits<double>::infinity();

for (size\_t i = 0; i < clusters.size(); ++i) {

for (size\_t j = i + 1; j < clusters.size(); ++j) {

double distance = calculateDistance(i, j);

if (distance < minDistance) {

minDistance = distance;

minCluster1 = i;

minCluster2 = j;

}}}}

void mergeClusters(int cluster1, int cluster2) {

clusters[cluster1].insert(clusters[cluster1].end(), clusters[cluster2].begin(), clusters[cluster2].end());

clusters.erase(clusters.begin() + cluster2);

}

int predictCluster(const FeatureVector& testVector) {

double minDistance = numeric\_limits<double>::infinity();

int predictedCluster = -1;

for (size\_t i = 0; i < clusters.size(); ++i) {

for (int index : clusters[i]) {

double distance = euclideanDistance(testVector, featureVectors[index]);

if (distance < minDistance) {

minDistance = distance;

predictedCluster = i;

} }}

return predictedCluster;

}};

int main() { vector<FeatureVector> featureData = {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {10, 12}, {11, 13}, {13, 14}};

AdaptiveDecisionBoundary decisionBoundaryModel(featureData);

decisionBoundaryModel.trainModel();

FeatureVector testFeatureVector = {5, 6};

decisionBoundaryModel.testModel(testFeatureVector);

return 0;

}

***NORTH WESTERN UNIVERSITY, KHULNA***

******

***Course Title:******Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab – 2***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Sajib Bhattacjarjee***  ***Id: 20201070010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

**2.1 Algorithm Name: Single Linkage Algorithm**

Code:

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

double euclideanDistance(const vector<double>& point1, const vector<double>& point2) {

double sum = 0.0;

for (size\_t i = 0; i < point1.size(); ++i) {

sum += pow(point1[i] - point2[i], 2);

}

return sqrt(sum);

}

void clustering(vector<vector<double>>& data) {

vector<vector<int>> clusters;

for (int i = 0; i < static\_cast<int>(data.size()); ++i) {

clusters.push\_back({i});

}

cout << "Initial Clusters:" << endl;

for (const auto& cluster : clusters) {

for (int index : cluster) {

cout << index << " ";

}

cout << endl;

}

int a;

cout << "For single or Complete linkage, type 1 or 2 respectively: ";

cin >> a;

while (clusters.size() > 1) {

double minDistance = numeric\_limits<double>::infinity();

pair<int, int> merge = {0, 1};

for (size\_t i = 0; i < clusters.size(); ++i) {

for (size\_t j = i + 1; j < clusters.size(); ++j) {

double distance;

if (a == 1) {

distance = euclideanDistance(data[clusters[i][0]], data[clusters[j][0]]);

if (distance < minDistance) {

minDistance = distance;

merge = {static\_cast<int>(i), static\_cast<int>(j)};

}

} else if (a == 2) {

distance = euclideanDistance(data[clusters[i][0]], data[clusters[j][0]]);

if (distance > minDistance) {

minDistance = distance;

merge = {static\_cast<int>(i), static\_cast<int>(j)};

}}}}

clusters[merge.first].insert(clusters[merge.first].end(), clusters[merge.second].begin(), clusters[merge.second].end());

clusters.erase(clusters.begin() + merge.second);

cout << "Clusters:" << endl;

for (const auto& cluster : clusters) {

for (int index : cluster) {

cout << index << " ";

}

cout << endl;

}

}

cout << "Final cluster:";

for (int index : clusters[0]) {

cout << " " << index;

}

cout << endl;

}

int main() {

vector<vector<double>> arr = {{1, 2}, {5, 8}, {1.5, 1.8}, {8, 8}, {1, 0.6}, {9, 11}};

clustering(arr);

return 0;

}

***NORTH WESTERN UNIVERSITY, KHULNA***

******

***Course Title***: ***Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab – 3***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Sajib Bhattacjarjee***  ***Id: 20201070010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

**3.1 Algorithm Name: DDA Line generation Algorithm**

Code:

#include <iostream>>

#include<conio.h>

#include<math.h>

using namespace std;

int RoundFunction(float number)

{

if (number - (int)number < 0.5)

{

return (int)number;

}

else

{

return (int)(number + 1);

}

}

void DDALineDrawing(int x0, int y0, int x1, int y1)

{

int dx = x1 - x0;

int dy = y1 - y0;

int maxCount;

if (abs(dx) > abs(dy))

