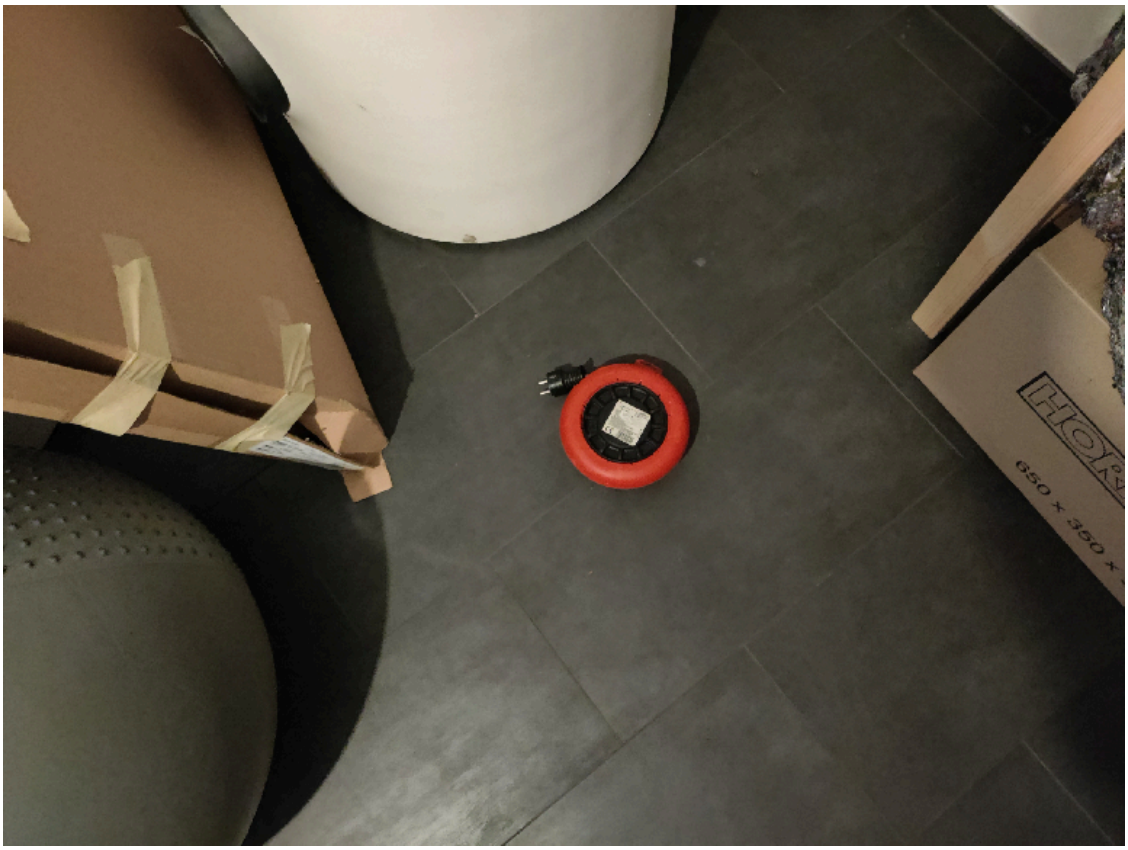
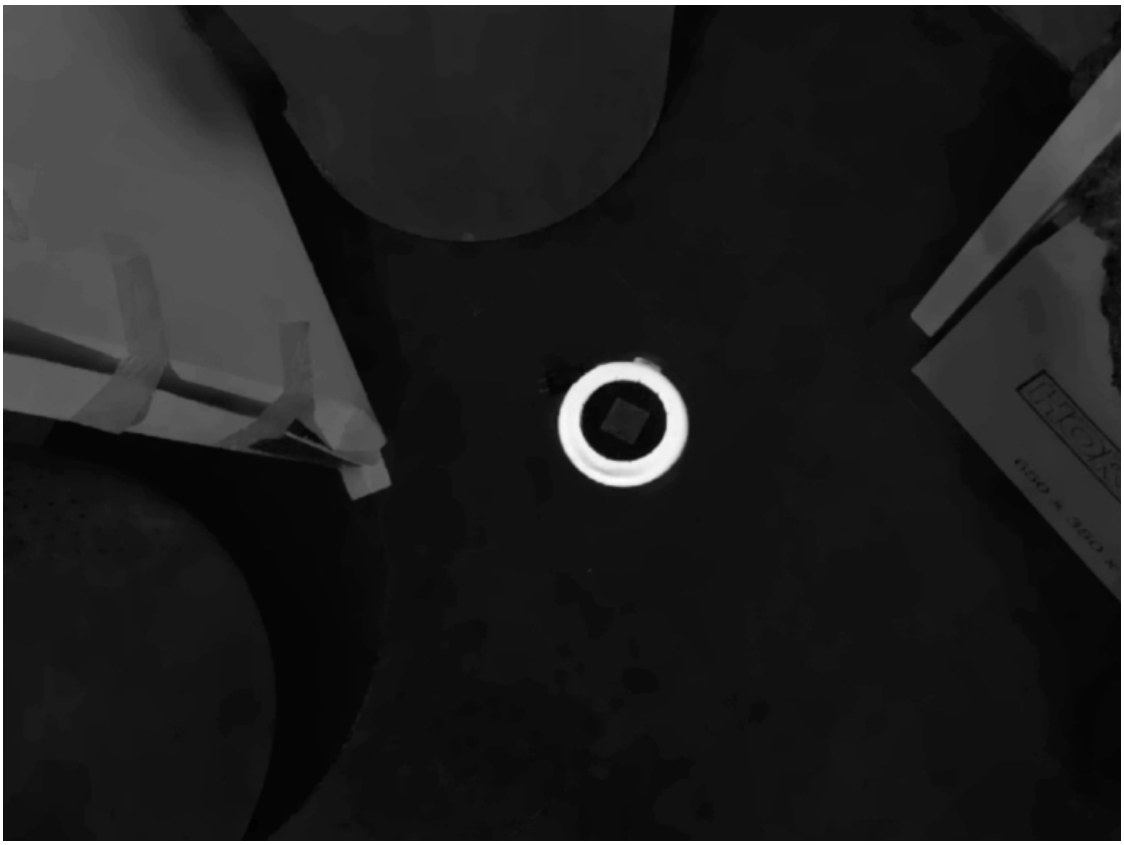
Übungsangabe

