

HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN

Abteilung für Elektronik – Technische Informatik

Klasse/ Jahrgang: 5BHEL	Gruppe: 5b	Übungsleiter: Prof. Wihsböck
Übungsnummer: II/6	Übungstitel: Bildverarbeitung	
Datum der Übung: 22.12.2015	Teilnehmer: Gorke, Haschke, Hogl, Mayerhofer	
Datum der Abgabe: 19.01.2016	Schriftführer: Haschke	Unterschrift:

	Beurteilung
Aufgabenstellung	
Dokumentation	
Messschaltungen	
Messtabellen	
Berechnungen	
Programmlistings	
Auswertung	
Diagramme	
Berechnungen	
Simulationen	
Schlußfolgerungen	
Kommentare	
Inventarliste	
Messprotokoll	
Form	
Summe	

Allgemeiner Teil

Titel der Übung: Bildverarbeitung
Übungsnummer: II/6
Übungsplatz: PRT
Datum der Übung: 22.12.2015
Klasse: 5BHEL
Schriftführer: Haschke
Übungsteilnehmer: Gorke
Haschke
Hogl
Mayerhofer

Inhaltsverzeichnis

1. Bildverarbeitung mit Eclipse	3
1.1 Projekte in Eclipse.....	3
1.2 Programmieren von Filtern	3
1.2.1 Invertierung	4
1.2.2 Helligkeit	5
1.2.3 Grenzwert	6
1.2.4 Medianfilter	7
1.2.5 Lineare Filter	8
1.3 Weitere Funktionen von ImageJ	9
1.3.1 Zählen von Objekten	9
1.3.2 Längenmessung	10
1.3.3 Flächenmessung.....	11
2. Bildverarbeitung über Raspberry Pi	12
2.1 LED Blinken über Simulink	12
2.2 VideoInverting	12
3. Erkenntnisse	13
4. Inventarliste	14
5. Messprotokoll	15

1. Bildverarbeitung mit Eclipse

1.1 Projekte in Eclipse

Zu Beginn der Übung wurde ImageJ, welches in Form eines Java-Projektes vorhanden war, in Eclipse eingebunden und gestartet. Dafür wurde zuerst ein neuer „Workspace“ im Workspace Launcher beim Starten von Eclipse erstellt. Nun wurde das entpackte Projekt in den Workspace-Ordner kopiert. Danach wurde das ImageJ-Projekt über „File -> Import -> General -> Existing Projects into Workspace“ und im folgenden Fenster dann „Select root directory -> Browse -> (Name des erstellten Workspace)“ das Projekt eingebunden. Dazu muss nach dem Auswählen des Workspace das vorhandene Projekt einfach angehakt werden und dann wird über den „Finish“-Button bestätigt.

Nun wurde der Launcher konfiguriert. Dafür wird das Projekt mit Rechtsklick angeklickt und „Run As -> Run Configurations...“ ausgewählt. Hier wird zuerst der Name auf ImageJ-Plugins geändert. Danach wird die erste Zeile der Datei „JVM-Arguments.txt“ im Unterpunkt „Arguments“ unter „VM Arguments“ hineinkopiert. Über „Apply -> Close“ werden die Einstellungen bestätigt.

ImageJ – Punkt- und Filteroperationen werden in Java geschrieben. ImageJ selbst ist ebenfalls nur ein Java Projekt.

1.2 Programmieren von Filtern

Ziel dieser Programmieraufgaben war es die grundlegende Erstellung von ImageJ-Plugins zu erlernen und Java Kenntnisse gegebenenfalls etwas aufzufrischen.

Ein Filter wird erstellt in dem man „default-package“ mit Rechtsklick anklickt und dann über „New -> Class“ eine neue Klasse erstellt. Obwohl es nicht der Java-Namensvergabe entspricht müssen ImageJ-Plugins einen „_“ im Namen enthalten von ImageJ erkannt zu werden. Im folgenden Fenster sowohl der Name, als auch unter „Interface -> Add“ „PluginFilter“ ausgewählt und hinzugefügt werden.

