Praktikum Messtechnik

Messautomatisierung und Bildverarbeitung mit Python

Verwenden der Programmiersprache Python 3 für ...

1) ... Konfiguration, Steuerung, Übetragung und PC-Visualisierung der Messdaten eines Oszillsokops nach dem IVI-Standard.
2) ... industrielle Bildverarbeitung: Bilder als Matrizen, Histogramm-Bildanalyse, Bildmanipulationen, Bildanalysen (Pixelklassifizierung, Labeling, Merkmalsextraktion)

Name: .....

Mitarbeiter(in): .....

Datum: .....

Testat: .....

Sie benötigen folgende Messgeräte für Ihren Versuch:

- Oszilloskop mit Tastköpfen und USB Schnittstellenkabel

Sie benötigen folgende Software für Ihren Versuch:

- Python 3 mit IDE Spyder
- Optional LibreOffice Calc

Bitte erscheinen Sie pünktlich und vorbereitet zum Praktikumstermin.

Führen Sie spätestens bis **sechs Stunden** vor dem Praktikumstermin den entsprechenden Vorbereitungstest auf Relax **erfolgreich** durch.

Zu dieser Praktikumsanleitung gehört ein Vorbereitungsteil, den Sie zusammen mit den Zusatzmaterial wie Datenblätter als Download auf Relax finden.

Inhaltsverzeichnis

1 Vorbereitungen auf das Praktikum....3
1.1 Messautomatisierung mit Pyhton....3
1.2 Industrielle Bildverabeitung mit Python....3
2 Durchführung des Praktikums....4
2.1 Messautomatisierung mit Pyhton....4
2.1.1 Manuelles Konfigurieren des Oszilloskops und Tastkopfabgleich....4
2.1.2 Automatisches Konfigurieren des Oszilloskops mit Python und Übertragung Messkurve auf PC....5
2.1.3 Pythonprogramm zur Erkennung eines korrekt abgeglichenen Tastkopfs....5
2.1.4 Automatisches Monitoring einer Oszilloskopmessung und Speichern deren Verlaufs in eine csv-Datei....6
2.2 Industrielle Bildverabeitung mit Python....7
2.2.1 Laden des Bildes und Ermitteln von Eigenschaften der Bildmatrix....7
2.2.2 Untersuchung der Submatrixder Rotkomponente des Bildes....7
2.2.3 Klassifizierung der Pixel der roten Legosteine....8
2.2.4 Labeling und Merkmalsextration der roten Legosteine im Bild....9
2.2.5 (Optional) Bestimmung von Position und Größe jedes erkannten roten Legosteins: Merkmalsextraktion....10

Den Abschnitt mit den Aufgabenstellungen ab Seite 4 erhalten Sie zum Praktikumstermin in ausgedruckter Form.

Lernziele

In diesem Praktikumstermin werden Sie …

... die Programmiersprache Python 3 kennen lernen, die momentan stark im Kommen ist und eine wichtige Rolle im Rapid Prototyping wie auch in der künstlichen Intelligenz spielt.

... den Umgang mit „Scientific Python“ und der IDE Spyder erlernen, wodurch Sie eine kostenlose Open Source Alternative zu MATLAB kennenlernen.

… den IVI-Standard kennenlernen, über den Messgeräte verschiedener Herstelle über verschiedene Schnittstellen einheitlich automatisiert werden.

... LabVIEW mit Python vergleichen können, wie mit beiden Programmiersprachen die selbe Automatisierungsaufgaben auf eine unterschiedliche Art und Weise gelost wird.

... lernen, wie man im Programmierhandbuch des Messgeräts die benötigten Befehle für dessen Steuerung findet und anwendet.

... lernen wie man vom PC aus Messgeräte konfiguriert, steuert und Zugriff auf deren Messdaten erhält.

... lernen, wie man die vom Messgerät abgerufenen Daten mit Python visualisiert und abspeichert.

... die Bildverabeitungsbibliothek OpenCV kennenlernen und über die Programmiersprache Python verwenden.

... lernen, wie Farb- und Grauwertbilder mathematisch als Daten im Computer dargestellt werden.

