Kestirimle Hedefi Bul Projesi Raporu

Gökhan HAS

September 4, 2023

1 Kullanılan Programlar ve Teknolojiler

- Java 11.0.20.1
- Apache Maven 3.6.3
- Apache Kafka 2.13-3.5.1
- Eclipse 4.26.0
- Javadoc
- Latex
- Linux (Ubuntu 22.04.3)

2 Problem/Proje Tanımı

Bu projede sensörlerden elde edilen verilerle merkezi birimde kestirimle hedefin konum bilgisine ulaşma işlemi gerçekleştirilmiştir. Sensörler ve merkezi birimlerin haberleşmesinde Apache Kafka kullanılması beklenilmektedir.

3 Geliştirme Ortamı

Geliştirme ortamı olarak Ubuntu 22.04.3 - 1.60GHz 8 Çekirdek - 16 GB RAM - Intel Core i5 8265U işlemci özelliği barındıran bilgisayarda Eclipse 4.26.0 IDE'si üzerinde Java, Maven, Apache Kafka kullanılmıştır.

4 Gereksinimler

4.1 Fonksiyonel Geliştirmeler

- Gerekli uygulamalar çalıştırıldığında haberleşebilmelidir. Bu haberleşme Kafka üzerinden olmalıdır.
- Sensör uygulamaları sadece kendi konumu ve hedefin kerteriz bilgisini hesaplayarak merkezi birime iletmelidir.
- Merkezi birim bu bilgileri kullanarak hedefin tam konumunu hesaplayabilmelidir.
- Uygulamalar çalıştırılmadan önce Zookeper/Kafka sunucularının ayağa kaldırılması gerekmektedir.

4.2 Fonksiyonel Olmayan Geliştirmeler

- Uygulamaların açılmasında veya kapanmasında gecikme yaşanmamalıdır.
- Geliştirme ortamı olan Ubuntu 22.04.3 - 1.60GHz 8 Çekirdek - 16 GB RAM - Intel Core i5 8265U işlemci özelliği barındıran bilgisayarda çalışmalıdır.
- Hem Eclipse üzerinde hem de terminal üzerinde çalışabilir olmalıdır.

5 Tasarım

- Uygulama iki farklı sınıftan oluşmaktadır.
- Sensor sınıfı bir Producer gibi davranmaktadır. Main metodunda Kafka konfigürasyon ayarları yapıldıktan sonra kendine rastgele koordinatlar atar. Daha sonra hedef ile arasındaki açı hesaplanır. Koordinat bilgileryle birlikte bu açı bilgisi Kafka yardımıyla gönderilir.

```java
public class Sensor {
    /**
     * Main method to simulate sensor data generation and send it to a Kafka topic.
     * This program generates random sensor data, calculates the azimuth angle between
     * the sensor and a fixed target location, and sends the data to a Kafka topic.
     *
     * @param args An array of command-line arguments. The first argument (args[0]) is expected
     *             to be the sensor ID, which uniquely identifies the sensor.
     */
    public static void main(String[] args) {
        String sensorId = args[0];
        String topic = "quickstart-events";

        // Kafka producer configuration properties
        Properties properties = new Properties();
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "127.0.0.1:9092");
        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

        try {
            while(true) {
                double sensorX = getRandomCoordinate(-500, 500);
                double sensorY = getRandomCoordinate(-500, 500);

                // Target
                double targetX = -1;
                double targetY = 5;

                // Calculate the azimuth angle between the sensor and the target
                double azimuth = calculateAzimuth(sensorX, sensorY, targetX, targetY);

                // Create a sensor data message with sensor information and azimuth angle
                String sensorData = "Sensor= " + sensorId + " X= " + sensorX + " Y= " + sensorY + " degrees= " + azimuth;
                producer.send(new ProducerRecord<>(topic, sensorData));
                producer.flush();
            }
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            producer.close();
        }
    }
```