Wird ImageJ beendet während noch Bilder geöffnet sind können diese auch über Eclipse wieder geschlossen werden. Dazu wird in Eclipse am unteren Rand im „Console“-Fenster einfach der „Terminate“-Button gedrückt.

1.2.1 Invertierung

Unsere erste Aufgabe war es eine Punktoperation zu programmieren, welche die Pixel eines 8-bit Graustufenbildes invertiert. Dafür muss das Bild lediglich Pixel für Pixel durchgegangen und jeweils 255, der Maximalwert eines Pixels, minus seinem jetzigen Wert gerechnet werden.

```
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class PointOp_Inverter implements PlugInFilter
{
    @Override
    public int setup(String arg, ImagePlus imp)
    {
        return DOES_8G;
    }

    @Override
    public void run(ImageProcessor original)
    {
        int width = original.getWidth();
        int height = original.getHeight();

        for(int x = 0; x < width; x++)
        {
            for(int y = 0; y < height; y++)
            {
                int PixelValue = original.getPixel(x, y);
                int newPixelValue = 255 - PixelValue;
                original.putPixelValue(x, y, newPixelValue);
            }
        }
    }
}
```

Der Return-Wert DOES_8G liefert ein 8-bit Graustufenbild. Die Funktionen original.getWidth() und original.getHeight() liefern jeweils Breite und Höhe des Bildes, wobei original das Originale Bild ist, welches der Funktion übergeben wird. Die beiden Schleifen gehen nun das Bild Pixel für Pixel durch und invertieren dabei jeden Pixel. Zum Auslesen des Pixel-Wertes wird dafür die Funktion original.getPixel() genutzt, wobei die angegebenen Zahlen der x- und y-Koordinate des Pixels entsprechen. Mit original.putPixel() wird der neue Wert gesetzt.

1.2.2 Helligkeit

Nun sollte die Helligkeit jedes Pixels im Bild um einen fixen Wert (in dieser Übung 50%) erhöht werden. Dafür wird ähnlich wie bei der Invertierung vorgegangen. Anstatt den Pixel zu invertieren wird bei dieser Punktoperation jedoch die Hälfte des aktuellen Wertes addiert, wodurch die gewünschten +50% Helligkeit erreicht werden. Anzumerken ist das der Wert 255 Weiß und 0 Schwarz entspricht.

```
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class PointOp_Inverter implements PlugInFilter
{
    @Override
    public int setup(String arg, ImagePlus imp)
    {
        return DOES_8G;
    }

    @Override
    public void run(ImageProcessor original)
    {
        int width = original.getWidth();
        int height = original.getHeight();

        for(int x = 0; x < width; x++)
        {
            for(int y = 0; y < height; y++)
            {
                int PixelValue = original.getPixel(x, y);
                int newPixelValue = 255 - PixelValue;
                original.putPixelValue(x, y, newPixelValue);
            }
        }
    }
}
```

1.2.3 Grenzwert

Die letzte Aufgabe zu den Punktoperationen war es ein 8-bit Graustufenbild anhand eines Grenzwertes in ein Schwarz-Weiß Bild umzurechnen. Dafür wird wieder jeder Pixel-Wert eingelesen, nur wird dieses Mal der Wert des Pixels mit einem vorher definierten Grenzwert verglichen, liegt der Wert des Pixels darüber wird der Pixel auf 255 (Weiß) gesetzt, liegt er darunter wird der Pixel auf 0 (Schwarz) gesetzt.

```
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class PointOp_Threshold implements PlugInFilter
{
    public static final int THRESHOLD = 92;

    @Override
    public int setup(String arg, ImagePlus imp)
    {
        return DOES_8G;
    }

    @Override
    public void run(ImageProcessor original)
    {
        int width = original.getWidth();
        int height = original.getHeight();
        int newPixelValue = 0;

        for(int x = 0; x < width; x++)
        {
            for(int y = 0; y < height; y++)
            {
                int PixelValue = original.getPixel(x, y);
                if(PixelValue < THRESHOLD)
                {
                    newPixelValue = 0;
                }
                else
                {
                    newPixelValue = 255;
                }
                original.putPixel(x, y, newPixelValue);
            }
        }
    }
}
```