... die Analyse und einfache Manipulation von Bildern erlernen und durchführen.

... den Prozess einer Bildverabeitungskette, bestehend aus Bildvorverarbeitung, Segmentierung (Pixelklassifizierung und Labeling sowie Merkmalsextration lernen und durchführen.

1 Vorbereitungen auf das Praktikum

Bereiten Sie sich über das Durcharbeiten der Jupyter Notebooks vor, die Sie bei GitHub unter folgenden Internetadressen finden:

github.com/StefanMack/PraktMesstBV bzw. github.com/StefanMack/PraktMesstVISA

Die Jupyter Notebooks zu den Pythongrundlagen und Bildverarbeitungsgrundlagen können Sie interaktiv durcharbeiten, indem Sie diese im Webservice „Binder“ ausführen D.h. Sie können mit den Codesequenzen experimentieren, indem Sie sie ändern und testweise im Jupyter Notebook ausführen lassen. Klicken Sie bitte auf das Icon "launch|binder". **Das Starten des Webservice binder kann bis zu einer Minute dauern.**

Um das Jupyter Notebook für die Messautomatisierung anzuschauen, klicken Sie bitte im GitHub File Explorer oben auf die Datei mit der Endung ".ipynb". Ein interaktives Ausführen des Notebooks ist hier nicht möglich.

Das eigenständige Durcharbeiten aller Jupyter Notebooks ist notwendig, um die Programmieraufgaben im Praktikumstermin zu bewerkstelligen. Bitte bereiten Sie sich gut vor!

In der Vorlesung Messtechnik wird die Verwendung der Python-IDE Spyder sowie der Umgang mit den Jupyter Notebooks vorgestellt. Eine Anleitung zu Spyder finden Sie unter spyder-ide.org.

1.1 Messautomatisierung mit Pyhton

Die Inhalte dieses Praktikumsteils bauen auf die Arbeiten mit LabVIEW im vorangegangenen Praktikumstermin auf. In der Vorlesung Messtechnik werden die Grundlagen des PC-gestützen automatisierten Messens vorgestellt. Die Vorlesungsfolien dazu finden Sie auch unter Relax.

1.2 Industrielle Bildverabeitung mit Python

In der Vorlesung Sensortechnik werden die Grundlagen der industriellen Bildverabeitung vorgestellt. Dieser Praktikumsteil dient lediglich dazu, ihnen die Möglichkeiten der Bildverabeitung und die dazu im Hintergrund nötige Mathematik vorzustellen.

„Echte“ Bildverarbeitung unter Realbedingungen wie schlechte Lichtverhältnisse oder Fokussierung erfordert weit mehr Wissen.

Wenn Sie nicht bis zum Masterstudium warten möchten, können Sie sich aber auf das hier gelernte aufbauend, diese Kompetenzen im Selbststudium beibringen. Hierfür gibt es inzwischen viele Lehrbücher und sehr gute Tutorials[1]. Da wir hier nur einem „Wrapper“ der C++-OpenCV Bibliothek verwenden, können Sie zur Not auch in den C++ Dokumentationen für OpenCV nachschauen, die Namen der Methoden, Attribute usw. sind unter Python fast alle identisch.

1 Siehe z.B.: docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials

2 Durchführung des Praktikums

Verwenden Sie für die Programmerstellung, für das Debuggen und das Ausführen des Pythonprogramms die IDE `Spyder`, welche Sie auf den Laborrechnern unter **d:\MT_V3_SW** finden. Damit die Grafiken in einem gesonderten Fenster ausgegeben werden, stellen Sie bei Spyder unter `Tools > Preferences > IPython console > Graphics` das „`Graphics backend`“ auf „`Automatic`“.

Prüfen Sie für jedes Quellcodefile bitte, ob unter `Run > Configuration per file...` „`Execute in current console`“ ausgewählt ist. Denn nur dann wird der Code innerhalb der IPython-Konsole ausgeführt, so dass anschließend z.B. die Variablen im Variable explorer zum Debuggen verfügbar sind.

Loggen Sie sich mit Ihrem Hochschullogin ein und speichern Sie Ihre Arbeiten in Form von Python-Quellcodedateien auf Ihrem Desktop.

