**Звіт**

**до лабораторної роботи №2**

Оформив студент групи ТТП-31

Казмірчук Володимир

Викладач: Голубєва Катерина

Постановка задачі

|  |  |  |
| --- | --- | --- |
|  |  | (градус) |
| 3 | 19 | 1 |
| 5 | 21 | 127 |
| -9 | -21 | 346 |
| -15 | 17 | 72 |
| 20 | 15 | 45 |
| -20 | -15 | 280 |
| 12 | 15 | 182 |
| -5 | 0 | 17 |
| 4 | 9 | 54 |

Для визначення місця положення деякого судна застосовують радіо-пеленгування. При цьому в деякій системі координат (x,y) відомі координати передавачів , які вимірюють кути напрямів на судно відносно вісі x проти годинникової стрілки.

Відомі координати розташування кількох передавачів, які зафіксували невідоме судно, а також відповідні кути напряму на це судно. Знайти координати положення невідомого судна.

Завдання 1

1) Розв’язати задачу.

2) Знайти відхилення розв’язку.

3) Намалювати геометричну інтерпретацію, зображення має мінятися відповідно до змін у постановці задачі. Площу орієнтовного розташування судна виділити.

3) Бути готовим до повного циклу комп’ютерного моделювання (уточнення вхідних даних, знов проходження всіх етапів комп’ютерного моделювання).

Завдання 2

Судно знайшли і зафіксували координати його розташування (x,y)

1) Знайти відстань між координатами положення судна (із завдання 1) та координатами його фактичного розташування.

2) З’ясувати чи фактичне розташування судна знаходиться в області його теоретичного розташування, яке визначається за даними пеленгування (можна це зробити графічно).

3) Намалювати геометричну інтерпретацію (аналогічно завданню 1), зображення має мінятися відповідно до змін у постановці задачі. Зобразити фактичне розташування судна, теоретичне розташування (знайдене у завданні 1), область теоретичного розташування (завдання 1).

Розв’язок

Для вирішення цієї задачі було написано наступний код на мові програмування Javascript:

const sensors = [

    {x: 3, y: 19, a: 1},

    {x: 5, y: 21, a: 127},

    {x: -9, y: -21, a: 346},

    {x: -15, y: 17, a: 72},

    {x: 20, y: 15, a: 45},

    {x: -20, y: -15, a: 280},

    {x: 12, y: 15, a: 182},

    {x: -5, y: 0, a: 17},

    {x: 4, y: 9, a: 54}

];

const rx = 25, ry = 25;

let segments = [];

let intersections = [];

let ox = 0, oy = 0;

// Draw axes

function axes (ctx, w, h, gx, gy) {

    ctx.beginPath ();

    ctx.strokeStyle = "rgb(0 0 0 / 50%)";

    ctx.lineWidth = 2;

    // X-axis

    ctx.moveTo (0, h / 2);

    ctx.lineTo (w, h / 2);

    // Y-axis

    ctx.moveTo (w / 2, 0);

    ctx.lineTo (w / 2, h);

    ctx.stroke ();

    // Setting style and font

    const fs = 12;

    ctx.font = `${fs}px Arial`;

    ctx.fillStyle = "rgb(0 0 0 / 50%)";

    let step = 1 + Math.round  (gx \* gy / 2000);

    step = step > 5 ? step - step % 5 : 1;

    console.log (step);

    ctx.strokeStyle = "rgb(0 0 0 / 20%)";

    ctx.lineWidth = 1;

    // Grid and labels for X

    const lsx = 0.2 \* w / gx;

    for (let x = -gx; x <= gx; x ++) {

        if (x == 0) { continue; }

        const pos = w / 2 + x \* w / gx;

        ctx.moveTo (pos, h / 2 + lsx);

        ctx.lineTo (pos, h / 2 - lsx);

        ctx.lineTo (pos, h / 2 + lsx);

        ctx.moveTo (pos, 0);

        ctx.lineTo (pos, h);

        if (Math.abs (x) % step == 0) {

            ctx.fillText (`${x}`, pos - fs \* 0.5, h / 2 + fs \* 1.5);

        }

    }

    // Grid and labels for Y

    const lsy = 0.2 \* h / gy;

    for (let y = -gy; y <= gy; y ++) {

        if (y == 0) { continue; }

        const pos = h / 2 + y \* h / gy;

        ctx.moveTo (w / 2 + lsy, pos);

        ctx.lineTo (w / 2 - lsy, pos);

        ctx.lineTo (w / 2 + lsy, pos);

        ctx.moveTo (0, pos);

        ctx.lineTo (w, pos);

        if (Math.abs (y) % step == 0) {

            ctx.fillText (`${-y}`, w / 2 - fs \* 1.5, pos + fs \* 0.5);

