Programming C++

Project **Ransac Algorithmus**

Status: 25. Mai 2021, 19:36

Supervisor: MSc. S. Rosenberger, stefan.rosenberger@uni-graz.at

1. Dieses Projekt entspringt einer praktischen Anwendung. Um die Anschlussstelle ihres Projektes mit der praktischen Implementierung zu simulieren wird ein Zufallsgenerator zur Verfügung gestellt, welcher Ihnen einen Output liefert der dem praktischen Problem entspricht.[1] Ziel Ihres Projektes ist es zum Einen die Zufalls-Messung zu implementieren und zum Anderen existierende Algorithmen zu implementieren sowie mit Ihrer Implementierung zu vereinen.
   Voraussetzungen: Lineare Algebra 1 (3×3 Matrizen erstellen und invertieren können), Analysis 1 (Grundkenntnisse von Grenzprozessen) und C++ Kenntnisse.

   Ransac Algorithmus:
   Der Ransac Algorithmus (vgl. `https://de.wikipedia.org/wiki/RANSAC-Algorithmus`) ist eine effiziente Methode robust aus gegebenen Daten eine Geometrie zu erkennen. Wie Sie im Zuge des Projektes sehen werden hat er aber einen großen Nachteil.
   Für dieses Projekt wird Ihnen die Datei *randomGenerator.h* zur Verfügung gestellt. Dieser Generator kann Ihnen 2D-Punktwolken[2] für eine Linien und Kreisapproximation erstellen. Achtung: Der Generator benötigt eine von Ihnen zu erstellende Klasse *Point2D*. Weiters werden Sie die Algorithmen von Fit-Circle (:`https://people.cas.uab.edu/~mosya/cl/CPPcircle.html`) benötigen.

   Aufgabenstellung: (6 Pkt.)

   (a) Schreiben Sie eine Klasse *Point2D* welche einen zweidimensionalen Punkt bezüglich des Urpsrungs (0,0) beschreibt. Die beiden Koordinaten sollen dabei als templates (T) definiert werden können.
   Die Klasse **muss** die Routinen

```
// Konstruktoren
Point2D() = default;
Point2D(const T x, const T y)
// getter-setter Funktionen:
x()
setX(const T val)
```

[1] Eine ähnliche Aufgabenstellung wurde in einem Messsystem implementiert (also das Ding gibt's). Hierbei wird eine Geometrie mittels Laser-Messsystem vermessen (bis zu 5000 Messungen pro Sekunde werden/wurden verarbeitet → also definitiv eine Performance- Herausforderung). Um die Geometrie schnell vermessen zu können soll die Geometrie mithilfe einer Zufalls-Annäherung bestimmt werden.

[2] Mit einem sogenannten *nois* → Abweichungen entlang der Kontur.

```
y()
setY(const T val)

// Operatoren:
// Addition zweier Punkte (p1 += p2)
Point2D<T>& operator+=(const Point2D<T>& p)
// Mult. mit Skalar (p1 *= c)
Point2D<T>& operator*=(const T c)
// Differenz zweier Punkte (p = p1 - p2)
inline Point2D<T> operator-(Point2D<T> left,
        const Point2D<T>& right)
// Mult. mit Skalar (p = factor * p1)
inline Point2D<T> operator*(const S factor, Point2D<T> point)
```

besitzen damit der Algorithmus funktionieren wird. (Die Operatoren müssen nicht zwingend diesen Syntax folgen, es müssen lediglich die Operationen ausführbar sein).

(b) Verwenden Sie den *randomGenerator.h* um eine Gerade erstellen zu lassen. Hierfür könne Sie die Funktion

```
void getRandomLineElements(std::vector<Point2D<S>> &elements);
```

des Generators verwenden. Die Funktion

```
void setPercentDivergens(const float val);
```

des Generators beeinflusst die Streuung der Daten. Hierbei entspricht 1.0 einer Streuung von 100% bezogen auf die Länge der Gerade. (Sinnvolle Werte sind Zahlen im Bereich $0.001 - 0.1$). Zu Beginn kann es nützlich sein die Streuung auf 0 zu setzten, womit die erste Berechnung einfacher wird.

(c) **Überprüfungsumgebung:** Schreiben Sie eine Funktion welche die Punkte aus dem *randomGenerator* in eine Datei schreibt (Sie können auch die im *randomGenerator* vorhandene Funktion nutzen). Erstellen Sie anschließend ein Matlab (oder Octave) Script um die Punkte anzuzeigen (dieses wird nützlich sein, damit Sie ihr Resultat überprüfen können).

(d) **Basisklasse:** Erstellen Sie die (pure-abstract) Basisklasse

```
template <typename T>
class ransac
{
public:
  virtual void approximatePoints(
        const std::vector<Point2D<T>> &elements,
        const int percent_fit) = 0;
};
```

Leiten Sie beide folgenden ransac-Algorithmen von dieser Basis ab.
Hierbei ist *percent_fit* die maximale Abweichung die Ihre Approximation haben darf (wie Sie die Approximation umsetzen werden bleibt Ihnen überlassen).