Sie haben in den letzten beiden Einheiten grundlegende Funktionen der Bildverarbeitung kennen gelernt. In dieser Hausübung müssen Sie diese selbstständig implementieren. In dieser Hausübung sollen Sie eine Funktion für einen Serviceroboter entwickeln. Ihre Aufgabe ist es einen Detektor für die rote Kabeltrommel im folgenden Bild zu entwickeln (laden Sie das Bild hier herunter für die Durchführung der Hausübung).

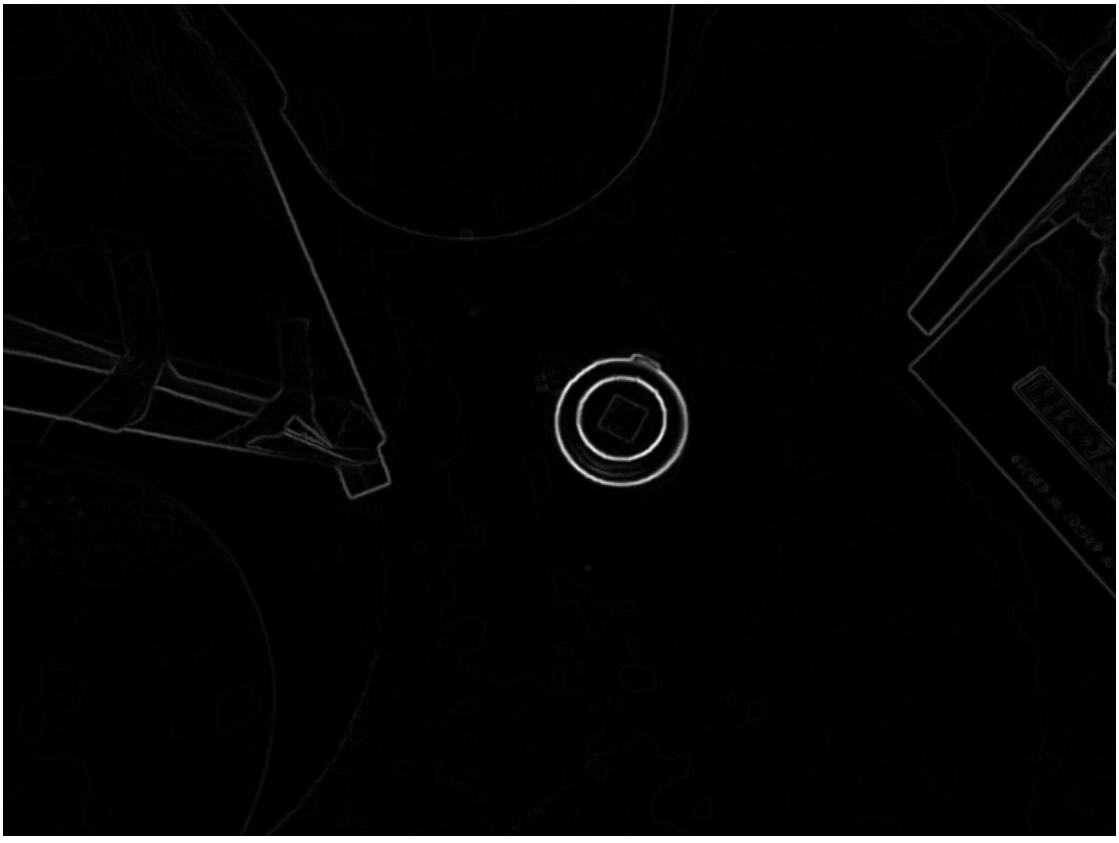


Gehen Sie dabei bitte folgendermaßen vor:

- Sie dürfen keine globalen Variablen verwenden
- Laden Sie das Bild
- Die Kabeltrommel ist farblich gut von den Umgebung trennbar. Schrieben Sie eine Funktion *segment*, welche den Kabeltrollen vom Hintergrund abhebt. Vermeiden Sie hardgecodede Parameter Ihres Ansatzes. Ein Beispiel ist in folgender Abbildung gegeben.