- Sensorün hedef ile olan kerteriz bilgisi hesaplamak için aşağıdaki metot kullanılmıştır. Bu açıyı derece cinsinden hesaplar.

```java
    /**
     * Calculates the azimuth angle in degrees between a sensor location and a target location.
     *
     * Azimuth represents the angle measured clockwise from the north direction (0 degrees) to
     * the direction pointing towards the target location. This method computes the azimuth
     * based on the sensor's coordinates (sensorX, sensorY) and the target's coordinates (targetX, targetY).
     *
     * @param sensorX The X-coordinate of the sensor's location.
     * @param sensorY The Y-coordinate of the sensor's location.
     * @param targetX The X-coordinate of the target's location.
     * @param targetY The Y-coordinate of the target's location.
     * @return The azimuth angle in degrees, where 0 degrees represents north and angles increase clockwise.
     */
    private static double calculateAzimuth(double sensorX, double sensorY, double targetX, double targetY) {
         // Calculate the change in X and Y coordinates.
        double deltaX = targetX - sensorX;
        double deltaY = targetY - sensorY;

        // Calculate the tangent of the angle formed by deltaY and deltaX.
        double tan = deltaY / deltaX;

         // Calculate the azimuth angle in radians and convert it to degrees.
        double azimuthInDegrees = Math.atan(tan) * 180 / Math.PI;

        // Handle special cases when deltaX is 0 or deltaY is negative.
        if(deltaX == 0) {
            azimuthInDegrees = sensorY < targetY ? 90 : 270;
        } else if(deltaY < 0) {
            azimuthInDegrees += 180;
        }

        // Ensure that the azimuth angle is within the range [0, 360) degrees.
        if(azimuthInDegrees < 0)
            azimuthInDegrees += 360;

        return azimuthInDegrees;
    }
```

- Merkezi birim ise bir Consumer gibi davranır. Kafka kuyruğunu dinleyerek verileri alır. Veriler belli formatta geldiği için ayrıştırılması gerekmektedir. Aşağıdaki ekran görüntüsünde bu işlemler yapılmaktadır.

```java
public class CentralUnit {
    /**
     * Main method for consuming Kafka messages, processing sensor data, and calculating target coordinates.
     *
     * This program subscribes to a Kafka topic, receives sensor data messages, processes the data to
     * extract sensor information (ID, X-coordinate, Y-coordinate, and degrees), and calculates the
     * target coordinates based on pairs of received sensor data points. It continuously listens for
     * incoming messages and performs calculations when two sensor data points are received.
     *
     * @param args An array of command-line arguments (not used in this program).
     */
    public static void main(String[] args) {
        String bootstrapServer = "localhost:9092";
        String topic = "quickstart-events";
        Properties properties = new Properties();
        properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServer);
        properties.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        properties.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, "test-consumer-group");
        KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(properties);

        consumer.subscribe(Arrays.asList(topic));
        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
            List<Double> xValues = new ArrayList<>();
            List<Double> yValues = new ArrayList<>();
            List<Double> degreesValues = new ArrayList<>();

            for (ConsumerRecord<String, String> record : records) {
                String value = record.value();
                String[] parts = value.split("\\s+");
                String sensorValue = "";
                double xValue = 0;
                double yValue = 0;
                double degreesValue = 0;

                for (int i = 0; i < parts.length; i++) {
                    if (parts[i].equals("Sensor=")) {
                        sensorValue = parts[i + 1];
                    } else if (parts[i].equals("X=")) {
                        xValue = Double.valueOf(parts[i + 1]);
                    } else if (parts[i].equals("Y=")) {
                        yValue = Double.valueOf(parts[i + 1]);
                    } else if (parts[i].equals("degrees=")) {
                        degreesValue = Double.valueOf(parts[i + 1]);
                    }
                }

               System.out.println("Sensor= " + sensorValue);
```

- İki sensörden veriler geldikten sonra hedefin koordinatları hesaplanır.

```java
62
63              System.out.println("Sensor= " + sensorValue);
64              System.out.println("X= " + xValue);
65              System.out.println("Y= " + yValue);
66              System.out.println("degrees= " + degreesValue);
67              System.out.println("************************");
68
69              xValues.add(xValue);
70              yValues.add(yValue);
71              degreesValues.add(degreesValue);
72
73              // Calculate target coordinates when two sensor data points are received
74              if(xValues.size() == 2) {
75                  double x = calculateX(xValues.get(0), yValues.get(0), degreesValues.get(0), xValues.get(1), yValues.get(1), degreesValues.g
76                  double y = calculateY(xValues.get(0), yValues.get(0), degreesValues.get(0), xValues.get(1), yValues.get(1), degreesValues.g
77
78                  System.out.println("Hedef nokta: (" + x + ", " + y + ")");
79
80                  xValues.clear();
81                  yValues.clear();
82                  degreesValues.clear();
83              }
84          }
85      }
86  }
87
88
```