(e) **Geradenapproximation:**

- Implementieren Sie eine Methode zur linearen Regression (Wiki: Lineare Einfachregression `https://de.wikipedia.org/wiki/Lineare_Einfachregression`).
- Erstellen Sie eine Klasse *ransac_line* welche von einem gegebenen

  ```
  std::vector<Point2D<T>>
  ```

  die Geradengleichung $ax + by + c = 0$ (bzw. $y = kx + d$) bestimmt. Der Algorithmus soll folgender Beschreibung folgen:

  Algorithm 1: Ransac Lineare Approximation

  initialization;
  while *Genauigkeit der Approximation ist nicht erreicht* **do**
  - Wähle 2 zufällige Punkte aus dem gegebenen Vektor. Diese müssen unterschiedlich sein.;
  - Bestimmen Sie aus diesen beiden Punkten die Geradengleichung.;
  - Berechnen Sie ein *Maß* für Ihre gefundene Gerade um die Approximation zu *beurteilen*. Dieses Maß darf nur von den Punkten (des übergebenen Vektors) abhängen und keine anderen Informationen verwenden.;

  end

- Implementieren Sie die Möglichkeit, dass der Algorithmus erst bei einem gegebenen Maß für die Genauigkeit abbricht. (Der Parameter *percent_fit* ist dafür vorgesehen.)
  Hinweis: Es wird nützlich sein, das Ergebnis der Berechnung als Member der Klasse zu erstellen und diese dann per getter-Funktion auslesbar zu machen. Die Basisklasse soll und darf keine Informationen beinhalten, welche nur in der abgeleiteten Klasse zu verwenden sind.
- Vergleichen Sie das Ergebnis der linearen Regression mit der Geraden aus Ihrer Ransac-Methode.

(f) Verwenden Sie den *randomGenerator.h* um einen Kreis erstellen zu lassen. Hierfür könne Sie die Funktion

```
void getRandomCircleElements(std::vector<Point2D<S>> &elements);
```

des Generators verwenden. Die Funktion

```
void setPercentDivergens(const float val);
```

beeinflusst in diesem Fall die Streuung in der Form, dass 1.0 einer Streuung im Maß des Radius des Kreises entspricht. Auch hierbei sind sinnvolle Werte im Bereich $0.001 - 0.1$.

(g) **Kreisapproximation:**

- Binden Sie die Routinen aus Fit-Circle (:`https://people.cas.uab.edu/~mosya/cl/CPPcircle.html`) in Ihr Projekt ein. Dies sind open-License Routinen um einen Kreis aus einer Punktwolke zu fitten. Testen Sie diese Anhand gegebener Daten. (Achtung, Sie werden die Daten aus dem Random-Generator konvertieren müssen damit Sie die Routinen anwenden können da diese natürlich Ihren Point2D nicht kennen.)

- Erstellen Sie eine Klasse *ransac_circle* welche von einem gegebenen

  ```
  std::vector<Point2D<T>>
  ```

  den Radius und das Zentrum bestimmt. Der Algorithmus soll folgender Beschreibung folgen:

  Algorithm 2: Ransac Kreis Approximation

  initialization;
  while *Genauigkeit der Approximation ist nicht erreicht* **do**
  - Wähle 3 zufällige Punkte aus dem gegebenen Vektor. Diese müssen unterschiedlich sein.;
  - Bestimmen Sie aus diesen drei Punkten Zentrum und Radius der Punktwolke.;
  - Berechnen Sie ein Maß für Ihr gefundenes Zentrum und Radius um die Approximation zu beurteilen. Dieses Maß darf nur von den Punkten (des übergebenen Vektors) abhängen und keine anderen Informationen verwenden.

  end

  Hinweis: Achten Sie darauf, dass es die Möglichkeit gibt, dass Ihr Algorithmus zwei Punkte wählt welche entweder die gleiche x oder die gleiche y Koordinate haben. Das ist kein Problem, solange die Berechnungen richtig implementiert werden.

(h) Führen Sie einen Performance-Vergleich durch um die Geschwindigkeit Ihrer Ransac-Routinen mit der Regressionsroutinen vergleichen zu können. Mithilfe der Routine

```
void setNumOfPoints(const int val)
```

des randomGenerators können Sie die Anzahl der Punkte in der Punktwolke fixieren lassen (damit ist diese nicht mehr zufällig). Sie sollten die Geschwindigkeit bei verschieden vielen Punkten durchführen.

(i) (Bonus:) Erstellen Sie eine Routine welche Zufällig eine Gerade oder einen Kreis erstellt. Erstellen Sie eine Routine welche nur die Basis der Ransac-Algorithmen und die erstellen Punkte als Übergabe-Parameter bekommt und die Approximation retourniert. Schließen Sie aus den Rückgabewerten ob es sich um eine Gerade oder einen Kreis handelte.
Damit lässt sich identifizieren welche Kontur gegeben ist, wenn diese unbekannt ist. (+2 Pkt.)