2.1 Messautomatisierung mit Pyhton

Alle nachfolgenden Aufgaben werden am selben Oszilloskop durchgeführt, welches auf Kanal 1 das Tastkopf-Testsignal aufnimmt. Die Tastköpfe sowie den kleinen Schraubendreher für deren Abgleich finden Sie oben in einem verschließbaren Fach unterhalb des Oszilloskophenkels.

2.1.1 Manuelles Konfigurieren des Oszilloskops und Tastkopfabgleich

Schalten Sie das Oszilloskop ein und schließen Sie einen Tastkopf an Kanal 1 an. Der Tastkopf wird anschließend an das Testsignal des Rechteckgenerators am Oszilloskop angeschlossen.

Stellen Sie den Trigger und die Vertikalempfindlichkeit des Oszilloskops mit der Grobeinstellung auf einen optimalen Wert ein. Wählen Sie eine passende Horizontalauflösung, so dass etwas mehr als ein Rechteckverlauf (halbe Periode) des Rechtecksignals mittig auf dem Oszilloskop sichtbar sind.

Führen Sie einen Tastkopfabgleich durch und bestimmen Sie dann über die Bedienknöpfe am Oszilloskop die Frequenz, den Mittelwert, den Spitzenwert und die Anstiegszeit des Testsignals. Tragen Sie diese Werte in die unten stehende Tabelle inkl. Einheiten ein.

Wiederholen Sie diese Messung für einen deutlich unterkompensierten sowie für einen ebenso überkompensierten Tatskopf und tragen Sie diese Werte ebenfalls in die Tabelle ein.

	Tastkopf korrekt kompensiert	**Tastkopf unterkompensiert**	**Tastkopf überkompensiert**
Überschwingen			
Pulsbreite			
Mittelwert			
U Sptize-Spitze			
Anstiegszeit			
Abfallzeit			
Vertikalempfindlichkeit (pro DIV)			
Horizontalauflösung (pro DIV)			

2.1.2 Automatisches Konfigurieren des Oszilloskops mit Python und Übertragung Messkurve auf PC

Schreiben Sie nun unter Verwendung der Bibliothek `pyVISA` ein Pythonprogramm, welches genau die selben Einstellungen am Oszilloskop wie im Abschnitt 2.1.1 vornimmt. Abschließend soll dieses Programm das aufgezeichnete abgetastete Signal grafisch als Plot darstellen.

Orientieren Sie sich hierbei an dem Codebeispiel des Jupyter Notebooks für die Vorbereitung[2]. Die nötigen Befehle für das im Praktikum verwendete Oszilloskop finden Sie im dessen Programmierhandbuch des Oszillokops, siehe PDF-Download unter Relax.

Sie haben nun mit Python das Gleiche erreicht, wie die letze Aufgabe im vorherigen Praktikumstermin, wo Sie dies mit der Software LabVIEW programmiert hatten.

Der letzte Programmschritt soll den Plot als png-Grafik abspeichern.

Speichern Sie den fertigen Quellcode unter dem Namen `OsziKonfigMessplot.py` ab.

Aufgabe: Nennen Sie je zwei Vor- und Nachteile der Pythonprogrammierung gegenüber der Verwendung von LabVIEW für die hier geforderte Aufgabe.

Tipp: *Nach einem Autoset-Befehl muss am Oszilloskop die Horizontalauflösung und die Vertikalverstärkung eingestellt werden.*
Da das Triggerereignis nach dem Autoset in Bildschirmmitte dargestellt wird, muss das Signal noch horizontal um einen „Offset“ verschoben werden.
Denken Sie daran, `sleep()`-Befehle dort einzufügen, wo das Programm auf die Reaktionszeit des Oszilloskops abwarten muss, bevor der nächste Befehl gesendet wird.

2.1.3 Pythonprogramm zur Erkennung eines korrekt abgeglichenen Tastkopfs

Es soll ein Programm erstellt werden, das über seine Textausgabe einmal alle zwei Sekunden ausgibt, ob der momentane Tastkopfabgleich korrekt, unterkompensiert oder überkompensiert ist.