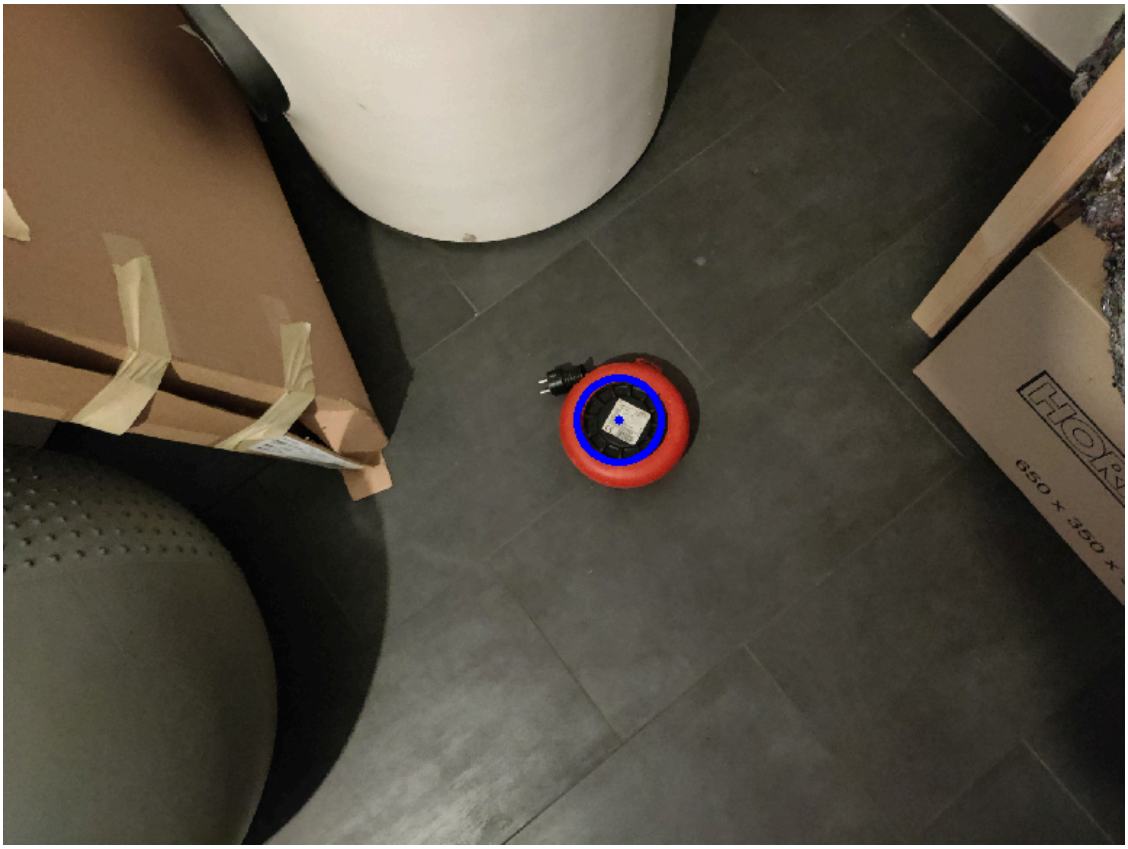


- Das segmentierte Bild muss entrauscht werden. Schreiben Sie eine C++ Klasse *discreteConvolution*. Diese Klasse soll im Konstruktor eine Kernelmatrix (*cv::Mat*) als private Member übergeben bekommen. Diese Funktion muss mindestens die Methode *conv* implementiert haben. Die *conv* Methode implementiert die diskrete Faltung. Die Methode implementiert demnach $\mathcal{I} * k = \sum_{k,l} \mathcal{I}(i+k, j+l)h(k,l)$. Diese Berechnung müssen Sie **selber** implementieren (=Summe und Produkt müssen Sie selber implementieren). Das Bild $\mathcal{I}$ ist ein Parameter der Methode. Erstellen Sie in der main Funktion eine Instanz von *discreteConvolution* aufgerufen mit einem 3×3 Gaussian Kernel.
- Sie müssen eine kantenbasierte Detektion durchführen. Erstellen Sie dazu eine C++ Klasse *sobelDetector*, welche zwei private Instanzen der *discreteConvolution* enthält (eine für Sobel in X und eine für Sobel in Y). Der Konstruktor soll die Sobel-Kernel übergeben bekommen. Implementieren Sie die Methode *getEdges*, welche ein Sobel-Kantenbild basierend auf einem Graustufenbild berechnet. Erstellen Sie eine Instanz *sobelDetector* in der main Funktion und extrahieren Sie Kanten aus dem geglätteten segmentierten Bild.



- Führen Sie OTSUs Thresholding am Kantenbild durch

- Nutzen Sie Hough Circles um die Kabeltrommel zu detektieren. Zeichnen Sie den detektierten Kreis ein. Dies ist in folgender Abbildung dargestellt.



- Ihr Programm soll das RGB Bild laden (Achtung: OpenCV kodiert wie die meisten anderen Libraries in BGR), die genannten Schritte durchführen und schlussendlich alle Bilder Zwischenergebnisse in separaten Fenster darstellen (ähnlich wie oben gezeigt).