{

maxCount = abs(dx);

}

else

{

maxCount = abs(dy);

}

float x\_increment = (float)dx / maxCount;

float y\_increment = (float)dy / maxCount;

float x = x0;

float y = y0;

cout<<"Output: "<<endl<<endl;

for (int i = 0; i < maxCount; i++)

{

cout << RoundFunction(x) << " " << RoundFunction(y) << "\n";

x += x\_increment;

y += y\_increment;

}

}

int main()

{

int x0,y0, x1, y1;

cout<<"Enter the value for X0: ";

cin>>x0;

cout<<"Enter the value for y0: ";

cin>>y0;

cout<<"Enter the value for x1: ";

cin>>x1;

cout<<"Enter the value for y1: ";

cin>>y1;

DDALineDrawing(x0, y0, x1, y1);

getch();

}

**3.2 Algorithm Name: Bresenham's Line Algorithm**

Code:

#include<iostream>

#include<conio.h>

using namespace std;

void BresenhamLineDrawing(int x1, int y1, int x2, int y2)

{

int newValue = 2 \* (y2 - y1);

int slop\_Err = newValue - (x2 - x1);

cout<<"Output: "<<endl<<endl;

for (int x = x1, y = y1; x <= x2; x++) {

cout << "(" << x << "," << y << ")\n";

slop\_Err += newValue;

if (slop\_Err >= 0) {

y++;

slop\_Err -= 2 \* (x2 - x1);

}

}

}

int main()

{

int x1, y1 , x2 , y2;

cout<<"Enter the value for X0: ";

cin>>x1;

cout<<"Enter the value for y0: ";

cin>>y1;

cout<<"Enter the value for x1: ";

cin>>x2;

cout<<"Enter the value for y1: ";

cin>>y2;

BresenhamLineDrawing(x1, y1, x2, y2);

getch();

}

**3.3 Algorithm Name: Bresenham’s circle drawing algorithm**

Code:

#include <stdio.h>

#include <dos.h>

#include <graphics.h>

#include<conio.h>

void CircleDrawing(int x\_Coordinate, int y\_Coordinate, int x, int y)

{

putpixel(x\_Coordinate+x, y\_Coordinate+y, RED);

putpixel(x\_Coordinate-x, y\_Coordinate+y, RED);

putpixel(x\_Coordinate+x, y\_Coordinate-y, RED);

putpixel(x\_Coordinate-x, y\_Coordinate-y, RED);

putpixel(x\_Coordinate+y, y\_Coordinate+x, RED);

putpixel(x\_Coordinate-y, y\_Coordinate+x, RED);

putpixel(x\_Coordinate+y, y\_Coordinate-x, RED);

putpixel(x\_Coordinate-y, y\_Coordinate-x, RED);

}

void BresenhamCircle(int x\_Coordinate, int y\_Coordinate, int Radius)

{

int x = 0, y = Radius;

int d = 3 - 2 \* Radius;

CircleDrawing(x\_Coordinate, y\_Coordinate, x, y);

while (y >= x)

{

x++;

if (d > 0)

{

y--;

d = d + 4 \* (x - y) + 10;

}

else

d = d + 4 \* x + 6;

CircleDrawing(x\_Coordinate, y\_Coordinate, x, y);

delay(50);

}

}

int main()

{

int x\_Coordinate, y\_Coordinate , Radius;

cout<<"Enter the value for x\_Coordinate: ";

cin>>x\_Coordinate;

cout<<"Enter the value for y\_Coordinate: ";

cin>>y\_Coordinate;

cout<<"Enter the value for Radius: ";

cin>>Radius;

int gd = DETECT, gm;

initgraph(&gd, &gm, "");

BresenhamCircle(x\_Coordinate, y\_Coordinate, Radius);

getch();

}

**3.4 Algorithm Name: Mid-Point Circle Drawing Algorithm**

Code:

#include<iostream>

#include<conio.h>

using namespace std;

void midPointCircleDrawing(int x\_Coordinate, int y\_Coordinate, int Radius)

{

int x = Radius, y = 0;

cout<<"Output: "<<endl;

cout << "(" << x + x\_Coordinate << ", " << y + x\_Coordinate << ") ";

if (Radius > 0)