Zum Definieren der Konstanten wurden die Schlüsselwörter „static“ und „final“ verwendet. „final“ kennzeichnet die Eigenschaft der Klasse als eine Konstante, während „static“ angibt das diese Eigenschaft nicht bei jedem Aufruf dieser Klasse neu angelegt werden muss, sondern nur beim ersten Aufruf angelegt wird.

1.2.4 Medianfilter

Als nächstes sollte eine Filteroperation, im speziellen ein Medianfilter programmiert werden. Bei einem Medianfilter werden jeweils das ausgewählte Pixel und alle 8 umliegenden in ein Array gespeichert. Dieses wird dann den Werten nach sortiert und der Wert in der Mitte des Arrays wird als neuer Wert genommen. Zur Vereinfachung wurde der äußere Rand des Bildes weggelassen. Überprüft wird der Filter, in dem man ein sogenanntes „Salz- und Pfefferrauschen“ zum Bild hinzufügt. In ImageJ wird dieses über „Process -> Noise -> Salt and Pepper“ hinzugefügt.

```
import java.lang.reflect.Array;
import java.util.Arrays;

import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class FilterOp_3_3_Median implements PlugInFilter
{
    @Override
    public int setup(String arg, ImagePlus imp)
    {
        return DOES_8G;
    }

    @Override
    public void run(ImageProcessor original)
    {
        ImageProcessor copy = original.duplicate();
        int width = original.getWidth();
        int height = original.getHeight();
        int [] filterArray = new int [9];

        for(int x = 1; x < width-1; x++)
        {
            for(int y = 1; y < height-1; y++)
            {
                filterArray[0] = copy.getPixel(x-1, y-1);
                filterArray[1] = copy.getPixel(x, y-1);
                filterArray[2] = copy.getPixel(x+1, y-1);
                filterArray[3] = copy.getPixel(x-1, y);
                filterArray[4] = copy.getPixel(x, y);
                filterArray[5] = copy.getPixel(x+1, y);
                filterArray[6] = copy.getPixel(x-1, y+1);
                filterArray[7] = copy.getPixel(x, y+1);
                filterArray[8] = copy.getPixel(x+1, y+1);
                Arrays.sort(filterArray);
                original.putPixelValue(x, y, filterArray[4]);
            }
        }
    }
}
```

