1	2	3	$\sum$

Prof. Daniel Göhring

Robotik, WiSe 18/19

Übung 09

Mai-Phú Pham, Yichi Chen, Dominik Dreiner

1 Map description (2 Punkte)

Im *gimp* haben wir die Positionen von einigen wichtigen Pixel aus der Karte "*map.bmp*" gemessen. Somit zeigen sich zunächst die geometrischen Funktionen für die beiden Geraden wie folgt.

$$G_1: \quad y = 0.95, \quad 1.95 \leq x \leq 4.05.$$
$$G_2: \quad y = 3.35, \quad 1.95 \leq x \leq 4.05.$$

Für die beiden Halbkreise verwenden wir allerdings die Polarkoordinaten für die Darstellung der Halbkreise wie folgt.

$$K_1: \begin{cases} x = 1.20 \cdot \cos\varphi + 1.95 \\ y = 1.20 \cdot \sin\varphi + 2.15 \end{cases}, \text{ für } 0 \leq x < 1.95. \tag{1}$$

$$K_2: \begin{cases} x = 1.20 \cdot \cos\varphi + 4.05 \\ y = 1.20 \cdot \sin\varphi + 2.15 \end{cases}, \text{ für } 4.05 < x \leq 6.00. \tag{2}$$

Hierbei wird die Metrik Meter verwendet.

2 Closest point on trajectory (3 Punkte)

Zunächst zeigt sich der Pseudocode für die Aufteilung der Karte wie folgt.

```
// Hierbei verwenden wir Zentimeter als Metrik statt Meter, so dass ein
// Zentimeter einem Pixel entspricht.
if x < 195 and x >= 0:
    if y >= 0 and y <= 430:
        calculate the closest distance to K1
    else:
        input mistake!
if x > 405 and x <= 600:
    if y >= 0 and y <= 430:
        calculate the closest distance to K2
    else:
        input mistake!
else if x >= 195 and x <= 405:
    if y < 0 or y > 430:
        input mistake!
    else:
        if y < 215:
            calculate the closest distance to G1
        else:
            calculate the closest distance to G2
else:
    input mistake!
```

Somit ergibt sich die aufgeteilte Karte wie in Abb. 1.

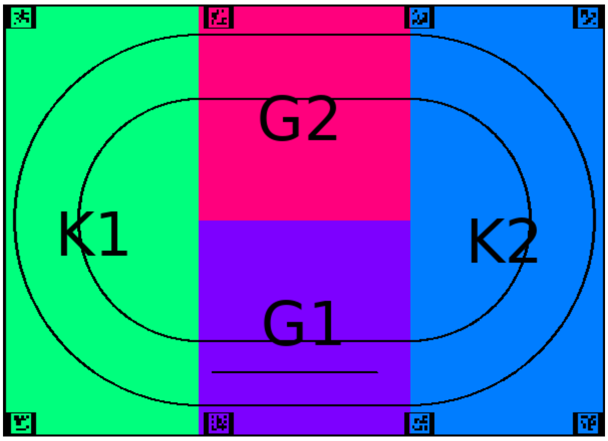

Abbildung 1: Die Aufgeteilte Karte

Unser Kode für das obere Bild befindet sich im folgenden Link: `https://git.imp.fu-berlin.de/phup/robotik-uebungen/blob/master/Aufgaben/aufgabe09/Programmierung/map_verarbeitung.py`.

Wenn sich der zutreffende Punkt auf der beiden Geraden G_1 oder G_2 befindet, ist die Rechnung relativ einfach. Sie zeigt sich wie folgt:

$$G_1: \quad (P_x, P_y) = (x, 0.95).$$
$$G_2: \quad (P_x, P_y) = (x, 3.35).$$

Wenn sich der zutreffende Punkt auf den beiden Kurven K_1 oder K_2 befindet, verwenden wir die tolle Eigenschaft der Polarkoordinaten, so dass es sich die folgenden Gleichungen ergeben.

$$K_1: \ (P_x, P_y) = (R \cdot \left(\frac{x - M_x}{d}\right) + M_x, \ R \cdot \left(\frac{y - M_y}{d}\right) + M_y),$$

$$K_2: \ (P_x, P_y) = (R \cdot \left(\frac{x - M'_x}{d'}\right) + M'_x, \ R \cdot \left(\frac{y - M'_y}{d'}\right) + M'_y),$$

wobei

$$R = 1.2, \ d = \sqrt{(x - M_x)^2 + (y - M_y)^2}, \ d' = \sqrt{(x - M'_x)^2 + (y - M'_y)^2},$$

und

$$(M_x, \ M_y) = (1.95, \ 2.15), \quad (M'_x, \ M'_y) = (4.05, \ 2.15)$$

gelten.
Anschließend berechnen wir die zutreffenden Punkte für gegebene Punkte. Die Ergebnisse zeigen sich wie in Abb. 2, 3, und 4. Der dafür zuständige Code befindet sich im folgenden Link:
`https://git.imp.fu-berlin.de/phup/robotik-uebungen/blob/master/Aufgaben/aufgabe09/Programmierung/calc_track_v2.py`.