        }

    }

    ctx.stroke ();

    ctx.closePath ();

}

// Draw sensors, points and lines

function drawsensors (ctx, w, h, gx, gy) {

    ctx.beginPath ();

    ctx.strokeStyle = "rgb(0 0 0)";

    ctx.lineWidth = 3;

    const fs = 12;

    ctx.font = `${fs}px Arial`;

    ctx.fillStyle = "rgb(0 0 0)";

    // Draw lines

    segments.forEach (s => {

        const px = w / 2 + s [0] \* w / gx;

        const py = h / 2 - s [1] \* h / gy;

        const fx = w / 2 + s [2] \* w / gx;

        const fy = h / 2 - s [3] \* h / gy;

        ctx.moveTo (px, py);

        ctx.lineTo (fx, fy);

        ctx.fillText (`${s [0]}; ${s [1]}`, px - fs \* 1.2, py + fs \* 1.5);

    });

    ctx.stroke ();

    ctx.closePath ();

    // Draw sensors

    sensors.forEach (s => {

        ctx.beginPath ();

        const px = w / 2 + s.x \* w / gx;

        const py = h / 2 - s.y \* h / gy;

        ctx.fillStyle = "rgb(0 0 255)";

        ctx.arc (px, py, 5, 0, 2 \* Math.PI);

        ctx.fill ();

        ctx.closePath ();

    });

    // Draw intersections

    intersections.forEach (i => {

        ctx.beginPath ();

        const px = w / 2 + i [0] \* w / gx;

        const py = h / 2 - i [1] \* h / gy;

        ctx.fillStyle = "rgb(255 0 0)";

        ctx.arc (px, py, 5, 0, 2 \* Math.PI);

        ctx.fill ();

        ctx.closePath ();

        ctx.fillStyle = "rgb(0 0 0)";

        ctx.fillText (`${Math.round (i [0])}; ${Math.round (i [1])}`, px - fs \* 1.2, py + fs \* 1.5);

    });

    // Draw estimated position

    ctx.beginPath ();

    const px = w / 2 + ox \* w / gx;

    const py = h / 2 - oy \* h / gy;

    ctx.lineWidth = 1;

    ctx.fillStyle = "rgb(0 255 0)";

    ctx.arc (px, py, 5, 0, 2 \* Math.PI);

    ctx.fill ();

    ctx.stroke ();

    ctx.closePath ();

    ctx.fillStyle = "rgb(0 0 0)";

    ctx.fillText (`${Math.round (ox)}; ${Math.round (oy)}`, px - fs \* 1.2, py + fs \* 1.5);

}

// Draw answer

function drawans (ctx, w, h, gx, gy) {

    const px = w / 2 + rx \* w / gx;

    const py = h / 2 - ry \* h / gy;

    const opx = w / 2 + ox \* w / gx;

    const opy = h / 2 - oy \* h / gy;

    const dist = Math.sqrt ((px - opx) \* (px - opx) + (py - opy) \* (py - opy));

    // Draw line

    ctx.beginPath ();

    ctx.strokeStyle = "rgb(0 0 0)";

    ctx.lineWidth = 1;

    ctx.moveTo (opx, opy);

    ctx.lineTo (px, py);

    ctx.stroke ();

    // Draw real point and approximate zone

    ctx.beginPath ();

    ctx.fillStyle = "rgb(255 0 255)";

    ctx.arc (px, py, 5, 0, 2 \* Math.PI);

    ctx.fillStyle = "rgb(255 0 255 / 20%)";

    ctx.arc (opx, opy, dist + 2, 0, 2 \* Math.PI);

    ctx.fill ();

    ctx.stroke ();

    ctx.closePath ();

    // Add text

    const fs = 12;

    ctx.font = `${fs}px Arial`;

    ctx.fillStyle = "rgb(0 0 0)";

    ctx.fillText (`${rx}; ${ry}`, px - fs \* 1.2, py + fs \* 1.5);

    ctx.fillText (`D = ${Math.round (dist)}`, px + (opx - px) / 2 - fs \* 1.2, py + (opy - py) / 2 + fs \* 1.5);

}

// Display everything

function draw () {

    const canvas = document.querySelector(".Canvas");

    const w = (canvas.width = window.innerWidth);

    const h = (canvas.height = window.innerHeight);

    const ctx = canvas.getContext ("2d");

    ctx.clearRect (0, 0, w, h);

    const gx = Math.round (w / 15);

    const gy = Math.round (h / 12);