{

cout << "(" << x + x\_Coordinate << ", " << -y + y\_Coordinate << ") ";

cout << "(" << y + x\_Coordinate << ", " << x + y\_Coordinate << ") ";

cout << "(" << -y + x\_Coordinate << ", " << x + y\_Coordinate << ")\n";

}

int Point = 1 - Radius;

while (x > y)

{

y++;

if (Point <= 0){

Point = Point + 2\*y + 1;

}

else

{

x--;

Point = Point + 2\*y - 2\*x + 1;

}

if (x < y)

break;

cout << "(" << x + x\_Coordinate << ", " << y + y\_Coordinate << ") ";

cout << "(" << -x + x\_Coordinate << ", " << y + y\_Coordinate << ") ";

cout << "(" << x + x\_Coordinate << ", " << -y + y\_Coordinate << ") ";

cout << "(" << -x + x\_Coordinate << ", " << -y + y\_Coordinate << ")\n";

if (x != y)

{

cout << "(" << y + x\_Coordinate << ", " << x + y\_Coordinate << ") ";

cout << "(" << -y + x\_Coordinate << ", " << x + y\_Coordinate << ") ";

cout << "(" << y + x\_Coordinate << ", " << -x + y\_Coordinate << ") ";

cout << "(" << -y + x\_Coordinate << ", " << -x + y\_Coordinate << ")\n";

}

}

}

int main()

{

int x\_Coordinate,y\_Coordinate,Radius;

cout<<"Enter the value for x\_Coordinate: ";

cin>>x\_Coordinate;

cout<<"Enter the value for y\_Coordinate: ";

cin>>y\_Coordinate;

cout<<"Enter the value for Radius: ";

cin>>Radius;

midPointCircleDrawing(x\_Coordinate, y\_Coordinate, Radius);

getch();

NORTH WESTERN UNIVERSITY, KHULNA



Course Title: Computer Graphics and Pattern Recognition Sessional

Course Code: CSE-4302

Lab – Report

|  |  |
| --- | --- |
| **Submitted by:**  Name: Chandan Sourav Mallick  Id: 20201065010  Department of Computer Science and Engineering  North Western University, Khulna | **Submitted to:**  Name: M. Raihan  Assistant Professor  Department of Computer Science and Engineering  North Western University, Khulna. |

**Submission Date :**  **Teacher’s Signature**

***NORTH WESTERN UNIVERSITY, KHULNA***

******

***Course Title: Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab – 1***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Chandan Sourav Mallick***  ***Id: 20201065010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

* 1. **Algorithm Name: Adaptive Decision Boundary**

#include <iostream>

#include <vector>

#include <svm.h>

using namespace std;

int main() {

vector<vector<double>> inputData = {{1, 2}, {2, 3}, {3, 4}, {4, 5}};

vector<int> outputLabels = {-1, -1, 1, 1};

svm\_problem trainingProblem;

trainingProblem.l = inputData.size(); // Number of training examples

trainingProblem.y = new double[trainingProblem.l]; // Labels

trainingProblem.x = new svm\_node\*[trainingProblem.l]; // Data points

for (int i = 0; i < trainingProblem.l; ++i) {

trainingProblem.y[i] = outputLabels[i];

trainingProblem.x[i] = new svm\_node[inputData[i].size() + 1];

for (int j = 0; j < inputData[i].size(); ++j) {

trainingProblem.x[i][j].index = j + 1;

trainingProblem.x[i][j].value = inputData[i][j];

}

trainingProblem.x[i][inputData[i].size()].index = -1;

trainingProblem.x[i][inputData[i].size()].value = 0;

}

svm\_parameter svmParams;

svm\_init\_param(&svmParams);

svmParams.svm\_type = C\_SVC;

svmParams.kernel\_type = RBF;

svmParams.gamma = 0.5; // Adjust this for adaptability

svm\_model \*trainedModel = svm\_train(&trainingProblem, &svmParams);

vector<double> testInput = {5, 6};

svm\_node testNode[testInput.size() + 1];

for (int i = 0; i < testInput.size(); ++i) {

testNode[i].index = i + 1;

testNode[i].value = testInput[i];

}

testNode[testInput.size()].index = -1;

testNode[testInput.size()].value = 0;

double prediction = svm\_predict(trainedModel, testNode);

cout << "Prediction: " << prediction << endl;

svm\_free\_and\_destroy\_model(&trainedModel);

svm\_destroy\_param(&svmParams);

for (int i = 0; i < trainingProblem.l; ++i) {

delete[] trainingProblem.x[i];