1.2.5 Lineare Filter

Die letzte Programmieraufgabe bestand darin mit der schon vorhandenen Implementation eines Linearen Filters durch Variieren der Parameter verschiedene Lineare Filter zu testen. Dieses Variieren der Parameter geschah durch ändern der Werte in dem zweidimensionalen Array „filter matrix“.

```
public class Filter_ArbitraryMxN implements PlugInFilter
{
    public int setup(String arg, ImagePlus imp)
    {
        return DOES_8G;
    }

    public void run(ImageProcessor original)
    {
        int w = original.getWidth();
        int h = original.getHeight();

        //filter matrix
        int[][] filter =
        {
            {0,1,2,1,0},
            {1,2,5,2,1},
            {2,5,9,5,2},
            {1,2,5,2,1},
            {0,1,2,1,0}
        };
        int calcSum = calcFilterSum(filter);
        double s = 1.0 / calcSum;
        int K = filter[0].length/2;
        int L = filter.length/2;

        ImageProcessor copy = original.duplicate();

        for (int y = L; y <= h - L - 1; y++)
        {
            for (int x = K; x <= w - K - 1; x++)
            {
                // compute filter result for position (x,y)
                int sum = 0;
                for (int j = -L; j <= L; j++) {
                    for (int i = -K; i <= K; i++) {
                        int p = copy.getPixel(x + i, y + j);
                        int c = filter[j + L][i + K];
                        sum = sum + c * p;
                    }
                }
                int q = (int) (s * sum);

                // clamp result
                if (q < 0) q = 0;
                if (q > 255) q = 255;
                original.putPixel(x, y, q);
            }
        }
    }
}
```

```

private int calcFilterSum(int filter[][])
{
    int sum=0;

    for(int i=0;i<filter.length;i++)
    {
        for(int j=0;j<filter[i].length;j++)
        {
            sum = sum+filter[i][j];
        }
    }

    return sum;
}
}

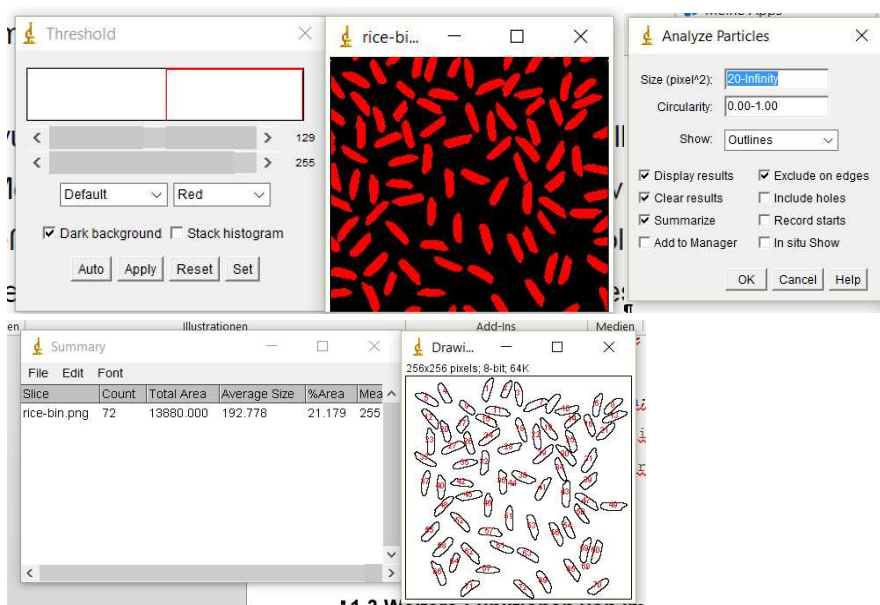
```

1.3 Weitere Funktionen von ImageJ

Nun wurden die verschiedenen Funktionen noch zum Zählen und Vermessen von Objekten in Bildern verwendet.