- Başlangıç noktaları ve yönlendirildikleri açılar tarafından temsil edilen iki doğrunun kesişim noktasının koordinatları hesaplanır.

```java
    /**
     * Calculates the X-coordinate of the point of intersection between two lines, defined by
     * their starting points (x1, y1) and (x2, y2), and their respective azimuth angles (azimuth1, azimuth2).
     *
     * The method computes the X-coordinate of the intersection point of two lines represented by their starting
     * points and the angles at which they are oriented. It is assumed that the lines are infinite in length.
     *
     * @param x1 The X-coordinate of the starting point of the first line.
     * @param y1 The Y-coordinate of the starting point of the first line.
     * @param azimuth1 The azimuth angle in degrees for the first line, measured clockwise from the north direction.
     * @param x2 The X-coordinate of the starting point of the second line.
     * @param y2 The Y-coordinate of the starting point of the second line.
     * @param azimuth2 The azimuth angle in degrees for the second line, measured clockwise from the north direction.
     * @return The X-coordinate of the point where the two lines intersect.
     */
    private static double calculateX(double x1, double y1, double azimuth1, double x2, double y2, double azimuth2) {
        double tan1 = Math.tan(Math.toRadians(azimuth1));
        double tan2 = Math.tan(Math.toRadians(azimuth2));
        double x = ((y2 - y1) + (tan1 * x1) - (tan2 * x2)) / (tan1 - tan2);
        return x;
    }

    /**
     * Calculates the Y-coordinate of the point of intersection between two lines, defined by
     * their starting points (x1, y1) and (x2, y2), and their respective azimuth angles (azimuth1, azimuth2).
     *
     * The method computes the Y-coordinate of the intersection point of two lines represented by their starting
     * points and the angles at which they are oriented. It is assumed that the lines are infinite in length.
     *
     * @param x1 The X-coordinate of the starting point of the first line.
     * @param y1 The Y-coordinate of the starting point of the first line.
     * @param azimuth1 The azimuth angle in degrees for the first line, measured clockwise from the north direction.
     * @param x2 The X-coordinate of the starting point of the second line.
     * @param y2 The Y-coordinate of the starting point of the second line.
     * @param azimuth2 The azimuth angle in degrees for the second line, measured clockwise from the north direction.
     * @return The Y-coordinate of the point where the two lines intersect.
     */
    private static double calculateY(double x1, double y1, double azimuth1, double x2, double y2, double azimuth2) {
        double tan1 = Math.tan(Math.toRadians(azimuth1));
        double tan2 = Math.tan(Math.toRadians(azimuth2));
        double y = y1 + tan1 * (calculateX(x1, y1, azimuth1, x2, y2, azimuth2) - x1);
        return y;
    }
}
```

6 Çalıştırma Talimatları

Uygulamalar çalıştırılmadan önce "yukleme_dokumanı.pdf" 'te yer alan kurulumların yapılması gerekmektedir. Kurulumlar yapıldıktan sonra Apache Kafka sunucularının ayağa kaldırılmalıdır. Aksi halde uygulama çalışmaz. Bunun için iki farklı terminal açılmalıdır. Zookeeper sunucusu için aşağıdaki komut bir terminalde çalıştırılmalıdır.

```
zookeeper-server-start.sh ~/kafka_2.13-3.5.1/config/zookeeper.properties
```

Sunucu başarılı ayağa kaldırıltan sonra diğer terminalde aşağıdaki komut çalıştırılmalıdır.