}

delete[] trainingProblem.x;

delete[] trainingProblem.y;

return 0;

}

***NORTH WESTERN UNIVERSITY, KHULNA***

******

***Course Title: Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab – 2***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Chandan Sourav Mallick***  ***Id: 20201065010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

**2.1 Algorithm Name: Single Linkage Algorithm**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

struct Point {

double x, y;

Point(double x, double y) : x(x), y(y) {}

};

double euclideanDistance(const Point& p1, const Point& p2) {

return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));

}

class HierarchicalClustering {

public:

HierarchicalClustering(const vector<Point>& points) : points(points) {}

void performSingleLinkage() {

initializeClusters();

while (clusters.size() > 1) {

int minCluster1, minCluster2;

findClosestClusters(minCluster1, minCluster2);

mergeClusters(minCluster1, minCluster2);

}

}

void printClusters() {

for (const auto& cluster : clusters) {

cout << "Cluster:";

for (int pointIndex : cluster) {

cout << " (" << points[pointIndex].x << ", " << points[pointIndex].y << ")";

}

cout << endl;

}

}

private:

vector<Point> points;

vector<vector<int>> clusters;

void initializeClusters() {

clusters.clear();

for (size\_t i = 0; i < points.size(); ++i) {

clusters.push\_back({static\_cast<int>(i)});

}

}

double calculateDistance(int cluster1, int cluster2) {

double minDistance = numeric\_limits<double>::infinity();

for (int index1 : clusters[cluster1]) {

for (int index2 : clusters[cluster2]) {

double distance = euclideanDistance(points[index1], points[index2]);

if (distance < minDistance) {

minDistance = distance;}}}

return minDistance;

}

void findClosestClusters(int& minCluster1, int& minCluster2) {

double minDistance = numeric\_limits<double>::infinity();

for (size\_t i = 0; i < clusters.size(); ++i) {

for (size\_t j = i + 1; j < clusters.size(); ++j) {

double distance = calculateDistance(i, j);

if (distance < minDistance) {

minDistance = distance;

minCluster1 = i;

minCluster2 = j;

}}}}

void mergeClusters(int cluster1, int cluster2) {

clusters[cluster1].insert(clusters[cluster1].end(), clusters[cluster2].begin(), clusters[cluster2].end());

clusters.erase(clusters.begin() + cluster2);

}};

int main() {

vector<Point> dataPoints = {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {10, 12}, {11, 13}, {13, 14}};

HierarchicalClustering hierarchicalClustering(dataPoints);

hierarchicalClustering.performSingleLinkage();

hierarchicalClustering.printClusters();

return 0;

}

***NORTH WESTERN UNIVERSITY, KHULNA***

******

***Course Title: Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab – 3***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Chandan Sourav Mallick***  ***Id: 20201065010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

* 1. **Algorithm Name: DDA Line Drawing**

#include <bits/stdc++.h>

using namespace std;

int round(float n)

{

if (n - (int)n < 0.5)

return (int)n;

return (int)(n + 1);

}

void DDALine(int x0, int y0, int x1, int y1)

{

int dx = x1 - x0;

int dy = y1 - y0;

int step;

if (abs(dx) > abs(dy))

step = abs(dx);

else

step = abs(dy);

float x\_incr = (float)dx / step;

float y\_incr = (float)dy / step;

float x = x0;

float y = y0;

for (int i = 0; i < step; i++) {

cout << round(x) << " " << round(y) << "\n";

x += x\_incr;

y += y\_incr;

}

}

int main()

{

int x0 = 200, y0 = 180, x1 = 180, y1 = 160;

DDALine(x0, y0, x1, y1);

return 0;

}

* 1. **Algorithm Name: Bresenham Line drawing**

#include <bits/stdc++.h>

using namespace std;

void bresenham(int x1, int y1, int x2, int y2)

{

int m\_new = 2 \* (y2 - y1);

int slope\_error\_new = m\_new - (x2 - x1);

for (int x = x1, y = y1; x <= x2; x++) {

cout << "(" << x << "," << y << ")\n";

slope\_error\_new += m\_new;

if (slope\_error\_new >= 0) {

y++;

slope\_error\_new -= 2 \* (x2 - x1);

}

}

}

int main()