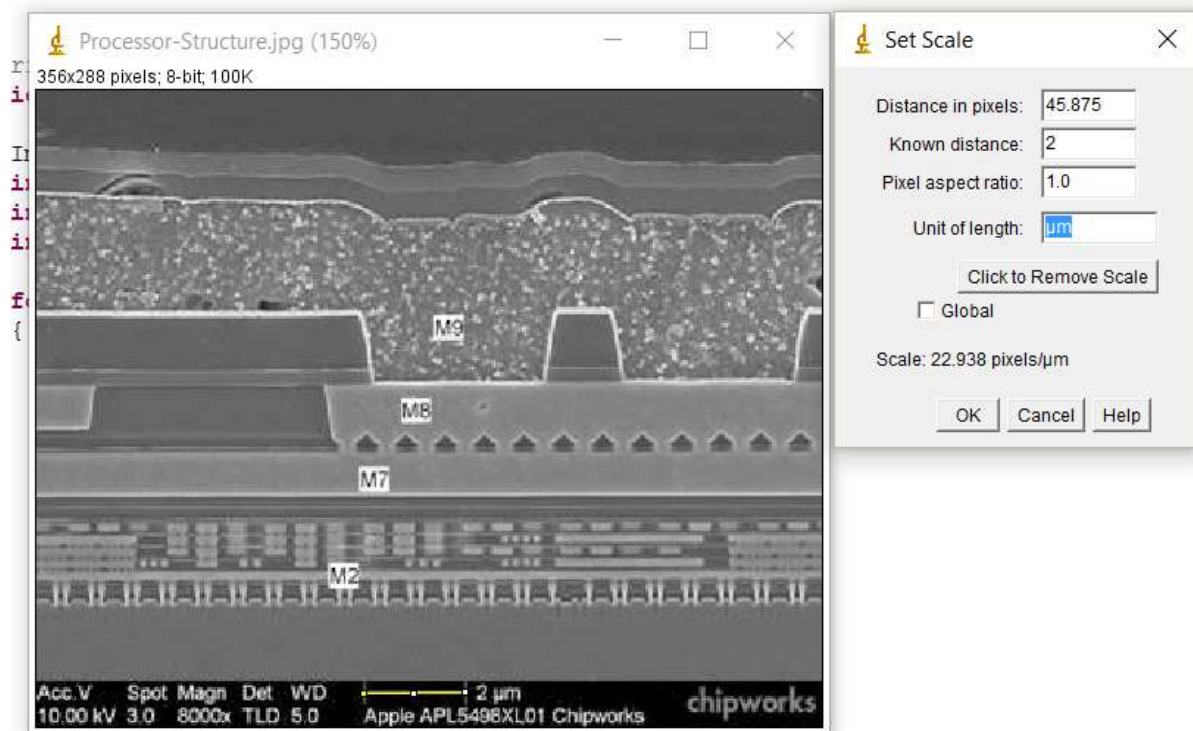
1.3.1 Zählen von Objekten

Die Zählfunktion wurde an einem Schwarz-Weiß Bild von Reiskörnern getestet. Dafür musste dieses jedoch erst in ein 8-bit Graustufenbild umgewandelt werden. Dies kann über „Image -> Type -> 8-bit“ erledigt werden. Über „Image -> Adjust -> Threshold“ kann der Grenzwert, ab welchem Objekte als solche erkannt werden eingestellt werden. Da das Bild helle Objekte auf dunklem Hintergrund hat, wird zusätzlich noch „Dark Background“ ausgewählt. Zum Zählen der Objekte wird die Funktion „Analyze -> Analyze Particles“ verwendet. Hier kann zusätzlich noch die Größe der Objekte, welche gezählt werden sollen, vorgegeben werden. Bei dieser Übung wurde die minimale Größe auf 20 Pixel gesetzt um kleine Bildfehler bei der Zählung auszuschließen. Weiters werden mit „Exclude on edges“ Objekte die den Rand des Bildes berühren ebenfalls ausgeschlossen. Außerdem kann man mit der Option „Show -> Outlines“ die Ränder der gezählten Objekte anzeigen lassen.