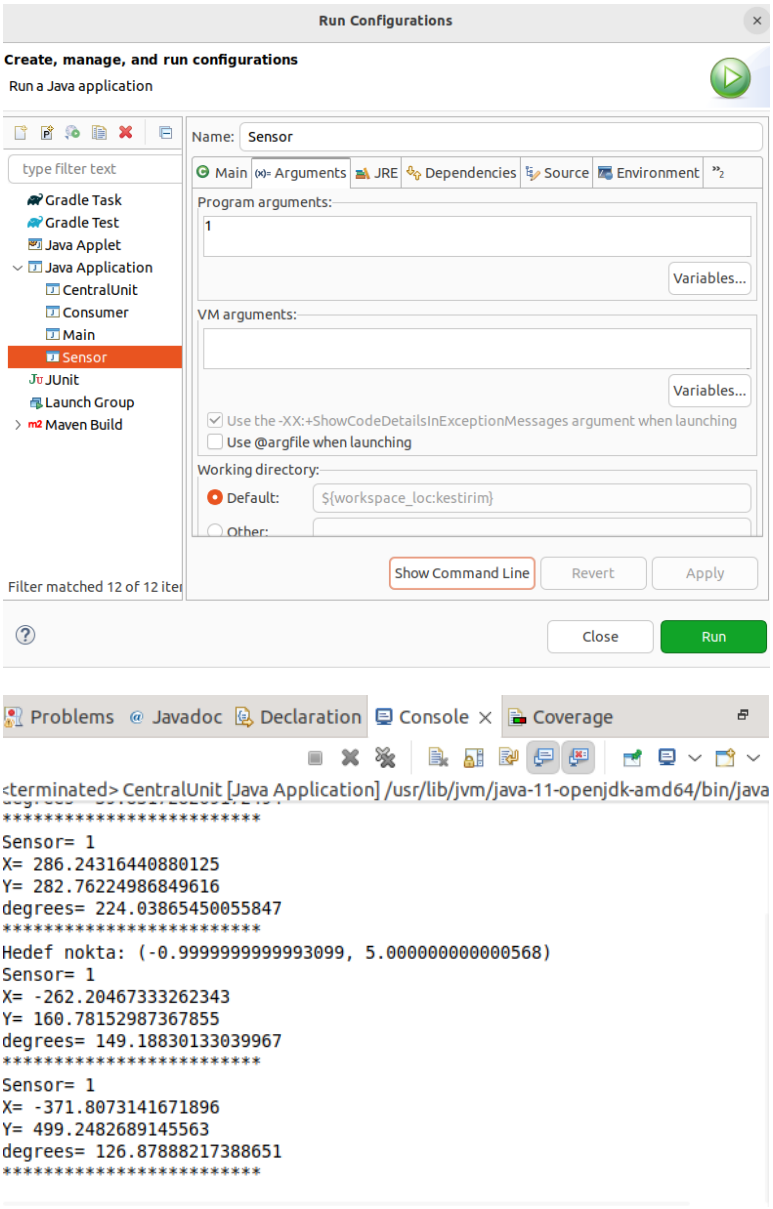
```
kafka-server-start.sh ~/kafka_2.13-3.5.1/config/server.properties
```

Proje videosunda bu adımlar yapılmıştır. İlerki maddelerden proje videosunun linki izlenebilir. Eğer sunucular başarılı bir şekilde ayağa kalkarsa uygulamaları çalıştırabiliriz. Maven bağımlılıklarında kurulması gerekebilir. Bu yüzden prjede bulunan pom.xml dosyasının dizininde terminal açıp, aşağıdaki komutu çalıştırmamız gerekebilir.

```
mvn clean install
```

6.1 Eclipse 4.26 ile

- Eclipse IDE'de proje dosyaları açıldıktan sonra Sensor.java dosyasını çalıştırabiliriz. Sonrasında CentralUnit.java'yı çalıştırdığımızda konsolda hesaplama sonuçları görülür. Burada dikkat edilmesi gereken Sensor uygulaması çalıştırılırken Eclipse'de command-line verilmesidir.
- Eclipse üzerinden uygulamaların çalıştırıldığında konsol ekran görüntüsü örneği aşağıda verilmiştir.



6.2 Linux (Ubuntu) Terminal ile

- Uygulamalar Maven projesi olarak geliştiridiği için terminalden çalıştırma konfigürasyon ayarları yapılmıştır.
- Sensor uygulamasını çalıştırmak için proje dizinide terminale aşağıdaki komut yazılmalıdır. -Dexec.args=”1” 1 numaralı sensörü temsil eder. Başka bir terminal açılıp, -Dexec.args=”2” şeklinde diğer sensör çalıştırılabilir.