{

int x1 = 3, y1 = 2, x2 = 15, y2 = 5;

bresenham(x1, y1, x2, y2);

return 0;

}

* 1. **Algorithm Name: Bresenham Circle Drawing**

#include <stdio.h>

#include <dos.h>

#include <graphics.h>

void drawCircle(int xc, int yc, int x, int y)

{

putpixel(xc+x, yc+y, RED);

putpixel(xc-x, yc+y, RED);

putpixel(xc+x, yc-y, RED);

putpixel(xc-x, yc-y, RED);

putpixel(xc+y, yc+x, RED);

putpixel(xc-y, yc+x, RED);

putpixel(xc+y, yc-x, RED);

putpixel(xc-y, yc-x, RED);

}

void circleBres(int xc, int yc, int r)

{

int x = 0, y = r;

int d = 3 - 2 \* r;

drawCircle(xc, yc, x, y);

while (y >= x)

{ x++;

if (d > 0)

{

y--;

d = d + 4 \* (x - y) + 10;

}

else

d = d + 4 \* x + 6;

drawCircle(xc, yc, x, y);

delay(50);

}

}

int main()

{

int xc = 50, yc = 50, r = 30;

int gd = DETECT, gm;

initgraph(&gd, &gm, "");

circleBres(xc, yc, r);

return 0;

}

**3.4 Algorithm Name: Mid Pint Circle Drawing**

#include<iostream>

using namespace std;

void midPointCircleDraw(int x\_centre, int y\_centre, int r)

{

int x = r, y = 0;

cout << "(" << x + x\_centre << ", " << y + y\_centre << ") ";

if (r > 0)

{

cout << "(" << x + x\_centre << ", " << -y + y\_centre << ") ";

cout << "(" << y + x\_centre << ", " << x + y\_centre << ") ";

cout << "(" << -y + x\_centre << ", " << x + y\_centre << ")\n";

}

int P = 1 - r;

while (x > y)

{

y++;

if (P <= 0)

P = P + 2\*y + 1;

else

{

x--;

P = P + 2\*y - 2\*x + 1;

}

if (x < y)

break;

cout << "(" << x + x\_centre << ", " << y + y\_centre << ") ";

cout << "(" << -x + x\_centre << ", " << y + y\_centre << ") ";

cout << "(" << x + x\_centre << ", " << -y + y\_centre << ") ";

cout << "(" << -x + x\_centre << ", " << -y + y\_centre << ")\n";

if (x != y)

{

cout << "(" << y + x\_centre << ", " << x + y\_centre << ") ";

cout << "(" << -y + x\_centre << ", " << x + y\_centre << ") ";

cout << "(" << y + x\_centre << ", " << -x + y\_centre << ") ";

cout << "(" << -y + x\_centre << ", " << -x + y\_centre << ")\n";

}

}

}

int main()

{

midPointCircleDraw(0, 0, 3);

return 0;

}

NORTH WESTERN UNIVERSITY, KHULNA



Course Title:Computer Graphics and Pattern Recognition Sessional

Course Code: CSE-4302

Lab – Report

|  |  |
| --- | --- |
| **Submitted by:**  Name: Saifur Rahman  Id: 20201165010  Department of Computer Science and Engineering  North Western University, Khulna | **Submitted to:**  Name: M. Raihan  Assistant Professor  Department of Computer Science and Engineering  North Western University, Khulna. |

**Submission Date :**  **Teacher’s Signature**

***NORTH WESTERN UNIVERSITY, KHULNA***

******

***Course Title: Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab – 1***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Saifur Rahman***  ***Id: 20201165010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

* 1. **Algorithm Name: Adaptive decision boundary algorithm**

#include <iostream>

#include <vector>

#include <svm.h>

using namespace std;

int main() {

vector<vector<double>> trainingData = {{1, 2}, {2, 3}, {3, 4}, {4, 5}};

vector<int> labels = {-1, -1, 1, 1};

svm\_problem problem;

problem.l = trainingData.size();

problem.y = new double[problem.l];

problem.x = new svm\_node\*[problem.l];

for (int i = 0; i < problem.l; ++i) {

problem.y[i] = labels[i];

problem.x[i] = new svm\_node[trainingData[i].size() + 1];

for (int j = 0; j < trainingData[i].size(); ++j) {

problem.x[i][j].index = j + 1;

problem.x[i][j].value = trainingData[i][j];