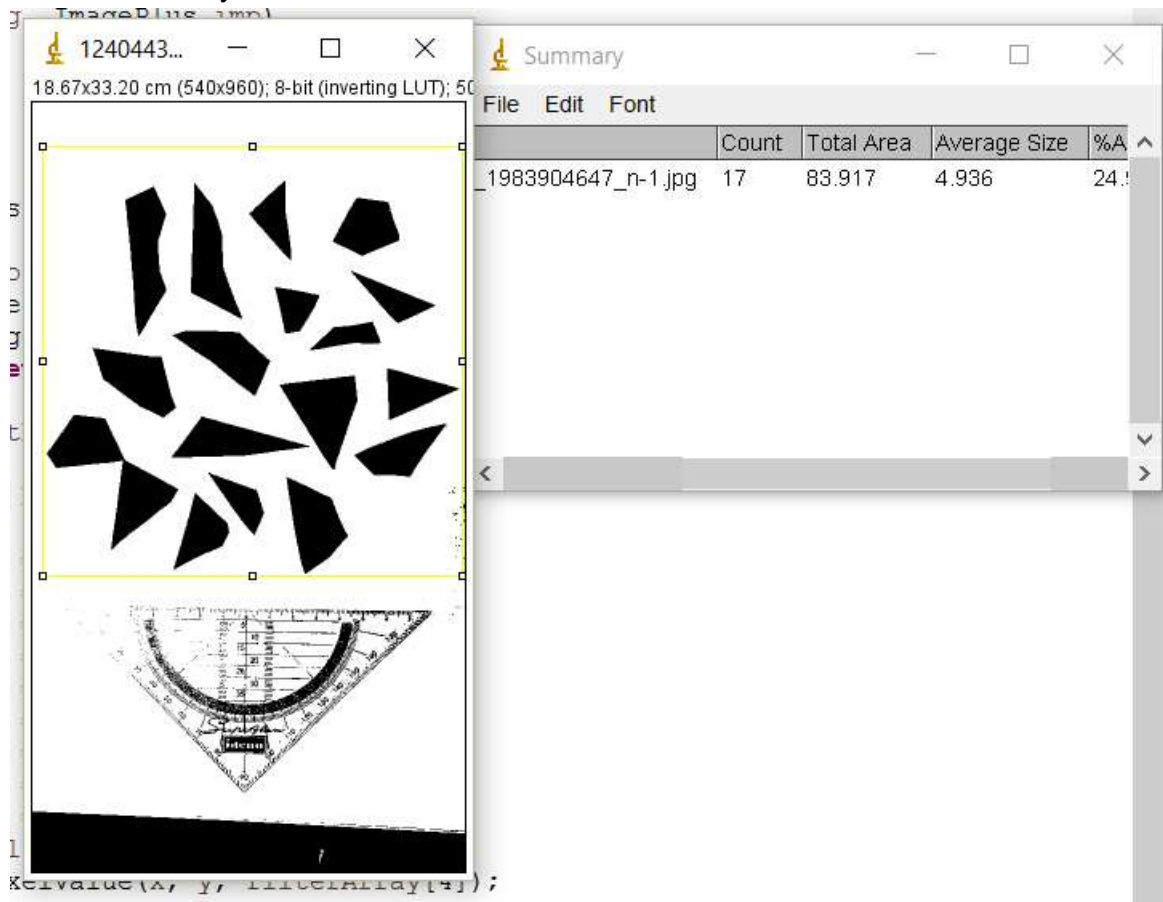
1.3.2 Längenmessung

Als nächstes wurde auf einem Bild eines Prozessorquerschnitts eine Längenmessung durchgeführt. Dafür wurde der auf dem Bild vorhandene Maßstab mit der Funktion „Straight“, einer geraden Linie, nachgezeichnet. Anschließend wird mit „Set Scale“ der Maßstab gesetzt.



1.3.3 Flächenmessung

Nun sollte mit ImageJ noch eine Flächenmessung durchgeführt werden. Dazu wurde ein 9x9 cm großer Zettel zerschnitten, die Teile zusammen mit einem Maßstab (Lineal, Geodreieck) ab fotografiert und anschließend wurde das Bild mit ImageJ geöffnet. Zuerst wird der Typ wieder auf 8-bit Graustufenbild geändert. Anschließend werden dieselben Einstellungen wie schon beim Zählen der Objekte wieder vorgenommen. Wie schon bei der vorherigen Aufgabe wird der Maßstab erneut mit „Straight“ nachgezeichnet mit „Analyze -> Set Scale“ gesetzt. Mit der „Rectangular“ Funktion von ImageJ kann man nun einen rechteckigen Bereich definieren, in welchem die Messung durchgeführt wird und über Alt + „Rectangular“ kann man Teile dieses Bereichs, wie zum Beispiel das Lineal, exkludieren. Weiters muss man unter „Analyze -> Set Measurements“ noch „Area“ auswählen.