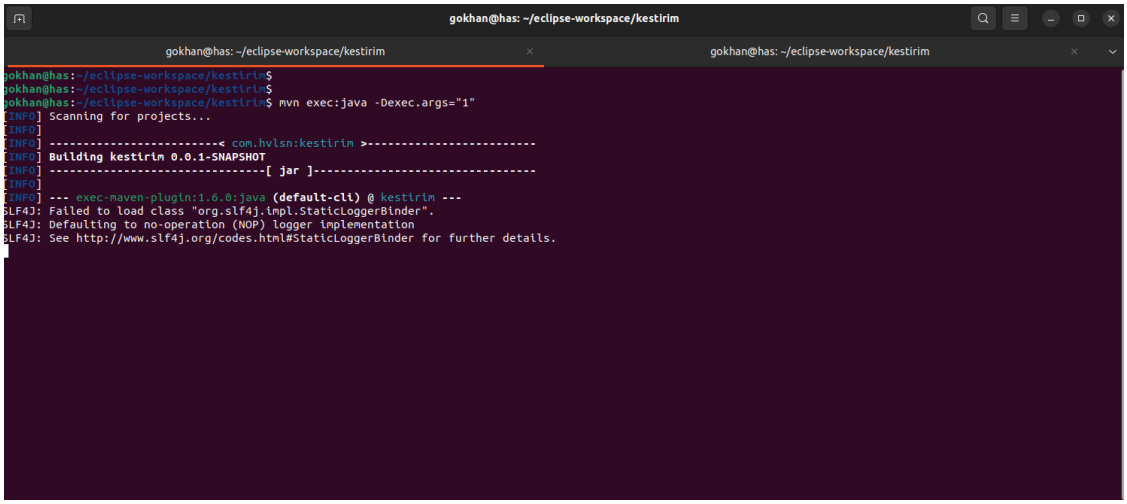
  ```
  mvn exec:java -Dexec.args="1"
  ```

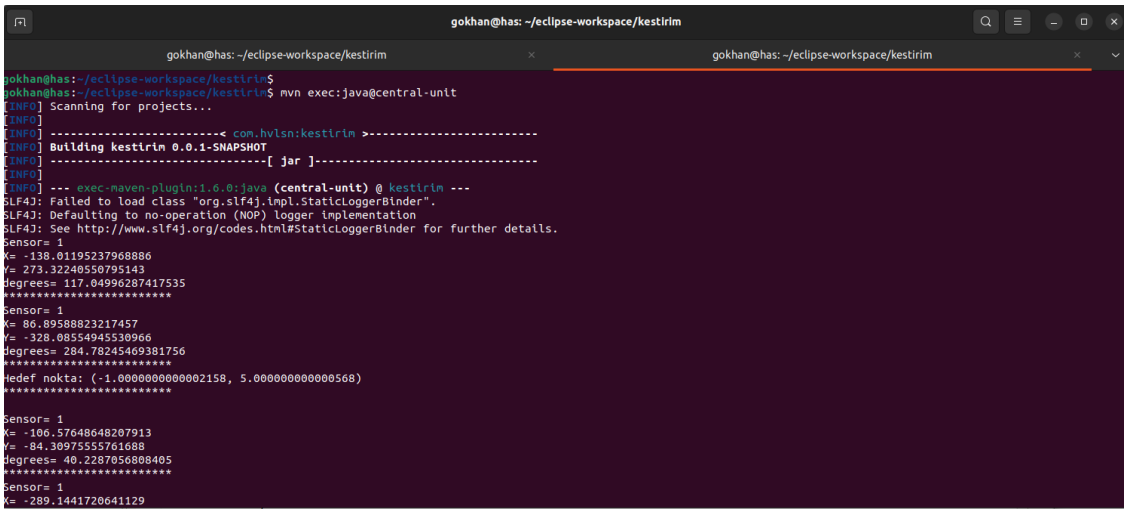
  ```
  mvn exec:java -Dexec.args="2"
  ```

- Merkezi birimin çalıştırılması için ise proje dizininde başka bir terminalden aşağıdaki komut çalıştırılmalıdır.

  ```
  mvn exec:java@central-unit
  ```

- Eğer gereklilikler yüklü veya entegre değilse raporla beraber sağlanan kurulum dokümanı yardımı ile kurulum yapmayı unutmayınız.





7 Çalıştırma Videosu

- Proje çalıştırma videosuna aşağıdaki linkten ulaşılabilir.

  `https://drive.google.com/file/d/1hxO3w0SPn0nMq2G5Qw94qQ1HUmNZR_Qa/view?usp=drive_link`

8 Kaynaklar

- https://kafka.apache.org/uses
- https://kafka.apache.org/quickstart
- https://kerteriz.net/linux-apache-kafka-kurulumu/
- https://linuxize.com/post/how-to-install-the-latest-eclipse-ide-on-ubuntu-18-04/
- https://www.cuemath.com/geometry/intersection-of-two-lines/
- https://gis.stackexchange.com/questions/108171/calculating-azimuth-from-two-points-both-having-latitude-longtiude
- https://www.overleaf.com/
- https://maven.apache.org/guides/introduction/introduction-to-the-pom.html