}

problem.x[i][trainingData[i].size()].index = -1;

problem.x[i][trainingData[i].size()].value = 0;

}

svm\_parameter param;

svm\_init\_param(&param);

param.svm\_type = C\_SVC;

param.kernel\_type = RBF;

param.gamma = 0.5; // Adjust this for adaptability

svm\_model \*model = svm\_train(&problem, &param);

vector<double> testData = {5, 6};

svm\_node testNode[testData.size() + 1];

for (int i = 0; i < testData.size(); ++i) {

testNode[i].index = i + 1;

testNode[i].value = testData[i];

}

testNode[testData.size()].index = -1;

testNode[testData.size()].value = 0;

double prediction = svm\_predict(model, testNode);

cout << "Prediction: " << prediction << endl;

svm\_free\_and\_destroy\_model(&model);

svm\_destroy\_param(&param);

for (int i = 0; i < problem.l; ++i) {

delete[] problem.x[i];

}

delete[] problem.x;

delete[] problem.y;

return 0;

}

***NORTH WESTERN UNIVERSITY, KHULNA***

******

***Course Title: Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab – 2***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Saifur Rahman***  ***Id: 20201165010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

**2.1 Algorithm Name: Hierarchical Clustering Single linkage**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

struct Data {

double coord1, coord2;

Data(double c1, double c2) : coord1(c1), coord2(c2) {}

};

double euclideanDistance(const Data& d1, const Data& d2) {

return sqrt(pow(d1.coord1 - d2.coord1, 2) + pow(d1.coord2 - d2.coord2, 2));

}

class ClusterAnalysis {

public:

ClusterAnalysis(const vector<Data>& data) : data(data) {}

void performSingleLinkage() {

initializeClusters();

while (clusters.size() > 1) {

int minCluster1, minCluster2;

findClosestClusters(minCluster1, minCluster2);

mergeClusters(minCluster1, minCluster2);

}

}

void printClusters() {

for (const auto& cluster : clusters) {

cout << "Cluster:";

for (int dataIndex : cluster) {

cout << " (" << data[dataIndex].coord1 << ", " << data[dataIndex].coord2 << ")";

}

cout << endl;

}

}

private:

vector<Data> data;

vector<vector<int>> clusters;

void initializeClusters() {

clusters.clear();

for (size\_t i = 0; i < data.size(); ++i) {

clusters.push\_back({static\_cast<int>(i)});

}

}

double calculateDistance(int cluster1, int cluster2) {

double minDistance = numeric\_limits<double>::infinity();

for (int index1 : clusters[cluster1]) {

for (int index2 : clusters[cluster2]) {

double distance = euclideanDistance(data[index1], data[index2]);

if (distance < minDistance) {

minDistance = distance;

}

}

}

return minDistance;

}

void findClosestClusters(int& minCluster1, int& minCluster2) {

double minDistance = numeric\_limits<double>::infinity();

for (size\_t i = 0; i < clusters.size(); ++i) {

for (size\_t j = i + 1; j < clusters.size(); ++j) {

double distance = calculateDistance(i, j);

if (distance < minDistance) {

minDistance = distance;

minCluster1 = i;

minCluster2 = j;

}

}

}

}

void mergeClusters(int cluster1, int cluster2) {

clusters[cluster1].insert(clusters[cluster1].end(), clusters[cluster2].begin(), clusters[cluster2].end());

clusters.erase(clusters.begin() + cluster2);

}

};

int main() {

vector<Data> dataset = {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {10, 12}, {11, 13}, {13, 14}};

ClusterAnalysis clusteringAnalysis(dataset);

clusteringAnalysis.performSingleLinkage();

clusteringAnalysis.printClusters();

return 0;

}

***NORTH WESTERN UNIVERSITY, KHULNA***

******

***Course Title: Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab – 3***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Saifur Rahman***  ***Id: 20201165010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

**3.1 Algorithm Name: DDA Line generation Algorithm**

Code:

#include <graphics.h>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <iostream.h>

int main(void)

{

int gdriver = DETECT, gmode, errorcode;

initgraph( & gdriver, & gmode, "C:\\tc\\bgi");

cout << "\n Enter X1,Y1,X2,Y2";

int x1, y1, x2, y2;

cin >> x1 >> y1 >> x2 >> y2;

int dx = x2 - x1;

int dy = y2 - y1;

int length;

if (dx >= dy)

length = dx;

else

length = dy;

dx = dx / length;

dy = dy / length;

int sx;

if (dx >= 0)

sx = 1;

else

sx = -1;

int sy;

if (dy >= 0)

sy = 1;

else

sy = -1;

float x = x1 + 0.5 \* (sx);

float y = y1 + 0.5 \* (sy);

int i = 0;

while (i <= length)