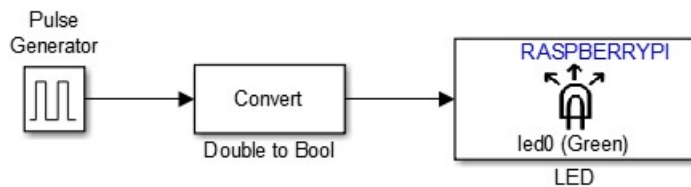
Wie im Bild zu erkennen ist beträgt die von ImageJ gemessene Fläche 83,917 cm². Da der Zettel 9x9 cm groß war, sollte eine Fläche von 81 cm² vorhanden sein.

2. Bildverarbeitung über Raspberry Pi

2.1 LED Blinken über Simulink

Dieser Teil der Übung bestand darin im Simulink ein Model zu erstellen, welches die LED0 auf dem Raspberry Pi zum Blinken bring.

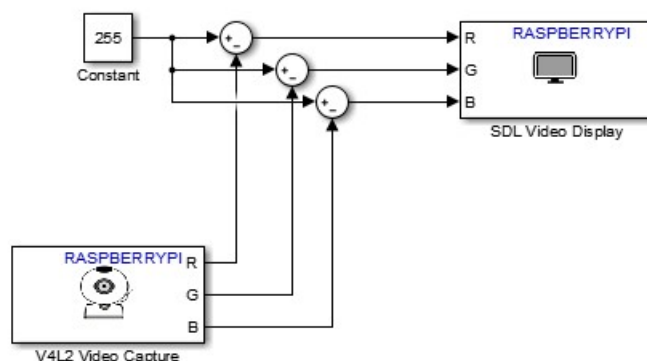
Dazu wurde ein Pulsgenerator mit einem Conversion-Block verbunden. Dieser wandelt den Datentyp des Signals von „double“ auf „bool“ um. Der darauf folgende Block steuert die LED auf dem Raspberry Pi an. Über „Tools -> Run on Target Hardware -> Prepare to Run“ können nun letzte Einstellungen vorgenommen werden. Hier werden die IP-Adresse des Raspberry Pi, welcher über ein Ethernet Kabel mit dem Schulnetz verbunden wurde, sowie der Username und das Passwort eingetragen. Sind mehrere passende Boards verfügbar kann hier das Gewünschte ausgewählt werden.



2.2 VideoInverting

Als letzte Aufgabe wurde nun ein Model entworfen welches über eine kompatible Webcam ein Video aufnimmt, invertiert und über den Raspberry Pi auf einem Bildschirm ausgibt. Ähnlich wie bei der Programmieraufgabe Invertierung wird hier jeweils der Maximalwert eines Farbkanals (255) weniger dem aktuellen Wert gerechnet. Wichtig ist das sowohl der „Video Capture“ als auch der „Video Display“ Block denselben Bild-Typ eingestellt haben. Als Auflösung wurde 160 x 120 Pixel und als Sample-time 0,1s gewählt.

Aufgrund anfänglicher Komplikationen wurde das Model über die Funktion „Deploy on Hardware“ gestartet. Dafür muss die Dauer auf „inf“ (Infinite, Unendlich) gestellt werden. Nach einem Reboot des Raspberry Pi lief das Model jedoch auch über die „Run on Target Hardware“ Funktion.



3. Erkenntnisse

In dieser Übung wurden uns die Grundlagen der Bildverarbeitung beigebracht. Weiters wurde das Arbeiten mit Java etwas aufgefrischt und das Arbeiten mit Eclipse und ImageJ besprochen und erlernt. Außerdem wurde Java im allgemeinen besprochen.

4. Inventarliste

5. Messprotokoll