    //const gx = 25;

    //const gy = 25;

    console.log (gx, gy);

    axes (ctx, w, h, gx, gy);

    drawsensors (ctx, w, h, gx, gy);

    drawans (ctx, w, h, gx, gy);

}

// Find endpoints

sensors.forEach (s => {

    const e = 0.01, ex = 1000;

    let fx = s.x, fy = s.y;

    s.a = s.a % 360;

    if (90 - s.a > e || s.a - 270 > e) {

        fx += ex;

        fy += ex \* Math.tan (s.a \* Math.PI / 180);

    }

    else if (s.a - 90 > e && 270 - s.a > e) {

        fx -= ex;

        fy -= ex \* Math.tan (s.a \* Math.PI / 180);

    }

    else if (Math.abs (s.a - 90) <= e) { fy += ex\*ex; }

    else { fx -= ex\*ex; }

    segments.push ([s.x, s.y, fx, fy]);

});

// Find intersections

for (let i = 0; i < segments.length; i ++) {

    for (let j = i + 1; j < segments.length; j ++) {

        const a = segments [i], b = segments [j];

        const s1\_x = a [2] - a [0];

        const s1\_y = a [3] - a [1];

        const s2\_x = b [2] - b [0];

        const s2\_y = b [3] - b [1];

        const s = (-s1\_y \* (a [0] - b [0]) + s1\_x \* (a [1] - b [1])) / (-s2\_x \* s1\_y + s1\_x \* s2\_y);

        const t = ( s2\_x \* (a [1] - b [1]) - s2\_y \* (a [0] - b [0])) / (-s2\_x \* s1\_y + s1\_x \* s2\_y);

        if (s >= 0 && s <= 1 && t >= 0 && t <= 1)

        {

            intersections.push ([a [0] + (t \* s1\_x), a [1] + (t \* s1\_y)]);

        }

    }

}

console.log (intersections);

// Find final point

for (let i = 0; i < intersections.length; i ++)

{

    ox += intersections [i] [0];

    oy += intersections [i] [1];

}

ox /= intersections.length;

oy /= intersections.length;

console.log (ox, oy);

// Draw on begin and resize

addEventListener ("resize", (event) => {});

onresize = (event) => draw ();

draw ();

Цей код написаний на мові JavaScript та використовує HTML5 Canvas для відображення графіки. Ця програма моделює розташування датчиків, лінії їхнього охоплення, перетини цих ліній та приблизне розташування об'єкта, використовуючи введені координати датчиків та їхні охоплюючі кути.

Основні елементи коду:

1. Ініціалізація даних:

* Створюється масив *sensors*, що містить об'єкти з координатами та кутом для кожного сенсора.
* Задаються початкові значення для змінних *rx*, *ry* (координати точки, яку потрібно знайти).
* Ініціалізуються порожні масиви *segments* (для визначення кінців ліній сенсорів) та *intersections* (для знаходження точок перетину ліній сенсорів).

1. Функції малювання:

* Функція *axes* малює координатні вісі та сітку на графіку.
* Функція *drawsensors* відображає сенсори, лінії та точки перетину на графіку.
* Функція *drawans* малює лінію між точкою та оціненою позицією, а також відображає текстову інформацію.

1. Розрахунки та обчислення точок:

* Для кожного сенсора обчислюються координати його кінцевих точок та додаються до масиву *segments* (власна реалізація променів відрізками величезної довжини).
* Проводиться перевірка перетину ліній між сенсорами, і якщо точка перетину знаходиться всередині області перетину, вона додається до масиву *intersections*.
* Обчислюються ймовірні координати розташування об’єкту методом найменших квадратів.

1. Виведення результатів:

* Виклик функції *draw* для відображення результатів на екрані.
* Обробка події *resize* для перерисовки графіку при зміні розміру вікна.
* Виведення координат точок перетину та оціненої позиції у консолі.

Цей код візуалізує задані датчики та їх зони охоплення, розраховує точки перетину та приблизне розташування об'єкта на площині, з подальшою оцінкою отриманих результатів, чого і вимагають поставлені задачі.

Приклади роботи програми

Приклад №1

A graph with lines and dots

Description automatically generatedВивід:

Розмір осей координат та сітка відображаються автоматично в залежності від розміру вікна. Під кожною точкою виведені її координати, округлені до цілих значень. Позиції сенсорів позначено синім кольором, точки перетину їх променів – червоним, розрахована позиція судна – зеленим, а реальна – рожевим. З кожного сенсора намальовано промінь у відповідному напрямку, розрахована точка має зону похибки, радіус якох дорівнює відстані до реальної точки.