Überlegen Sie sich zuerst, welche Kenngrößen des auf dem Oszilloskops dargestellten Signalverlaufs Ihnen Aufschluss darüber geben können, wie der Tastkopf kompensiert ist. Legen Sie dann für diese Größen Bereiche fest, die jeweils einem der drei Kompensationszustände zugeordnet werden. Tragen Sie ausgewählten Größen und jeweiligen Bereiche in die nachfolgende Tabelle ein.

	Tastkopf korrekt kompensiert	**Tastkopf unterkompensiert**	**Tastkopf überkompensiert**
Kenngröße 1:			
Kenngröße 2:			

2 ACHTUNG: Sie haben hier im Praktikum ein anderes Oszilloskopmodell, welches teils andere Programmierbefehle verwendet!

Erstellen Sie nun ein Programm, in dem

- das Oszilloskop konfiguriert wird,
- die ausgewählten Kenngrößen bestimmt werden und
- mit den gewählten Bereichen verglichen werden, womit
- der Kompensationszustand in Textform als True oder False ausgegeben wird.

Die nötigen Befehle für das Abrufen der gemessenen Kenngrößen aus dem Oszilloskop finden Sie im Programmierhandbuch.

Speichern Sie den fertigen Quellcode unter dem Namen `OsziKenngrTastkopf.py` ab.

Tipp: *Sie benötigen zwei ineinander verschachtelte `if`-Abfragen, die jeweils eine Kenngröße prüfen.*
Für den Kompensationszustand verwenden Sie am besten eine Boolsche Variable. Die Wiederholung der Messung erreichen Sie entweder durch eine Endlos-„`While True:`“-Schleife oder eine „`for i in range(N):`“-Schleife. Erstere Schleife nuss mit dem Stopp-Button der IPython-Konsole abgebrochen werden, letzere Schleife endet automatisch nach N Durchläufen.

2.1.4 Automatisches Monitoring einer Oszilloskopmessung und Speichern deren Verlaufs in eine csv-Datei

Erstellen Sie ein Pythonprogramm, welches jede Sekunde den Wert von U Spitze-Spitze des Tastkopfsignals abruft und diesen mit einem Zeitstempel in die (zu erstellende) Textdatei `OsziMonitor.csv` schreibt mit einem Zeilenumbruch („`\n`“) als Trennung zwischen den Datenpaaren und Kommata als Trennung innerhalb eines Datenpaars.

Das schreiben zweier Strings `x` und `y` mit Werten in eine Datei funktioniert analog zur `print()`-Funktion. Vorher muss die Datei zum Schreiben noch geöffnet werden und nachher wieder geschlossen werden:

```
datei = open('DateiName.csv','w')
datei.write('{0} {1}'.format(x,y))
datei.close()
```

Für den Zeitstempel können Sie die Pythonfunktion `strftime('%H:%M:%S')` aus der Bibliothek `time` verwenden. Sie erzeugt einen String der aktuellen Uhrzeit mit dem Format 'Stunden:Minuten:Sekunden'.

Das Programm soll nach 30 Messungen abbrechen und während der Ausführung die Nummer der aktuellen Messung ausgeben.

Speichern Sie den fertigen Quellcode unter dem Namen `OsziKenngrMonitor.py` ab.

Stellen Sie abschließend die in die Datei `OsziMonitor.csv` gespeicherten Messdaten mit dem Officeprogramm Libre Office Calc grafisch dar.

Tipp: *Verwenden Sie für das Wiederholen der Mess- und Schreibfunktion eine „`for i in range(N):`“-Scheife.*
Beim Schreibbefehl in die Datei muss zwischen den geschweiften Klammern das richtige Trennzeichen und danach ein Zeichen für den Zeilenumbruch eingefügt werden. Testen Sie dafür die Schreibbefehle in der IPython-Konsole und kontrollieren das Ergebnis im File explorer von Spyder.