Ihre muss aus Ihrer C++ Quelldatei und einem Makefile bestehen. Sollte das Programm nicht kompilieren/linken bekommen Sie 0 Punkte auf die Abgabe. Jede Abgabe wird einem Plagiatstest unterzogen. Sie bekommen Punkte für Codequalität (sauberes C++ programmieren, siehe MMR1 MPK) Lesbarkeit Ihrer Abgabe, Funktionsfähigkeit und Richtigkeit der Implementierung. Bei Unregelmäßigkeiten und Fragen kann es zu einem Abgabegespräch kommen. Ihre Punkte basieren dann auf diesem Abgabegespräch.

Tipps zur Durchführung der Hausübung

- Um Ihre Entwicklung zu beschleunigen empfehlen wir zunächst das Bild zu verkleinern. Dazu können Sie *cv::resize* verwenden. Ihre Abgabe muss mit dem Rohbild arbeiten.
- Die Rows, Column und Kanäle der *cv::Mat M* erhalten Sie via *M.rows*, *M.cols* und *M.channels()*
- Sie können OpenCV Datentypen mit *M.convertTo(otherM, < DT >)* konvertieren. Dies müssen Sie in dieser Übung machen um die Schritte richtig implementieren zu können. Als Datentyp stehen unter anderem *CV_32F* (float), *CV_8U* (8 bit) und *CV_64F* (double) zur Verfügung
- RGB Kanäle können Sie mit *cv::split(RGB, Mat_vec)* von der RGB Matrix *RGB* in den std. Vektor *Mat_vec* übertragen.
- *cv::minMaxIdx* liefert den Minimal- und Maximalwert einer Matrix
- Erinnern Sie sich daran, was bei einem Funktionsaufruf/Methodenaufruf mit den Parameter geschieht. Nutzen Sie für *cv::Mat* gegebenenfalls call-by-reference um Ihr Programm ev. signifikant zu beschleunigen und Fehler zu vermeiden (abhängig von Ihrer Implementierung)
- *cv::Mat::zeros(R,C,DT)* erzeugt eine $R \times C$ Matrix vom Datentyp *DT*
- Sie können mit einer *cv::Rect* Rechtecke aus Bilder ausschneiden
- *Mat1.mul(Mat2)* ist eine element-wise Multiplikation
- RGB Bilder sind 8-bit 3-kanal Bilder
- Sie können auf Matrixelemente mit *MAT.at< DT >(...)* auf die Elemente einer Matrix des Datentypen *DT* (float, double, ...) zugreifen
- *cv::namedWindow* erstellt Ihnen ein Fenster, wo Sie ein Bild anzeigen können. Erstellen Sie das Fenster mit 0 als Flag.
- Nutzen Sie 3 als dp Parameter für *HoughCircles*. Weiters gehen wir davon aus, dass wir Kreise zwischen 20 und 50 Pixel suchen.

Zuletzt geändert: Donnerstag, 2. März 2023, 14:54