{

putpixel(int(x), int(y), 15);

x = x + dx;

y = y + dy;

i = i + 1;

}

getch();

closegraph();

}

**3.2 Algorithm Name: Bresenham's Line Algorithm**

Code:

#include<iostream.h>

#include<graphics.h>

void drawline(int x0, int y0, int x1, int y1)

{

int dx, dy, p, x, y;

dx=x1-x0;

dy=y1-y0;

x=x0;

y=y0;

p=2\*dy-dx;

while(x<x1)

{

if(p>=0)

{

putpixel(x,y,7);

y=y+1;

p=p+2\*dy-2\*dx;

}

else

{

putpixel(x,y,7);

p=p+2\*dy;

}

x=x+1;

}

}

int main()

{

int gdriver=DETECT, gmode, error, x0, y0, x1, y1;

initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");

cout<<"Enter co-ordinates of first point: ";

cin>>x0>>y0;

cout<<"Enter co-ordinates of second point: ";

cin>>x1>>y1;

drawline(x0, y0, x1, y1);

return 0;

}

**3.3 Algorithm Name: Bresenham’s circle drawing algorithm**

Code:

#include <iostream>

#include <graphics.h>

using namespace std;

void drawCircleBresenham(int xc, int yc, int r) {

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

int x = 0, y = r;

int p = 3 - 2 \* r;

putpixel(xc + x, yc - y, WHITE);

if (r > 0) {

putpixel(xc + x, yc + y, WHITE);

putpixel(xc - x, yc - y, WHITE);

putpixel(xc - x, yc + y, WHITE);

putpixel(xc + x, yc - y, WHITE);

}

while (x <= y) {

x++;

if (p > 0) {

y--;

p = p + 4 \* (x - y) + 10;

} else {

p = p + 4 \* x + 6;

}

putpixel(xc + x, yc - y, WHITE);

putpixel(xc - x, yc - y, WHITE);

putpixel(xc + x, yc + y, WHITE);

putpixel(xc - x, yc + y, WHITE);

if (x != y) {

putpixel(xc + y, yc - x, WHITE);

putpixel(xc - y, yc - x, WHITE);

putpixel(xc + y, yc + x, WHITE);

putpixel(xc - y, yc + x, WHITE);

}

}

delay(5000);

closegraph();

}

int main() {

int xc, yc, r;

cout << "Enter the center coordinates of the circle (xc yc): ";

cin >> xc >> yc;

cout << "Enter the radius of the circle: ";

cin >> r;

drawCircleBresenham(xc, yc, r);

return 0;

}

**3.4 Algorithm Name: Mid-Point Circle Drawing Algorithm**

Code:

#include <iostream>

#include <graphics.h>

using namespace std;

void drawCircleMidpoint(int xc, int yc, int r) {

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

int x = r, y = 0;

int p = 1 - r;

putpixel(xc + x, yc - y, WHITE);

if (r > 0) {

putpixel(xc - x, yc - y, WHITE);

putpixel(xc + x, yc + y, WHITE);

putpixel(xc - x, yc + y, WHITE);

}

while (x > y) {

y++;

if (p <= 0)

p = p + 2 \* y + 1;

else {

x--;

p = p + 2 \* y - 2 \* x + 1;

}

if (x < y)

break;

putpixel(xc + x, yc - y, WHITE);

putpixel(xc - x, yc - y, WHITE);

putpixel(xc + x, yc + y, WHITE);

putpixel(xc - x, yc + y, WHITE);

if (x != y) {

putpixel(xc + y, yc - x, WHITE);

putpixel(xc - y, yc - x, WHITE);

putpixel(xc + y, yc + x, WHITE);

putpixel(xc - y, yc + x, WHITE);

}

}

delay(5000);

closegraph();

}

int main() {

int xc, yc, r;

cout << "Enter the center coordinates of the circle (xc yc): ";

cin >> xc >> yc;

cout << "Enter the radius of the circle: ";

cin >> r;

drawCircleMidpoint(xc, yc, r);

return 0;

}