2.2 Industrielle Bildverabeitung mit Python

Die nachfolgenden Aufgabenstellungen werden in der IDE Spyder ausgeführt. Die Bibliotheken (statt als Bibliothek oft auch als „Package“ bezeichnet) „opencv-python[3]“ und Numpy[4] sind auf den Rechnern im Praktikum schon installiert. Hilfreich für OpenCV ist eine Website[5], in der die C++ und Pythonbefehle gegenüber gestellt werden. Für Numpy gibt es eine übersichtliche Webseite[6], auf der die Numpy-Befehle den funktionsgleichen MATLAB-Befehlen gegenübergestellt sind.

Sie werden hier ähnliche Bildverarbeitungsoperationen wie im Jupyter Notebook für die Vorbereitung durchführen. Jedoch verwenden Sie jetzt ein anderes Bild mit der Datei `legoMitWeiss.png`, auf dem sich auch weiße Legosteine befinden.

2.2.1 Laden des Bildes und Ermitteln von Eigenschaften der Bildmatrix

Laden Sie das Bild `legoMitWeiss.png`, konvertieren Sie es, damit es mit OpenCV mit den richtigen Farben dargestellt wird, und stellen Sie das Bild in einem eigenen Fenster dar (dafür den Befehl `%matplotlib inline` aus dem Notebook weglassen).

Ermitteln Sie die Dimension der Bildmatrix, den Datentyp ihrer Elemente, sowie das größte und das kleineste Matrixelement und tragen Sie diese in die nachfolgende Tabelle ein.

Speichern Sie den fertigen Quellcode unter dem Namen `LegoShow.py` ab.

Dimension Matrix	**Datentyp Elemente**	**Kleinstes Element**	**Größtes Element**

2.2.2 Untersuchung der Submatrixder Rotkomponente des Bildes

Extrahieren Sie aus der Farbbildmatrix mittels Slicing die Submatrix für die Rotkomponete des Bildes `legoMitWeiss.png`. Stellen Sie diese Matrix als Grauwertbild inkl. einer Legende dar, die den Grauwerten die entsprechenden Pixelwerte zuordnet.

Speichern Sie den fertigen Quellcode unter dem Namen `LegoRed_1.py` ab.

Frage: Wieso können aus dem Bild der Rotkomponente nicht wie im Beispiel aus der Vorbereitung die roten Legosteine über einen Pixelschwellwert klassifiziert werden?

Antwort:

3 Siehe: pypi.org/project/opencv-python/
4 Siehe: numpy.org/
5 Siehe: Sphinx Documentation unter docs.opencv.org/3.0-last-rst/ z.B. für Methode imwrite()
siehe: docs.opencv.org/3.0-last-rst/modules/imgcodecs/doc/reading_and_writing_images
6 Siehe: mathesaurus.sourceforge.net/matlab-numpy.html

Belegen Sie auch mit einem Histogramm der Rotkomponentenmatrix, dass eine Trennung von weißen und roten Legos alleine mit dieser Bildinformation nicht machbar ist. Stellen Sie dazu das Histgramm dar und speichern Sie den fertigen Quellcode unter dem Namen `LegoRed_2.py` ab.

Frage: Wie viel Prozent der Pixel haben einen Rotwert im Intervall [200,230]? Notieren Sie nachfolgend den Pythoncode für diese Aufgabe.

Antwort:

Tipp: *Die Summe aller Elemente eines Arrays berechnen Sie mit der Numpy-Methode sum(). Welchen Bereich des Arrays sie aufaddieren, bestimmen Sie durch ein vorangestelltes Slicing.*

2.2.3 Klassifizierung der Pixel der roten Legosteine

Im vorherigen Abschnitt wurde deutlich, dass die Rotwertverteilungen der roten und weißen Legosteine überlappen, weshalb mit einer einfachen Schwellwertfunkten diese beiden Steine nicht getrennt werden können.

Überlegen Sie, mit welchem Algorithmus die roten von den weißen Steinen dennoch getrennt werden, um als Ergebnis ein Binärbild mit den Fußabdrücken der roten Steine zu erzeugen.