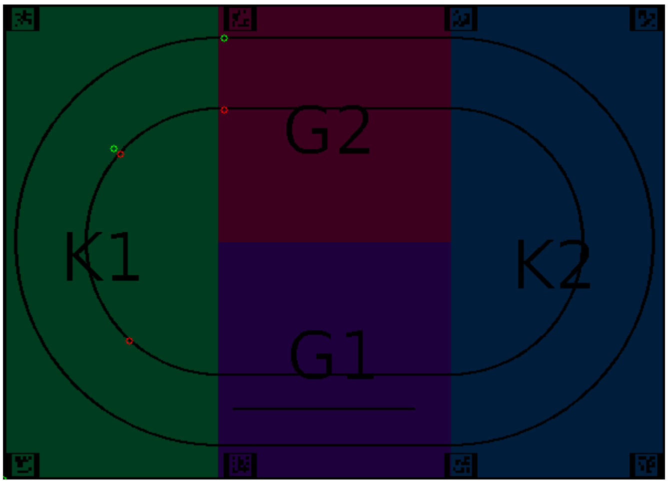

Abbildung 2: Gefundene Punkte (rot markiert) für die drei gegebenen Punkte (grün markiert)

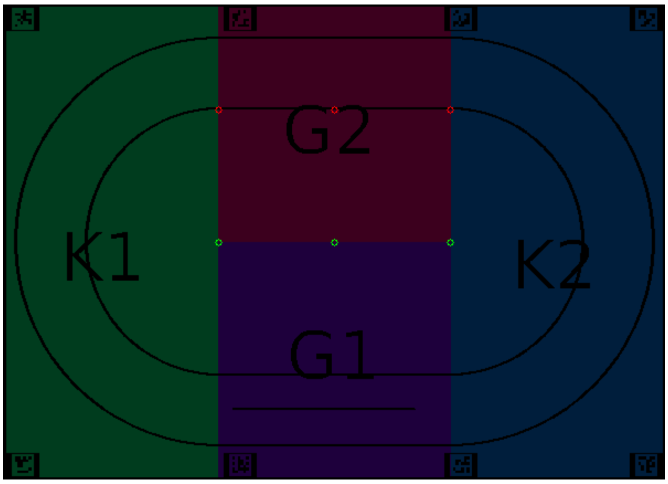

Abbildung 3: Gefundene Punkte (rot markiert) für drei Sonderfälle (grün markiert)

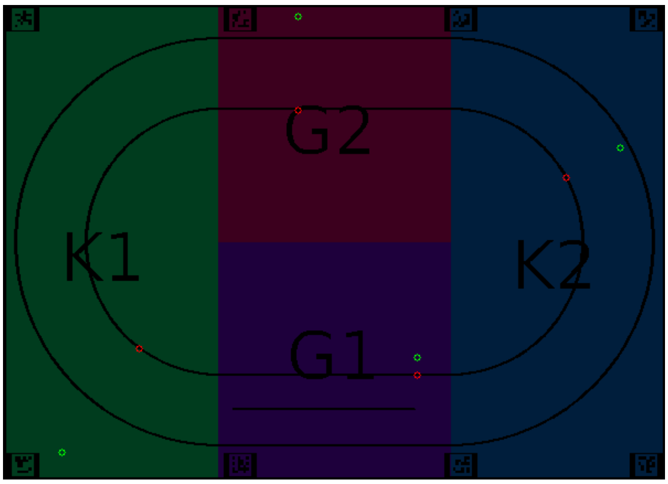

Abbildung 4: Gefundene Punkte (rot markiert) für weitere Eingabepunkte (grün markiert)

3 Ceiling camera GPS (5 Punkte)

Vorweg der Link für das von uns aufgenomme Video für die Kreisfahrt:
`https://git.imp.fu-berlin.de/phup/robotik-uebungen/blob/master/Aufgaben/aufgabe09/Videos/Konvertierte_Videos/v1_Rundfahrt_20190116_100kbs_16fps.mp4`.

Die Fahrtspur zeigt sich wie in Abb. 5. Dabei fing das Modelcar bei lila-farbigen Punkten an, und endet bei roten Punkten.

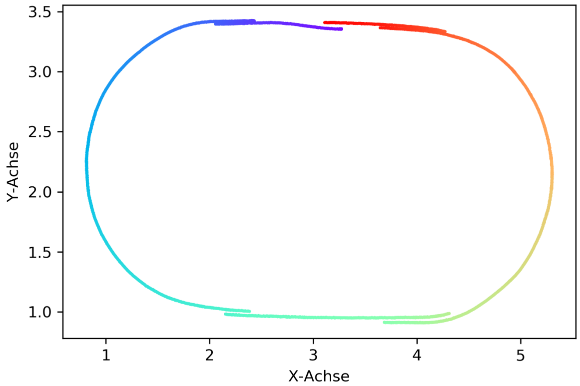

Abbildung 5: Fahrtspur für eine Kreisfahrt

Nun geben wir die Formeln für die **MAE** (*mean-absolute error*) und **MSE** (*mean-squared error*)[1]:

$$d_{MAE} = \frac{||x-y||_1}{n} = \frac{1}{n}\sum_{i=1}^{n}(|x_i - \hat{x}_i| + |y_i - \hat{y}_i|) \tag{3}$$

$$d_{MSE} = \frac{||x-y||_2^2}{n} = \frac{1}{n}\sum_{i=1}^{n}((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2) \tag{4}$$

Anschließend haben wir unseren Code für die oben gestellten Formeln wie im folgenden Link gezeigt entwickelt: `https://git.imp.fu-berlin.de/phup/robotik-uebungen/blob/master/Aufgaben/aufgabe09/Programmierung/aufgabe_3.py`.

Das Ergebnis zeigt sich dann wie folgt.

```
MAE = 4.75 cm
MSE = 24.47 cm^2
```

[1]Referenz für die Definition des absoluten und quadrierten Abstands: `https://numerics.mathdotnet.com/distance.html`