Tipp: *Hierzu reichen die im Jupyter Notebook zur Vorbereitung vorgestellten Bildverarbeitungsfunktionen aus. Denken Sie daran, dass die Steine mit hohem Rotanteil die Vereinigungsmenge aus den Steinen mit hohem Grauwert (weiße Steine) und den wirklich roten Steinen sind. Jeder dieser drei Mengen kann mit einem Binärbild dargestellt werden. Um zwei Bilder voneinander abzuziehen gibt es die OpenCV-Funktion* `cv2.subtract(img1,img2)`.

Erstellen und dokumentieren Sie nachfolgend zuerst Pseudocode, der den Lösungsalgorithmus enthält.

Pseudocode:

Erstellen Sie dann ein Python-Quellcode, der ein Binärbild der (wirklich) roten Steine ausgibt, und speichern Sie diesen unter dem Namen `LegoRedOnly.py` ab. In diesem Bild dürfen auch noch „Pixelspuren“ der weißen Steine enhalten sein.

2.2.4 Labeling und Merkmalsextration der roten Legosteine im Bild

Führen Sie wie in der Vorbereitung im Jupyter Notebook gezeigt ein Labeling auf des Ergebnis-Bínärbild der Klassifizierung aus dem vorherigen Abschnitt durch.

Speichern Sie den Quellcode unter dem Namen `LegoRedLabel_1.py` ab.

Frage: Wieso funktioniert hier das Labeling nicht so wie in der Vorbereitung? Was für eine Bildverarbeitungsfunktion wäre hier nötig?

Antwort:

Um kleinere Pixelbereiche, die oft durch Rauschen verursacht werden, aus einem Binärbild zu entfernen, wendet man die sogenannte morphologische Operation „Erosion“ (dt. Erosion) an. Dies ist ein auf Rechenzeit optimierter Algorithmus, welcher Pixelbereiche entfernt, die nicht zu mindestens zwei Seiten Nachbarpixelbereiche besitzen. Wie große diese Pixelbereiche mindestens sein müssen, wird durch eine Einheitsmatrix festgelegt, die als „Kernel“ bezeichnet wird. Wenn wie hier die störenden Pixelbereiche Flächen kleiner 2x2 aufweisen, dann reicht eine 2x2-Einheitsmatrix als Kernel.

Bereinigen Sie also das Binärbild mit folgender Bildverarbeitungsmethode:

```
import numpy as np
kernel = np.ones((2,2),np.uint8)
imgRealRedErode = cv2.erode(imgRealRed,kernel,iterations = 1)
```

Für den umgekehrten Fall, dass innerhalb einer weißen Fläche einzelne schwarze Pixel zu entfernen sind, wendet man die morphologische Operation „Dilation“ (dt. Dilatation) an. In der industriellen Bildverarbeitung werden diese beiden Funktion kombiniert nacheinander ausgeführt. Sie heißen „Opening“ bzw. „Closing“[7].

Führen Sie ein Erodieren durch, um die störenden Pixelbereiche zu entfernen und labeln Sie das resultierende Binärbild erneut. Geben Sie in der Textausgabe die Zahl der gefundenen Legosteine an. Speichern Sie den Quellcode hierzu unter dem Namen `LegoRedLabel_2.py` ab.

7 Siehe:
docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops

2.2.5 (Optional) Bestimmung von Position und Größe jedes erkannten roten Legosteins: Merkmalsextraktion

In den vorangegangenen Abschnitten haben Sie die Pixel der roten Legosteine klassifiziert und deren Fußabdrücke gelabelt.

Nun soll für jeden Legostein dessen Pixelschwerpunkt und dessen Fläche bestimmt werden. Die Fläche soll als Anzahl im Bild beanspruchter Pixel angegeben werden. Alle diese Merkmale sollen per Textausgabe ausgegeben werden.

Schreiben Sie hierfür einen Code, den Sie unter dem Namen `LegoRedFeat.py` abspeichern, und füllen Sie die nachfolgende Tabelle mit den Ergebnissen daraus aus.

Legostein Nr.	**x-Position (Pixel)**	**y-Position (Pixel)**	**Fläche (Pixel)**
1			
2			
3			
4			
5			
6			

Frage: Könnten Sie jetzt auch die Zal der Noppen für jeden Stein ausgeben?

Antwort:
