Handbuch und Funktionsbeschreibung

VisionSoftware PatControl

Software Manual

FlexxVision
HighSpeed VisionSystems

FlexxVisision Pillnitzer Weg 11 D-13593 Berlin, Germany Tel: **+49 (030) 120 102 26** Fax: **+49 (030) 364 044 20** www.FlexxVision.de	Autor: Karsten Michael Schulz 22.12.2014 Dokumentation [Status] Teil1	

FlexxVision
HighSpeed VisionSystems

Inhalt

1.0.0 Über dieses Handbuch und der Software

Willkommen zum Handbuch und der Funktionsbeschreibung der Software **PatControl** von **FlexxVision**. Die Software hat zur Aufgabe Maschinen das „sehen“ zu ermöglichen, und erkannte Objektinformationen schnellstmöglich an eine Prozessleitstelle zu übertragen. Das sind Vorgänge die bisweilen mit hochspezialisierten Einzellösungen und der Beteiligung vieler externer Softwareteile anderer Anbieter ermöglicht wird.

Wenn sich die Erkennungs- Bedingungen einer Maschine verändern weil Teile sich im Zuge der Weiterentwickelung ändern, muss nicht selten die vorhandene Bilderkennung umgebaut oder angepasst werden, Produktionsstopps sind die Folge. **PatControl** entschärft diese Situation, nicht nur deswegen weil alle Komponenten der Software von **FlexxVision** entwickelt wurden und keine fremden Softwarekomponenten bis hin zur Prozessleitstellen – Kommunikation Bestandteil des Konzeptes sind, sondern weil es ermöglicht wird zur Produktionslaufszeit ohne Anhalten der Maschinenprozesse Änderungen am laufenden Programm durchzuführen zu können So kann direkt auf die Bilderkennung gewirkt werden ohne kostspielige Maschinenpausen bei jeder Teile -Änderung einzuräumen.

Jeder Ingenieur ist in der Lage mit der eingebauten Skriptsprache **LUA** eigene Prüfpläne zu erstellen, und auch die Benutzereingabe –Felder für den Operator/Anwender Anzulegen sowie diese mit Hinweistexten transparent darzustellen. Selbstverständlich können wir hier mit langer Erfahrung diese Aufgaben unterstützen, oder ganze Messabläufe in Form angepasster Prüfpläne liefern.

In den folgenden Abschnitten möchten wir die Verfahren und detaillierten Vorgänge der Software –Konfiguration und Prüfplanung beschreiben so dass Sie auch selber befähigt werden mehr Kontrolle über Ihre Prozesse zu erlangen. Das gesamte Konzept der Software ist die freie Konfigurierbarkeit und Anpassungsfähigkeit so dass für fast jede Aufgabe eine Lösung implementiert werden kann.

Die Installation liefert Beispielprojekte aus denen hervorgeht, wie die Prüfplandetails aufgebaut sind, so können daraus eigene Messabläufe abgeleitet werden.

Zudem finden sich im SDK –Bereich Kompilierbare Projekte für die Microsoft Entwicklungsumgebung **VisualStudio** 2013, die aufzeigen wie mit Ihrer bestehenden Software eine direkte Datenverbindung zum **VisionServer PatControl** hergestellt werden kann. Die Software ist in der Lage auch autark im Hintergrund abzulaufen und lediglich mit der Prozessleitstelle zu kommunizieren. Oder gänzlich alleine Daten zu erheben, um eine statistische Datenverfolgung parallel zu einem bestehen Vision System zu Qualifizieren.

1.0.1 Systemvoraussetzungen

FlexxVision stellt mit dieser Software erhebliche Systemanforderungen an das **32/64Bit** Windows System, bei umfangreichen Rotationsprüfplänen mit Echtzeit 3D –Darstellung. Aber auch mit geringeren System -Anforderungen bei kleinen Messaufgaben die bereits auf einem 2Kern embedded Schaltschrank -PC ablaufen, kann die Software betrieben werden.

Unterstützte Betriebssysteme:
Microsoft Windows 8
Microsoft Windows 7
Microsoft Windows Vista (SP2)

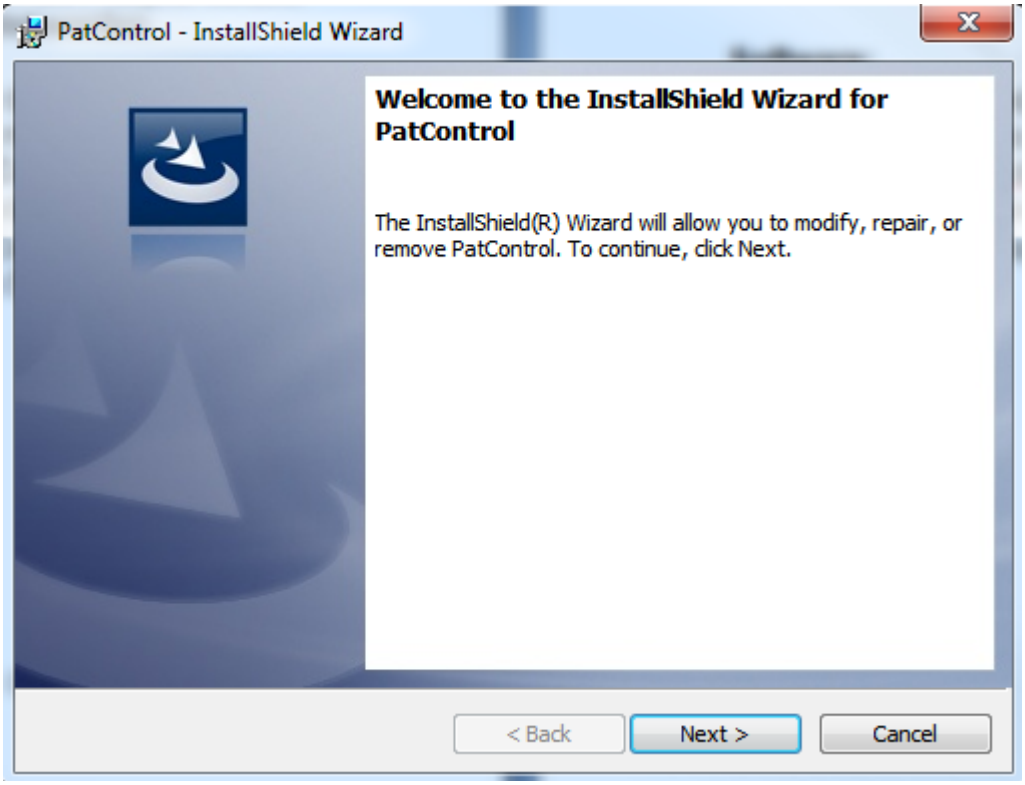
Hardware Maximal:
3000 MB Arbeitsspeicher
2000 MB freien Festplattenspeicherplatz
Netzwerkkarte (Intel Chipsatz)
3D Leistungs-Grafikkarte (2000 MB)
Intel i7 4-6 Kern 3.5Ghz
Monitor 1920x1200
Sound Mono (Text to Voice via LUA)

Hardware Minimal:
1000 MB Arbeitsspeicher
500 MB freien Festplattenspeicherplatz
Netzwerkkarte (On Board)
2D Leistungs-Grafikkarte (256 MB)(On Board)(3D ist dann offline)
AMD Athlon 2Kern 1.9Ghz

Software:
Microsoft Internet Explorer 9 oder höher
Microsoft Excel (optional)
Microsoft VisualStudio 20013 (optional)
Texteditor (zebu. Ultraedit optional)
TeamViewer (Desktop Fernwartung optional)

1.0.2 Installation von PatControl

Die Installation erfolgt über einen Standard Windows Installer (**InstallShield**) der Vorgang Ist bekannt und Voraussetzung grundliegender Systemhandhabe. Die Datei die alle Daten enthält lautet **PatControl.msi** ein Doppelklick löst den Installationsvorgang aus.



Im Anschluss besteht die Möglichkeit den Speicherort konform zu verändern, für Wartung und Support wird die Voreinstellung empfohlen.

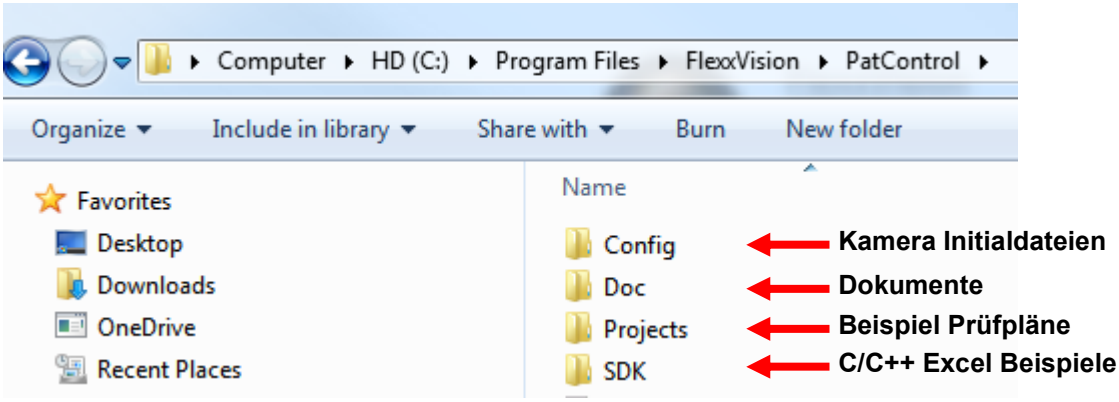
Am Ende der Installation wird gefragt, ob die Anwendung gestartet werden soll, dies wird empfohlen da hierdurch weitere Ersteinstellungen im Hintergrund erfolgen, und die Software Legitimation erfolgen kann.

Für die Deinstallation können sie das Setup erneut anklicken oder im Programme –Menü als auch über die Systemsteuerung Die Software entfernen. Ein Update kann einfach erneut über die bestehende Software installiert werden, das letzte Datum können sie auf unserer Downloadseite entnehmen.

1.0.3 Software Verzeichnisse

Nach der Installation hat der Setupprozess einige Verzeichnisse am Zielort angelegt. Das Wurzelverzeichnis der Bestandteile trägt den Namen **FlexxVision** und darin befindet sich Verzeichnis das den Produktnamen darstellt **PatControl**

Folgende Verzeichnisstruktur bildet sich ab:



Insbesondere das Verzeichnis **Config** enthält die Dateien die notwendig sind um eine Kamera auszuwählen, der Vorgang wird unter **Kameras** beschrieben

Im Verzeichnis **Doc** befindet sich auch diese Dokumentation als PDF Datei.

Das Verzeichnis **Projects** enthält viele Beispiele und Bilder/Videos zum nachstellen von Prüfplanungen mit Skripten für LUA und benutzerdefinierte Parameterbäume.

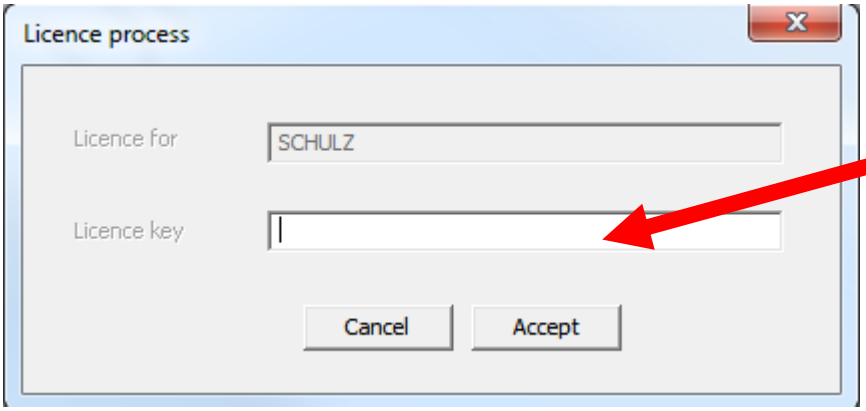
Im Verzeichnis SDK finden sich Beispiele für die Integration der Software **PatControl** in bestehende Projekte, auch ein **Excel VBS** Beispiel ist dabei um Logbuchdateien zu visualisieren es wird empfohlen diese für die Logbücher einzusetzen.

Für die Herstellung eines eigenen Prüfplanes kopieren sie ein Beispiel das am besten für Ihre Aufgabe geeignet scheint, und arbeiten sie mit dieser Kopie. Dies gilt generell auch für die Kamera Initalisierungsdateien.

1.0.4 Software Legitimation

Die Softwarelegitimation erfolgt nur in der freien Version, bei einem Programmstart wird die Software einen Dialog anzeigen indem sie Ihren Software –Schlüssel eingeben den sie über die Online Registrierung per E-Mail erhalten haben.

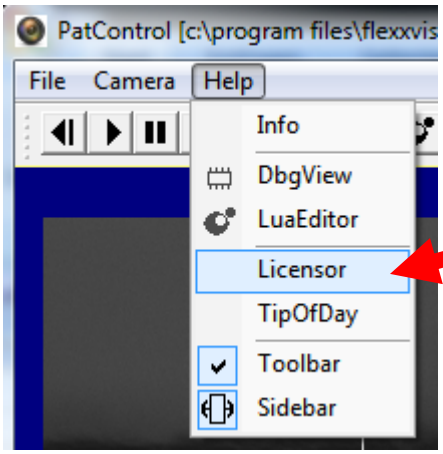
Ohne diese Legitimation funktioniert **PatControl** dennoch, und fragt gelegentlich nach einer Registrierung .



Softwareschlüssel Eingabe

Nach einem Klick auf **Accept** wir der der Request über den Standard WebPort 80 an den **FlexxVision** Server übertragen und dort die Software in einer Datenbank verzeichnet.

Wir wissen dass in einer Produktionsstätte oder im Prüffeld keine Verbindung zum Internet besteht, und bieten für die Vollversion einen Dongle von **Wibu KeySystems** an, dem führenden Hersteller von Softwarelegitimationen. Den Dongle können sie separat über das Kontakt -Formular oder per E-Mail sowie telefonisch anfordern um damit die Software gegen Rechnungslegung zu erwerben. Den Lizenzdialog können sie auch später über das Menü Erreichen.



Lizenz -Dialog aufrufen

1.0.5 Prinzipbeschreibung

PatControl ist eine Bildverarbeitungssoftware die Merkmale in Bildern erkennt und diese als Ergebnis auf eine angeschlossene Hardware ausgeben kann.

Als Datenquelle werden Bildgebende -Geräte unterstützt wie Digitalkameras, Scanner, Bilddateien und Videos. Verzeichnisse mit Bildern werden ebenso als Geräte verstanden, und ermöglichen früher aufgenommene Fotografien seriell erneut abzuspielen.

Ein Bildgebendes -Gerät liefert an **PatControl** einen **Trigger**, die **Dimension** (Höhe,Breite,Bittiefe) und eine **Bildnummer** sowie die **Pixelbitmap**.

Die gesamte Ablaufsteuerung wird über den **Bildtrigger** ausgelöst, jedes neue Bild stößt einen dazu angelegten Prüfplan an, der durch den Benutzer festgelegte Informationen auf die verfügbaren **Pixelsensoren** anwendet.

Pixelsensoren sind Messoperationen die auf das Eingangsbild wirken, und Informationen über die Eigenschaften eines Bildbereiches liefern.

Die **Pixelsensoren** werden mit der Skriptsprache **LUA** aufgerufen, und ermöglichen durch umfangreiche Parametrisierung der Funktionen auf unterschiedlichste Anforderungen in einem Messbild zu reagieren.

Nachdem im **LUA -Skript** die nötigen Informationen beschafft wurden, können diese auf die grafische Darstellung des Messbildes übertragen werden um die entsprechenden Bereiche in der Ansicht für den Betrachter markiert anzuzeigen.

Die Übertragung der grafischen Informationen sowie die damit verbunden Steuer und Regelausgänge werden über Ergebnispakete transportiert genannt **Patterns**, die auch zur Namensgabe der Software beitrugen.

Die für jede Messung benötigten Parameter des Anwenders, stammen aus einem vom Prüfplaner definierten **XML-Parameter -Baum**. Hier können Felder des Prüfplans angelegt werden, z.B. für **Eingriffsgrenzen Warngrenzen Schwellwerte** und **Bemaßungen**. Diese Informationen können während des Betriebes erfolgen Änderungen wirken direkt auf den **LUA-Skript-Prüfplan** zur Prozesszeit ohne die Anlage anhalten zu müssen.

Ein Prüfplan besteht aus folgenden Bestandteilen.

1) **Kamera/Bild/Video –Geräte -Initialisierungsdatei.**
2) **Standard-Setupdatei als Bindeglied für die Kommunikation. *.set**
3) **Benutzerdefinierter Parameter -Baum. *.xml**
4) **Lua –Skript. *.lua**

2.0.0 Hardware Kommunikation

Mit der Weiterleitung von Messergebnissen auf die Peripherie z.B. **Prozessleitstellen** von Siemens (**SPS**) oder FeldbusController (**WAGO750**) ist die Software einsatzfähig für die industrielle Messtechnik. Zusätzlich werden auch **TCP** Protokollverbindungen unterstützt. Es ist auch möglich direkt aus dem **LUA** –Interface eine Serielle **RS232/432** Verbindung mit einem Mikrocontroller herzustellen.

Datenverbindungen werden in der Datei ***.set** eines Prüfplans definiert, indem dort Die Verbindungsinformationen IP-Adresse und Port eingetragen werden. Für die **SPS** Verbindung werden zudem noch der SpsSockel sowie die Datenbausteinnummer definiert.

Prozessrelevante -Funktionsparameter sind also nur über einen Editor veränderlich.
In der Prüfplankonfigurationsdatei ***.set** wird unter anderem festgelegt, ob überhaupt eine Benutzeroberfläche dem Operator gezeigt werden soll. Ausgelegt ist die Software als **VisionServer** der minimiert seine Abläufe im Hintergrund ausführt und von einer übergeordneten **Prozessteuerung** visualisiert wird. Alle Informationen des **VisionServers** sind also auf einer Plattform wie z.B. **WinCC / VB / Net** darstellbar.

Die einfachste Möglichkeit der optischen stichprobenartigen Betrachtung der Messereignisse kann über den integrierten **WebServer** erfolgen. Wird dieser aktiviert, kann mit jedem „pluginbrowser“ Einsicht auf das letzte Messbild genommen werden.

Für eine Echtzeit -Abbildung des Messbildes, wird die **ActiveX** Komponente **PatConnect** in die eigene Anwendung integriert, selbst auf einem **Excel-Sheet** kann das aktuelle Messbild damit in Echtzeit eingesehen werden, während der **VisionServer** minimiert arbeitet.

PatConnect verkapselt das **SDK**-Beispiel **PatConnectTest**, hier wird genau aufgeschlüsselt wie das **Datagramm** des **VisionServers** via **C/C++** funktioniert.

In vielen Fällen reicht es aus den **VisionServer** als getrennte Anwendung mit seinen Darstellungsfähigkeiten ohne weitere Integration des Messbildes ablaufen zu lassen.

Der **VisionServer** wird über einen eingehenden **Kameratrigger** angestoßen.

(Es ist nicht vorgesehen das eine Kamera im „Freerun“ unentwegt Bilder sendet, jedem Ereignis geht also im protokolliertem Messablauf ein Trigger der Kamera voraus!)

Der **VisionServer** benötigt >10**[ms]** um eine Bewertung des Kamerabildes zu erzeugen und dieses an die **Prozesssteuerung** zu leiten.

Im Folgenden wird die Kommunikation mit der Siemens Prozessleitstelle **S7**, dem **WAGO750** FeldbusController. und der direkten **TCP** Verbindung und dessen Visualisierung beschrieben.

2.0.1 SPS-Kommunikation

Für die **SPS** basierende Kommunikation gilt ein Ablaufplan der bei der Schrittketten-Erstellung der **S7 Programmierung** zur Anwendung kommt. Weiter unten werden Die genauen Abläufe des Protokolls dargestellt.

Der **VisionServer** nimmt nicht teil am **Feldbus**, sondern schreibt die beschafften Informationen mit Geschwindigkeitsvorteil direkt in einen Datenbaustein (**DB**) der **S7** pro ausgelösten Messzyklus. Die Kommunikation erfolgt ausschließlich über den **TCP-Port**. Hier gilt das eine **SPS** mit **TCP**-Frontsteckverbindung schneller ist als gleiche mit einem Ansteckmodul.

Die **SPS** benötigt folgende Strukturdefinition die einen **DB** überlagert, daher muss der **DB** in der Projektplanung mindestens **286 Bytes** lang sein:

```
5   x  INT    =  2Bytes  =  10 Bytes
1   x  REAL   =  4Bytes  =  14 Bytes
3   x  DWORD  =  4Bytes  =  26 Bytes
128 x  INT    =  2Bytes  = 282 Bytes

Benötigte DB Size        = 282 Bytes
                           =========
```

Datenstruktur für einen beliebigen extra angelegten DB(Datenbaustein) :

```
STRUCT
Measure       : INT;                 //Messen Start
Command       : INT;                 //Befehlsnummer kann im Prüfplan verwendet werden
Acknowledge   : INT;                 //Messen-Ok ist immer dann 1 wenn PatControl fertig
Width         : INT;                 //Info Messbereichsweite (Zusatzinfo über die Bildbreite)
Height        : INT;                 //Info Messbereichshöhe  (Zusatzinfo über die Bildhöhe)
Calib         : REAL;                //float Kalibrierungsfaktor multiplikator nach Mikrometer
Trigger       : DWORD;               //der Kamera Trigger steigt mit jedem Bildtrigger
Size          : DWORD;               //Anzahl der befüllten Datenfelder 0-127
ProcessClock  : DWORD;               //Anzahl der Millisekunden seit SystemStart
Data          : ARRAY[0..128] OF INT;//enthält dem Prüfplan entsprechende Daten.
END_STRUCT;
```

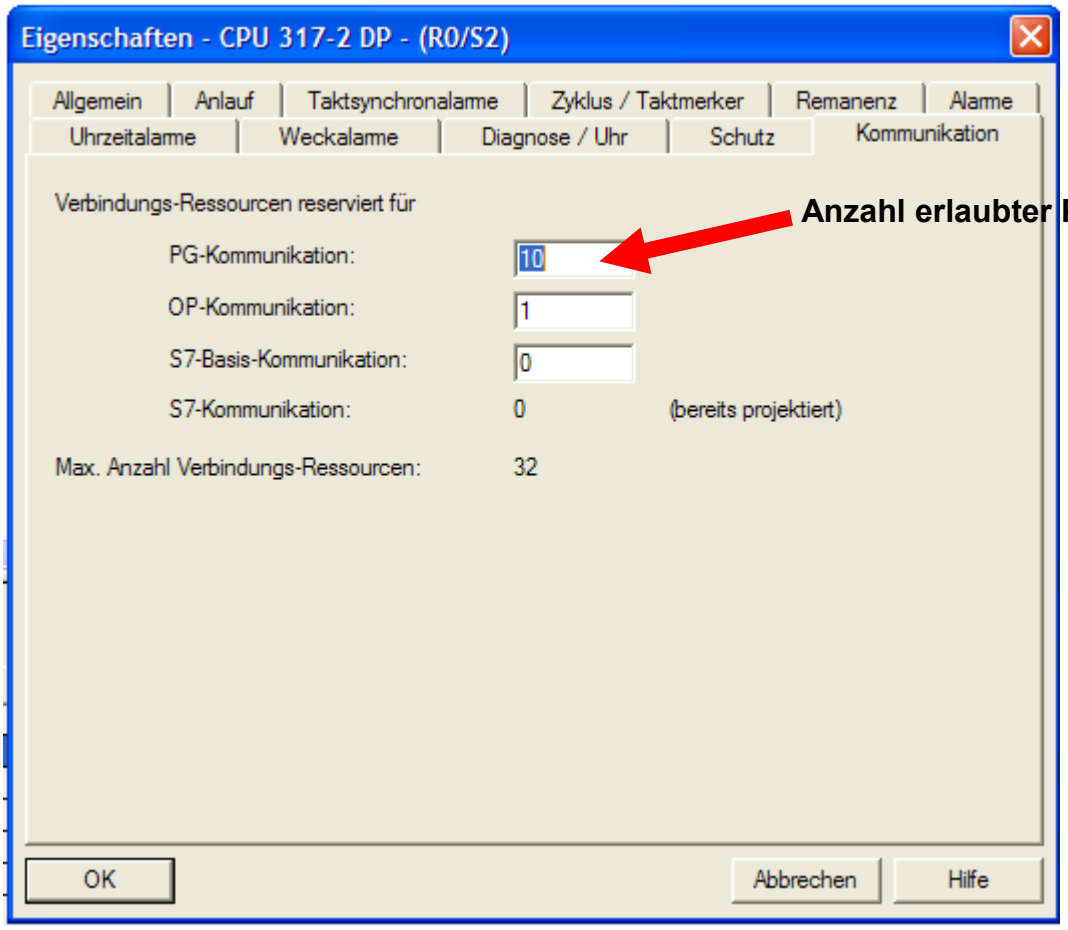
Diese Datenstruktur muss **1:1** kopiert und in einem Funktionsbaustein untergebracht werden. Der Speicherbereich des **DB**(n) kann mit der Struktur überlagert werden um die Datenfelder im **Simatic-Manager** als Variablen in Echtzeit einsehen zu können. Es ist erforderlich mindestens +1 **PG-Gerät** in der Konfiguration zu erlauben. Der **VisionServer** loggt sich als **PG** in die Prozesssteuerung ein. Nach einem **Reset** der **SPS** versucht der **VisionServer** bei einem Kameratrigger sich erneut einzuloggen ohne dass dieser neu gestartet werden muss.

Im Folgenden wird dargestellt wie eine Projektierung der **SPS** aussehen kann.
Es werden in diesem Fall 3 **DB**'s von 3 parallel laufenden **VisionServern** vereinbart. **DB201 DB202** und **DB203** Außerdem wird in **Bild 1** gezeigt wo die Anzahl der **PG's** voreingestellt wird und wie die **Datenstruktur** im **Funktionsbaustein** aussieht.

2.0.2 SPS PG-Anzahl

In den SPS CPU-Einstellungen muss die Anzahl der PG –Geräte > 2 sein damit fremde Anrufer akzeptiert werden.

Eigenschaft Simatic –Manager CPU Konfiguration:



Anzahl erlaubter PG -Teilnehmer

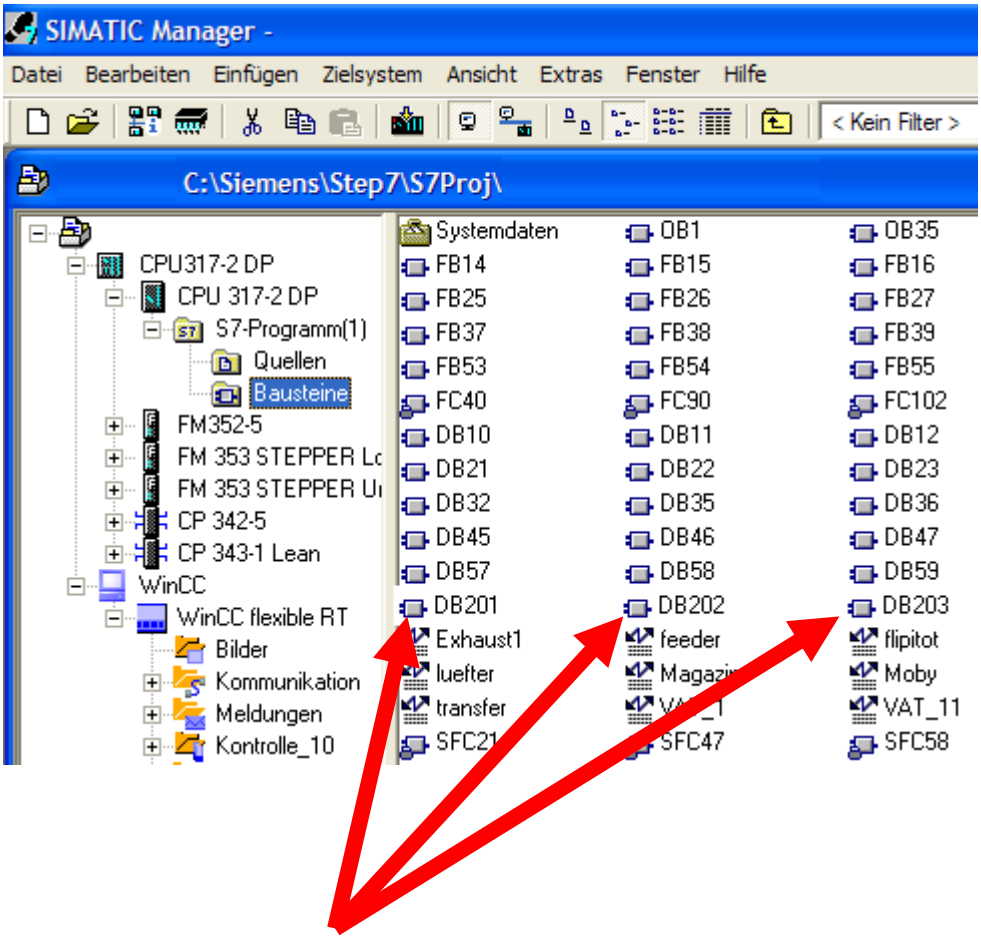
Es wird die empfohlen die **SPS** direkt mit einem **Industrie-Splitter** zu verbinden damit nicht nur der **VisionServer** sondern auch Wartungs-Laptops angeschlossen werden können.

Immer weitere Zugriffe wie Datenbank Logger und grafische Visualisierungen reduzieren die Latenzen, dennoch hat sich die Kommunikation über direkte **TCP -> DB** Beschreibung als effektiv erwiesen.

2.0.3 SPS Datenbausteinvergabe

Für jeden **VisionServer** (dieser kann mehr als einmal ablaufen) wird ein eigener Datenbaustein definiert der mindestens so lang in Bytes ist, wie die Datagramm –Struktur es vorgibt. Diese **DB**'s können dann aus einem Funktionsbaustein gelesen/beschrieben werden.

Eigenschaft Simatic –Manager Projektierung:

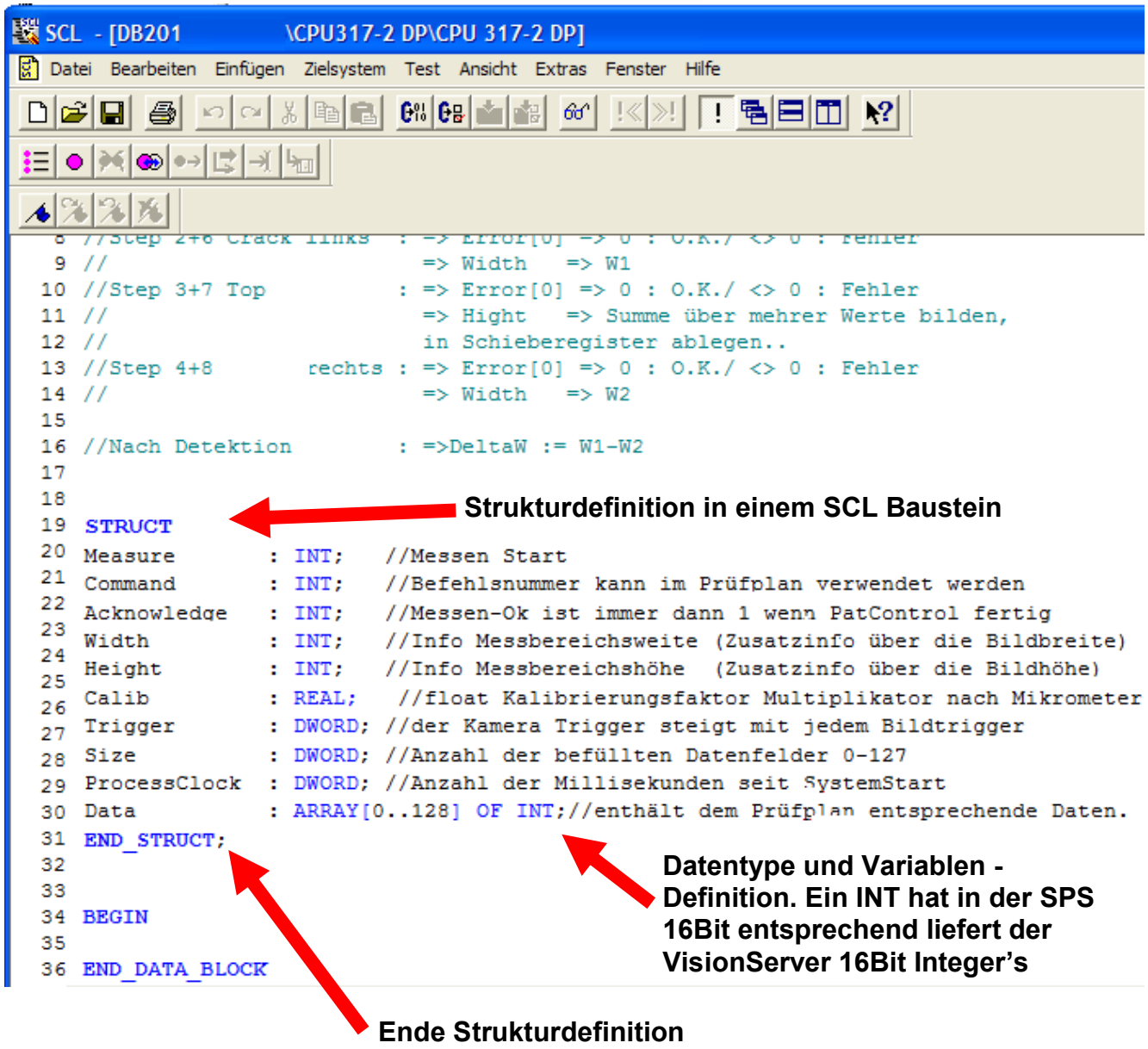


Angelegte Datenbausteine für die VisionServer -Kommunikation

2.0.4 SPS SCL-Datagramm Vereinbarung

In einem Funktionsbaustein kann die oben beschriebene Datagramm –Struktur zur Kommunikation mit den **VisionServern** angelegt werden, jeder Prüfplan enthält in der Datei ***.set** die Nummer des **DB**'s in den dieser schreiben/lesen wird.

Eigenschaft Simatic –Manager Funktionsbaustein:



2.0.5 SPS-Einstellungen am VisionServer

Der **VisionServer** wird durch seine Prüfplandatei die mit ***.set** endet für die **SPS**-Konfiguration passend zu den obigen Beispielen wie folgt konfiguriert.

In der Datei finden sich unter dem Schlüssel **[Frameinfo]** folgende Einträge die angepasst werden müssen.

```
[FrameInfo]
SpsChan = 201            //Der Datenbaustein für R/W
SpsIp   = 192.168.2.100 //Die IP der SPS im Intranet
SpsSlot = 2              //Der Slot für PG-Geräte normal = 2
SpsAccon= 0              //DELTALOGIC AGLink Unterstützung
```

Über den Schalter **SpsAccon=1** kann optional die Unterstützung der SPS-Kommunikation über das **AGLink**-Interface erfolgen.

Diese externe Software benötigt einen vom Hersteller Gelieferten Dongle für das **WIBU** – System.

Alternativ dazu kann die im Setup befindliche **AGLink.dll** durch die **AGLink-Demo.dll** ausgetauscht werden um die Verbindung ohne einen Dongle zu testen.

Nach dem speichern der Prüfplankonfiguration ***.set** und dem Neustart des **VisionServers** wird dieser versuchen die Verbindung herzustellen.

Betrachten sie während der Verbindung/Messvorgänge die
Debug-Ausgaben mit dem Tool **DebugView.exe** aus dem Installations-Verzeichnis.

2.0.6 SPS-Protokollablauf

Ist die Verbindung hergestellt, und eine Kamera angeschlossen die im Triggerfall bereits jetzt ein Bild auf dem **VisionServer** abbildet, kann eine Kommunikationsschrittkette in einem Funktionsbaustein wie folgt angelegt werden:

```
1)  Sps setzt Measure  = 1              //Prüfplannummer
2)  Sps setzt Command  = 0              //freier Hilfsparameter für Prüfplanung
3)  Sps Triggert Kamera                 //VisionServer bekommt jetzt ein Bild
4)  VisionServer setzt Measure = 0      //Bestätigung für Trigger erhalten
5)  Sps setzt Acknowledge = 0           //VisionServer darf DB beschreiben
6)  VisionServer setzt Acknowledge = 1 //VisionServer hat DB fertig beschrieben
```

Nach Schritt **(6)** hat der **VisionServer** alle Daten bereitgestellt, nicht nur daran zu erkennen das **Acknowledge** auf **1** steht sondern auch das das Datenfeld **ProcessClock** immer einen Anderen Wert enthält als zuvor. Das Datenfeld **Size** gibt Auskunft wie viele der möglichen **127 Werte** gesetzt wurden. Der Eintrag **Data**[0] ist ein Generalmarker für Messfehler, ist dieser Wert > 0 wurde ein Messfehler festgestellt oder das Teil ist nicht bewertbar bzw. Fehlerhaft. Ein Wert von **Data**[0] = **0** besagt das die Messung erfolgreich war und alle Dateninhalte den Prüfbedingungen entsprechen.

Je nach Umfang des Prüfplans ist mit folgenden Latenzen zu rechnen:

```
(3) – (4)   1 -   >5[ms]
(4) – (6)  10 - >100[ms]
```

Wie die Daten vom **VisionServer** aufgestellt werden wird im Kapitel **LUA-Prüfplanung** beschrieben.

2.0.6 WAGO FeldbusController 750 (MODBUS)

Für die Kommunikation mit dem **FeldbusController 750** wird das Protokoll durch die **Modbus** -Adressierung verkapselt. Die Kommunikation funktioniert in dieser Fassung ausgehend. Es können **IO-Ports** der Breite **8** und **16** beschrieben werden, das Ausgeben analoger Messwerte im **16 Bit** Modus ist möglich. Die maximale Busbreite beträgt 128 **WORDS(16Bit)**

Das entspricht z.B. **16 x 8Bit IO** Module oder **8 x 16Bit** Analogmodule.

Die Module die der **VisionServer** verwendet sollen zuerst auf der Hutschiene gesteckt werden. In der Folge : Digital -outputs ’s vor Analog -outputs.

Folgende Module werden unterstützt:

```
1) 750-536   Digital Ausgang x8
2) 750-563   Analog  Ausgang 2x16
```

Andere Module wurden nicht dauerhaft getestet.

1.1.1 MODBUS-Einstellungen am VisionServer

Der **VisionServer** wird durch seine Prüfplandatei die mit ***.set** endet für die Modbus-Konfiguration wie folgt konfiguriert.

In der Datei finden sich unter dem Schlüssel **[Frameinfo]** folgende Einträge die angepasst werden müssen.

```
[FrameInfo]
Modport  = 502            //Standard Modbusport
SpsIp    = 192.168.2.100  //Die IP des Feldbuscontrollers
```

Nach dem speichern der Prüfplankonfiguration ***.set** und dem Neustart des **VisionServers** wird dieser versuchen die Verbindung herzustellen.

Betrachten sie während der Verbindung/Messvorgänge die
Debug-Ausgaben mit dem Tool **DebugView.exe** aus dem Installations-Verzeichnis.

Wie die Daten vom **VisionServer** aufgestellt werden wird im Kapitel **LUA-Prüfplanung** beschrieben.

2.0.7 TCP-Kommunikation

Für die Kommunikation mit einem oder mehreren Teilnehmern über **TCP** unterliegt der Datenaustausch einer protokollierten Verbindung, in der ein Datagramm (Datenstruktur) beidseitig ausgetauscht wird. Die Struktur hat dieselbe Form wie schon im Thema **SPS** dargestellt, bildet sich aber für den C/C++ Programmierer wie folgt ab:

```
typedef struct ProcessStatus
{
 short   Measure;      //StartMeasure
 short   Command;      //Command
 short   Acknowledge;  //MeasureReady
 short   Width;        //Detect Width
 short   Height;       //Detect Width
 float   Calib;        //calibfactor multipiler
 DWORD   Trigger;      //Systemtime after restart
 DWORD   Size;         //amount of transfered elements
 DWORD   ProcessClock; //ProcessClock
 short   Data[DATALEN];//Different datas
}PRCSTAT;
```

Die Kommunikation über diese Datenstruktur mit dem **VisionServer** wurde bereits auf Kleincomputern ARM unter **Linux** implementiert sowie über einen in Windows erstellten Klienten. Beide Beispiele befinden sich im **SDK** der Software und werden dort im Detail genau dargestellt. Eine direkte Kommunikation erfordert die Implementierung der dortigen Beispiele.

Über das Datagramm werden also nicht nur die Messdaten des **VisionServers** übertragen sondern auch ein Signal das anzeigt das alle folgenden Daten mit der Länge **Size** Bestandteil eines **JPEG** komprimierten Bildes sind.

Das Signal das zum Empfang eines bevorstehend anliegenden Bild -Datenblockes hinweist tritt auf wenn **Command** = **-1** ist. Ab dann gilt das **Size** Bytes gelesen werden müssen bevor der nächste Datenblock wieder dem Datagramm entspricht.

Es ist möglich mit einer nur vom System begrenzten Anzahl von Anrufern auf diese Daten zuzugreifen da sie an alle verbunden Teilnehmer ausgesendet werden. Verbindlich getestet in der Produktion wurden drei Klienten.

Das Datenfeld **Calib** enthält einen durch den Prüfplaner hergestellten Kalibrier-Multiplikator mit dem es möglich ist von Pixel auf Mikrometer umzurechnen alle Daten werden also als Pixel Aus dem Verhältnis **Width : Height** geliefert. Um einen Messwert aus **Data**[n] in Mikrometer umzurechnen gilt:

X = XMesswert * **Calib**
Y = YMesswert * (1 + (**1-(Height/Width)**)) * **Calib**

Dem Server kann mit dem Text "**SENDIMG=1**" mitgeteilt werden, das Bilder gesendet werden sollen, außerdem kann mit "**TESTPLAN=3**" festgelegt werden das z.B. Prüfplan 3 für den folgenden Trigger aktiviert werden soll, diese Nachrichten können zu jeder Zeit übermittelt werden. Rotationsprüfpläne werden unten noch beschrieben.

2.0.8 TCP-PatConnect

Für **WinCC** / **Basic** / **Net** oder **Excel VBS** -Programmierer gibt es eine Verkapselung der Beispiele aus dem **SDK** zu einer **ActiveX -Com** basierenden Lösung, mit der eine **TCP** Verbindung ohne detaillierte Programmierung in **C/C++** hergestellt werden kann um das Echtzeitmessbild in eigenen Anwendungen verwenden zu können. In **Excel** kann Über das **Menü- Einfügen/Object** eine Liste verfügbarer im System registrierter **ActiveX** Objekte geöffnet werden In der sich das Objekt **PatConnect** befindet. Hier kann die Methode **Connect** verwendet werden, um eine Verbindung zum laufenden Messbild herzustellen.

1.2.2 TCP-WebServer

Alternativ kann auf das Messbild und auch auf die Konfiguration des Prüfplans über einen eignen **WebServer** zugegriffen werden. Somit wird ermöglicht Stichprobenartig das letzte Messbild und die dazugehörigen Prüfplan-Felder aus dem Benutzer definierten Parameter Baum einzusehen und zu verändern. Jeder Prüfplan verfügt über das Verzeichnis **Web** indem eine Freiland editierbare Version einer **HTML** Seite namens **index.htm** vorliegt in der der Inhalt beliebig erweitert werden kann. Die Datei **Setup.htm** wird bei jeder Eingabe im Benutzerdefinierten Parameterbaum neu erzeugt, ein Editieren hat also keine Auswirkung da diese Datei immer überschreiben wird. Wenn ungültige Seiten auf dem **Webserver** aufgerufen werden wird die Fehlerseite **error.htm** angezeigt.

1.2.3 TCP-Einstellungen am VisionServer

Der **VisionServer** wird durch seine Prüfplandatei die mit ***.set** endet für die **TCP**-Kommunikation wie folgt konfiguriert.

In der Datei finden sich unter dem Schlüssel **[Frameinfo]** folgende Einträge die angepasst werden müssen.

```
[FrameInfo]
Tcpport  = 888   //Port auf dem der VisionServer wartet
Webport  = 8080  //alternativ mit Webserver oder 0 kein Webserver
```

Nach dem speichern der Prüfplankonfiguration ***.set** und dem Neustart des **VisionServers** wird dieser auf eingehende Verbindungen von Anrufern auf dem jeweiligen Port warten.

Betrachten sie während der Verbindung/Messvorgänge die **Debug-Ausgaben** mit dem Tool **DebugView.exe** aus dem Installations-Verzeichnis.

Wie die Daten vom **VisionServer** aufgestellt werden wird im Kapitel **LUA-Prüfplanung** beschrieben.

2.1.0 Kameras

PatControl unterstützt zu fast jedem bekannten Verfahren eine oder mehrere Kameras, dennoch kommt es vor, dass einige Hersteller eigene Verfahren herleiten oder bestehende Verfahren sich ändern. Aus diesem Grunde werden alle Bildgebenden Geräte auf 5 Eigenschaften reduziert die diese alle gemeinsam haben. Dadurch ist es möglich, **PatControl** von den unterschiedlichen Verfahren zu isolieren, und für jede Gruppe eine eigene Handhabe in einer ***.DLL** zu implementieren. Auch nachträglich oder auf **Kundenwunsch** können neue Systeme einfach zu den bestehenden hinzu Integriert werden.

Gemeinsamkeiten aller Geräte:

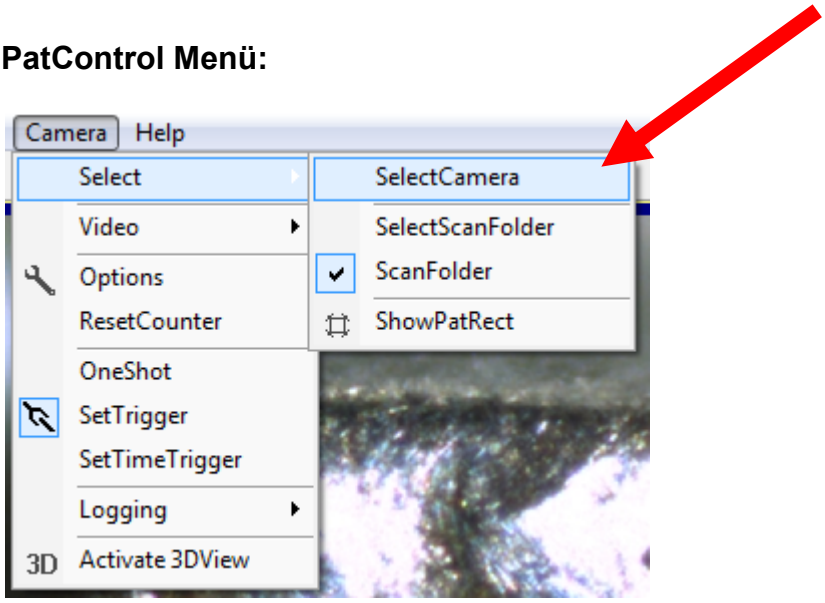
1) **Optionen Dialog**
2) **Trigger an/aus**
3) **Bilddimension**
4) **Bildinformation**
5) **Reset**

Diese 5 Grund Eigenschaften vermutet **PatControl** hinter jeder Bildquelle dazu gehören auch **Bilder**, **Videos** und **Verzeichnisse** mit Bildern. Im Folgenden wird jede Gruppe der bekannten Geräte erklärt. Es gibt noch eine weitere wichtige Gemeinsamkeit, alle Hersteller liefern für Ihre Geräte nicht nur einen Treiber sondern auch ein **SDK** -(SourceDevloperKit) um die Geräte öffnen und Benutzen zu können.

Diese **SDK's** werden von **FlexxVision** in ***.DLL's** verkapselt. Außerdem haben alle Hersteller eine sog. **API-Anwendung** die eine Konfigurationsdatei, in der alle Einstellungen verzeichnet sind speichern kann. Diese enden mit einer vom Hersteller vergeben Dateiendung anhand **PatControl** entscheidet welche **DLL** zu dieser Endung passt.

Zum öffnen einer Kamera verwenden sie das **Menü/Camera/Select/SelectCamera**

PatControl Menü:



Beachten sie die Möglichkeit im Dateidialog die Erweiterungen unten rechts zu öffnen.

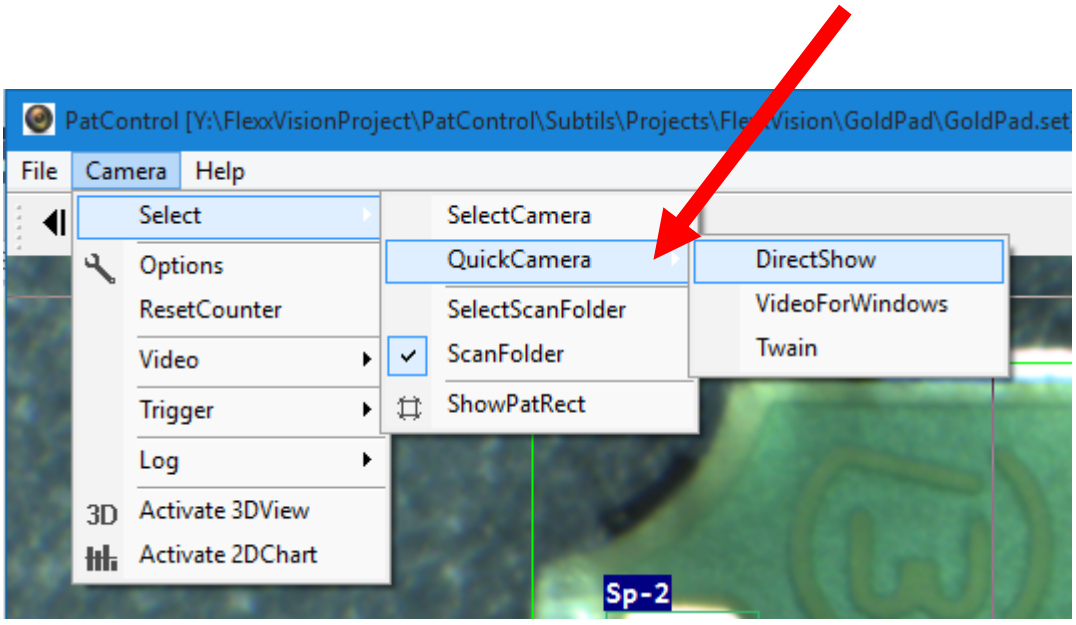
2.1.1 Unterstützte Kamera-Systeme

```
1)  CameraLink    (Dalsa,SiliconSoftware,ViewWorks)
2)  FireWire      (1394 Allied Sony u.a)
3)  CMU           (1394 Digital Camera Driver)
4)  GigE          (Network Protokoll Overlay)
5)  USB           (VideoForWindows [VWF])
6)  Twain         (Scanner aller Hersteller)
7)  FolderScan    (Bilder werden via Folderscan als Kamera verstanden)
8)  EdiMax        (IP-WebCam)
9)  DeskCam       (DesktopVideoCapture)
10) AVI,MPG,JPG,BMP,TIF,GIF (Dateien die als Kamera verstanden werden)
11) LIP           (Leuze LPS36/EN 3D Geometriesensor via UDP)
12) TUC           (Tucsen USB-Kamera Unterstützung HS/TS131/130HC)
13) VisualFilter  (ImageProcessFilterNetwork)
14) DShow         (DirectShow 9)
```

Kameras werden über Ihre **Dateiendung** ausgewählt, im Verzeichnis **Config** der Installation befinden sich zu den beschrieben Typen die entsprechenden Beispieldateien.

Im Folgen werden die unterschiedlichen Systeme in dieser Folge genauer erklärt und auf jeweilige Einzelheiten eingegangen. Neben dem Titel, wird die Dateiendung aufgeführt über die **PatControl** die Hardware identifiziert und öffnet.

QuickCamera Windows unterstützt 3 Arten von bildgebenden Geräten direkt, es ist möglich ohne besondere Konfiguration eine angeschlossene Kamera für **DirectShow,Twain oder VFW** „schnell“ zu öffnen, über den Menupunkt: Camera/Select/QuickCamera:



2.0.2 CameraLink [*.CCF] [*.MCF]

CamerLink ist ein Industrie-Standard der auf **FrameGrabber** basiert, hiermit ist es möglich erhebliche Datenmengen direkt in den Arbeitsspeicher über das **PCIe** Bussystem zu schreiben. Nur so können derartige Datenmengen über **LVDS**(LoVoltageDifferentSignal) sicher übermittelt werden. Hier werden zwei Anbieter unterstützt zum einen **SiliconSoftware** die ihre Konfigurationsdateien mit der Endung ***.mcf** speichern und **Teledyne –DALSA** die als Dateiendung ***.ccf** verwenden. Eine derartige Kamera wird also mit deren Firmware genau spezifiziert und Parametrisiert. **PatControl** öffnet diese fest eingestellten Systeme und ermöglicht es nicht auf deren Parameter zu zugreifen. Dies ist für die **Betriebssicherheit** ein wichtiger Faktor die Hardware nicht während des Messbetriebes verstellen zu können.

2.0.3 Allied Vision FireWire(1394) [*.XML]

Liefert neben dem **SDK** -**FirePackage** auch ein Konfigurations- Tool mit dem einige Grundfunktionen auf dem Flashspeicher der Kamera abgelegt werden. Diese Informationen bleiben auch über die Stromlosigkeit erhalten, während einige andere Parameter wie **Shutter** und **Gain** über einen von **PatControl** bereitgestellten Dialog verändert werden können. **FireWire**-Kameras von **AlliedVision** bilden sich als Netzwerk von Kameras ab, so können z.B. auch mehrere Kameras in einem System zum Einsatz kommen. Nach der Konfiguration mit der **FirmWare SmartView.exe** aus der **FirePackage** werden die Einstellungen gespeichert, hierfür wird von diesem Programm eine Datei mit der Endung ***.xml** erzeugt in der vor allem die eindeutige **ID** der Kamera zu finden ist: **GUID="A57023A0F8CA7"** Sollte die Datei leer sein oder die **GUID** nicht existieren, wird die zuerst gefundene Kamera aus dem **1394 Netz** verwendet.

2.1.4 CMU FireWire(1394) [*.IDC]

CMU 1394 Digital Camera Driver ist ein Universaltreiber der alle **1394 Kameras** ohne deren spezielle **FirmWare** ansprechen kann. So können auch Digitalkameras aus dem Consumer - Bereich einfach mit **PatControl** z.B. am Mikroskop verwendet werden, auch Industrie-Kameras von **AlliedVision** werden so alternativ unterstützt. Dabei ist kein Zugriff auf die "Future‘s" der jeweiligen Hardware möglich. Diese müssen vorher mit der jeweiligen Firmware als UserSet im Flashspeicher der Geräte hinterlegt werden. In der dazugehörigen Datei ***.idc** wird lediglich ein handerstellter Eintrag hergestellt der Auskunft über das zu verwendende Bildformat ermöglicht.

Inhalt der ***.idc** Datei:

```
Board  = 0  //Welche der gefunden Kameras
Format = 7  //Freies Bildformat 7
Mode   = 0  //Na
Bpp    = 24 //Bittiefe
```

2.1.5 GIGE (Network Protokoll Overlay) [*.GIG]

GigE -Kameras werden bisweilen nur von Herstellern unterstützt, die das **SDK** mit der Parametersoftware **SampleViewer.exe** enthalten**.** Dieses Tool erzeugt eine Parameterdatei mit der Endung ***.gig** und kann nur dann von **PatControl** verwendet werden, wenn alle Einträge gemäß des Formates enthalten sind. Außerdem ist es erforderlich eine auf **IntelChipsatz** basierende Netzwerkkarte im System vorzuhalten die mit einem sog. **Protokolloverlay** hohe Übertragungsraten gewährleistet. Da nun die Bilder als gepufferter Stream eintreffen war es bei Substrat -Scannern nicht möglich auf einer Trigger folge die dazugehörige Anzahl von Bildern mit der Aufnahmeposition zu synchronisieren.

2.1.6 TWAIN (Scanner aller Hersteller) [*.SCA]

TWAIN ist ein 1992 von den Unternehmen Aldus Corporation, Eastman-Kodak, Hewlett-Packard und Logitech festgelegter Standard zum Austausch von Daten zwischen Bildeingabegeräten (Scanner, Digitalkameras etc.) und Programmen für Microsoft Windows und Apple Macintosh. Die Dateiendung ***.sca** signalisiert **PatControl** zu versuchen das Scanner-Systeminterface zu öffnen, und bietet über das Menü Camera/Options erweiterte Einstellungen der jeweiligen Hersteller.

2.2.4 Microsoft DShow (DirectShow) [*.DXS]

Das VFW Interface von Microsoft wurde insbesondere ab Windows 10 durch DirectShow abgelöst. In der Steuerdatei mit der Endung *.dxs kann die zu öffnende Gerätenummer die Auflösung und ein Zeitschlitz angegeben werden. Die DirectShow Wiedergabegeschwindigkeit kann durch Angabe des Zeitwertes PollTime in der *.dxs Datei verlangsamt oder beschleunigt werden. (Voreingestellt 10Millisekunden).

Inhalt der Steuerungsdatei *.dxs

```
[Camera]
Id=1            //Nummer der Kamera
PollTime=10     //Zeitschlitz zur Bildratenanpassung
Width=640       //Bild soll breite
Height=480      //Bild soll Höhe
Bpp=16          //Anzahl der Bits pro Pixel
```

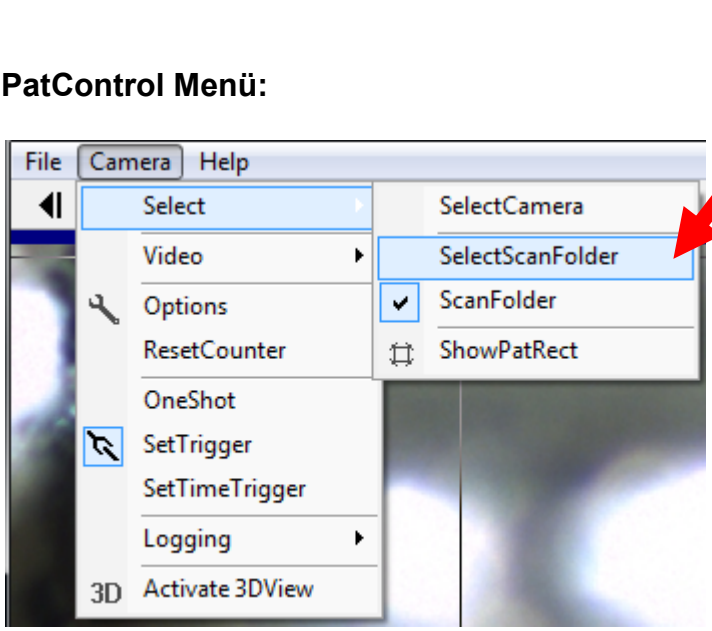
Sollte die Auflösung Width/Height/Bpp nicht bei der Iteration der Kamera gefunden werden können diese aus dem DebugView nach dem öffnen der Kamera eingesehen oder über den Property Dialog neu ausgesucht werden. Es wird sonst die Standard Auflösung der Kamera bevorzugt.

2.1.7 FolderScan (PatControl eigenschaft) [*.*]

Wird versucht eine Kamera ohne Dateiname zu öffnen verwendet **PatControl** den **FolderScan**, dies bedeutet das ein Verzeichnis iteriert wird und alle darin vorkommen Dateien als Bildquelle sequentiell als virtuelle Kamera dem System „vorgespielt“ werden.

Zum öffnen der **FolderCam** verwenden sie das **Menü/Camera/Select/SelectScanFolder**

PatControl Menü:



Diese Eigenschaft kommt insbesondere zur Anwendung, wenn durch das Logbuch(Logging) Raw -unkomprimierte Bitmapbilder in das Logbuchverzeichnis optional geschrieben wurden. Es kann ein ganzer Durchgang von Messungen wieder historisch korrekt abgespielt werden, um eine **Prozessoptimierung** oder **Diagnose** durchführen zu können auch ohne die **Messvorrichtung/Maschine** in der Nähe haben zu müssen.

*Weitere Details unter **Data-Logging**.*

2.1.8 EdiMax (webipcam) [localhost:port]

Erwartet als Dateiname eine IP -Adresse und öffnet dann eine Edimax IP-WEB-Camera. Diese Art von Kameras haben kein standardisiertes Verfahren zur Ansprechung, hier wurden Datenpakete direkt aus dem **JPG** –Stream wieder in Einzelbilder zurück gewonnen. Dieser Kameratype ist nur unter Vorbehalt unterstützt. Auf Anfrage können hier jedoch **Kundenspezifische** Integrationen durchgeführt werden.

2.1.9 DeskCam (WindowsDesktopCapture) [*.win]

Es ist mit diesem Kameratype möglich den **Desktop** als Kamera zu verwenden und ständig das Bild nach **PatControl** zu überbringen. Dies kann für **Wartung** oder **Diagnose** -Zwecke verwendet werden, auch Kamerabilder für die keine Behandlung vorliegt können so erfasst werden. Im Zusammenhang mit dem Webserver kann so der Arbeitsplatz grafisch übertragen, und auf einer **HTML** –Seite abgebildet eine **Fernbetrachtung** ermöglichen.

2.2.0 Microsoft VFW (VideoForWindows) [*.VID]

Auch moderne **USB-Web -Kameras** liefern heute mit **CMOS-Sensoren** akzeptable Bilder die auch für die Bildverarbeitung verwendet werden können. Die Einschränkung sind starre Optiken und ein höheres Rauschverhalten, oder Betriebsparameter die nach der Stromlosigkeit verloren gehen. Dennoch ist **VFW** eines der ältesten Windowsartefakte auf der schlussendlich auch andersnamige Interfaces wie **DirectShow** basierten. Oft werden Kameras über **USB2.0** mit horrenden Auflösungen angeboten, dies ist jedoch eine Filtertechnik die per Software auf der Rechnerseite erreicht wird. Die Sensorgröße ist meist **1/8“** und liefert selten mehr als **640x480** physikalische Pixel.

2.2.1 AVI MPG JPG BMP TIF GIF (Bild und Videodateien) [*.*]

Bilder sowie Videos vom Type **AVI** oder **MPG** können als Kamera über ihre jeweilige Dateiendung geladen werden. **PatControl** findet dann alle Eigenschaften einer Kamera in diesen Daten, auch der **Trigger** im Zusammenhang mit **vor/rück tasten** wirkt auf die Verarbeitung.

Wird ein Bild geladen so wird der Prüfplan nur einmal angestoßen, gerne möchte man dass dieser ständig mit diesem Bild aufgerufen wird. In diesem Fall kann der **Bild –Trigger** abschaltet werden. und es wird damit erreicht dass das Bild ständig an den Prüfplan übermittelt wird. Dies gilt auch für ein **Video -Bild** das gerade auf Pause steht.

Trigger und Vor/Rück/Pause wird weiter unten beschrieben.

2.2.2 Leuze LPS36/EN (3D Geometriesensor) [*.LIP*]

Die 3D –Sensoren von Leuze liefern über die Lasertriangulation z.B. 100 Zeilen/Sekunde mit 340 WORD Werte an Höheninformation. Diese Daten werden vom ImageWrapper (Kameraverkapselung) als 32Bit RGBA Bild zurück geliefert indem die Messwerte in den ersten 16Bit untergebracht werden, und im Alphakanal eine Grauwertinterpolation abgespeichert wird. Diese Technik ist nur mit Kundenabsprache für spezielle Lösungen verfügbar und liefert für die Standardbewertung im Prüfplan nicht ohne Anpassung Daten.

2.2.3 Tucsen (HS/TS131/130HC) [*.TUC]

Ist ein Chinesischer primär Hersteller von Kamerasystemen die in vielen Anwendungen z.B. in der Mikroskopie zum Einsatz kommen. Das Repertoire an Kamerasystemen ist umfassend und es gibt viele verschiedene Treiber und **SDK**'s einige davon unterstützt **PatControl** dazu gehören die Kameras aus der **HS-Serie** sowie die **1.3MP** Versionen **TS131** und die **130HC**. Kundenspezifische Integrationen von weiteren Versionen sind auf Anfrage jederzeit möglich.

2.2.5 VisualFilter (ImageFilterNetwork) [*.fpp]

PatControl unterstützt pro Anwendung nur eine Kamera. **PatControl** ist dahin optimiert mehrmals auf dem Desktop in der Taskbar als **VisionServer** kommunikativ ablaufen zu können, dann auch mit unterschiedlichen Kameras gleichzeitig in jedem Prozess verkapselt.

PatControl verwendet keine direkten Bildfilter vor der Auswertung, dies übernehmen die **PixelSensoren** intern. Dennoch kann es wichtig sein Bilder vor der Zuführung zur Berechnung über Standard **Matrixfilter** zu verändern um Merkmale zu verstärken oder abzuschwächen.

In beiden Fällen wirkt **VisualFilter.exe** Das Programm ist Teil des Setups und kann mehrere **Bildquellen** miteinander **mischen** und auch unterschiedlichste Filterketten herstellen. Damit ist es möglich stereografische oder quadgrafische Überlagerungen verschiedener Kameras herzustellen und dessen Bildausgaben weiter durch Bandfilter mit Matrizen zu multiplizieren. Dies kann so weit gehen, dass ein ganzes Netz von Filterketten hergestellt wird an dessen Ende ein hoch approximiertes Ausgabebild steht.

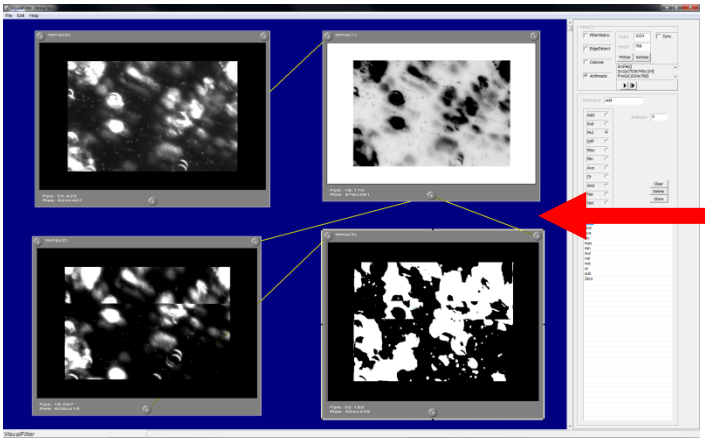
VisualFilter.exe ist also ein **Editor** der das **BildProzessFilterNetzwerk** abbildet das mit der Maus einfach hergestellt werden kann. Nach dem Speichern in eine ***.fpp** Datei wird das Netzwerk in eine interne Datenbank abgelegt die über die in der ***.fpp** Datei stehenden Schlüssel wieder hervorgebracht werden kann, ohne das Programm **VisualFilter.exe** überhaupt noch einmal starten zu müssen.

PatControl kann das finale Ausgangsbild der Filterkette über die Dateiendung ***.fpp** als Kamera „verstehen“ und diese Information wie eine normale Kamera weiterverarbeiten.

Im **SDK** Bereich sind **C/C++** Beispiele die verdeutlichen wie mit **VisualStudio** eine solche Filterkette auch in eigenen Anwendungen integriert werden kann, dort hat man die Möglichkeit auf jede Stufe der Filterkette separat zuzugreifen.

Auch dieses Verfahren ist nicht starr und kann durch **Kundenwünsche** ergänzt werden. Dies betrifft alle Komponenten von **PatControl**.

Bildnetzwerk als Virtuelle Kamera, das Verfahren wird im Anhang beschrieben



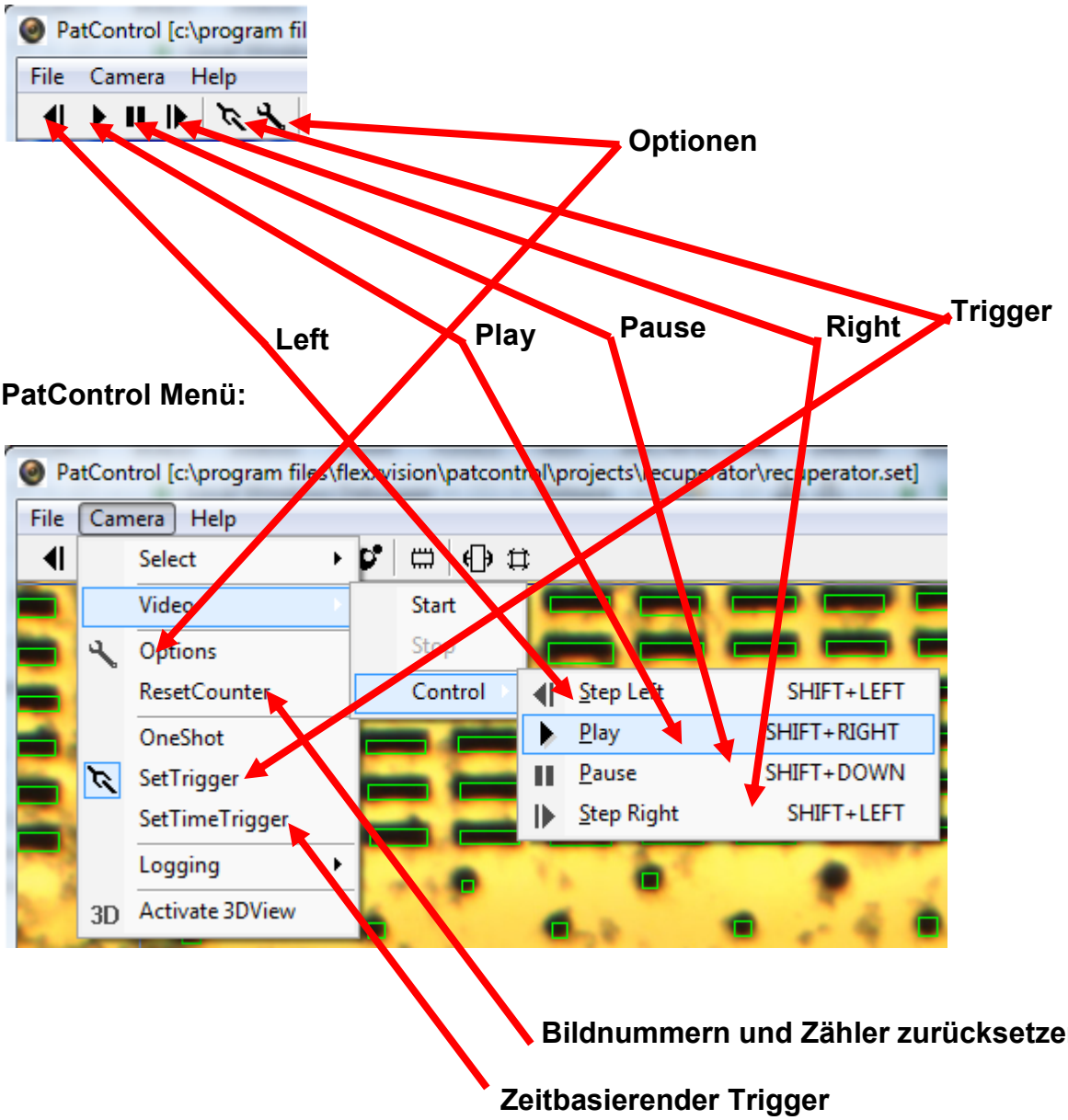
Projektierung mehrerer virtueller Kameras als Bildnetzwerk.

3.0.0 Bild Kontrolle

Unter Bild Kontrolle versteht **PatControl** die klassischen „Schalter“ für Bildlauf **Vor/Rück/Pause/Play**, **Standbild**, **Speichern**, **Laden**, **Anstoßen**(Triggern) .

Da die Software zwischen Video, Kamera und Einzelbild keine Unterschiede macht Bildet sich ein überlagerndes Verhalten bei der Bild Kontrolle ab. Diese Verhalten werden im Einzelnen aufgeführt.

Toolbar Buttons:



PatControl Menü:

3.0.1 Einzelbilder

Einzelbilder stammen aus einer Datei. Nach dem Laden des Bildes wird der Prüfplan einmal ausgeführt. Möchte man dass ein Einzelbild ständig den Prüfplan ausführt um z.B. den Programmablauf zu beobachten und die Ergebnisse der Modifizierungen in Echtzeit zu betrachten muss der **Bildtrigger** ausgeschaltet werden. Bei Einzelbildern sind die Schalter zur Bildsteuerung **Left/Right/Pause/Play** deaktiv. Einzelbilder haben keinen **Optionsdialog**.

3.0.2 Videos

Bildquellen vom Type Video werden in jetziger Version nur vom Format ***.avi** unterstützt. Bei Videos wirken die Schaltflächen **Left/Right/Pause/Play** der **Trigger** hat keine Auswirkung auf die Bildwiederholung, dafür bedarf es einer Einzelbildabspeicherung über die Logging -Funktionalität bezüglich **Raw-Images**. Beschrieben wird der Vorgang im Thema Prozess-Logbuch Aufzeichnung. Das Rückblättern der Bilder im Video erzeugt eine Dekrementierung des Bildtriggerzählers. Videos haben keinen **Optionsdialog**.

3.0.3 Folders

Das durchlaufen eines **Folders/Verzeichnisses** funktioniert am besten wenn dort alle Bilder dieselbe **Dimension** haben. Außerdem ist es vorteilhaft und schneller ein unkomprimiertes ***.bmp** zu laden anstatt einer ***.jpg** Version. In fast allen Beispielen wird aus Speicherplatz gründen in den Projekten der Dateitype ***.jpg** bevorzugt.

Left/Right/Pause/Play wirken wie bei einem Video, mit dem Unterschied das ein nicht pausierter **Folderscan** mit **Trigger** = „**on**“ auf der aktuellen Position den Prüfplan rotatorisch auslöst. Wenn der **Trigger** ausgeschaltet ist funktioniert der automatische Verzeichnis -Vorlauf nicht. Ein **Folderscan** der auf Pause steht wird weder getriggert noch durchlaufen, man kann jedoch mit **Left/Right** durch das Verzeichnis schreiten. Das rückblättern im **Folderscan** erzeugt eine Dekrementierung des Bildtriggerzählers. **Folderscan**’s haben keinen **Optionsdialog**.

3.0.4 Digitale Bildgeber

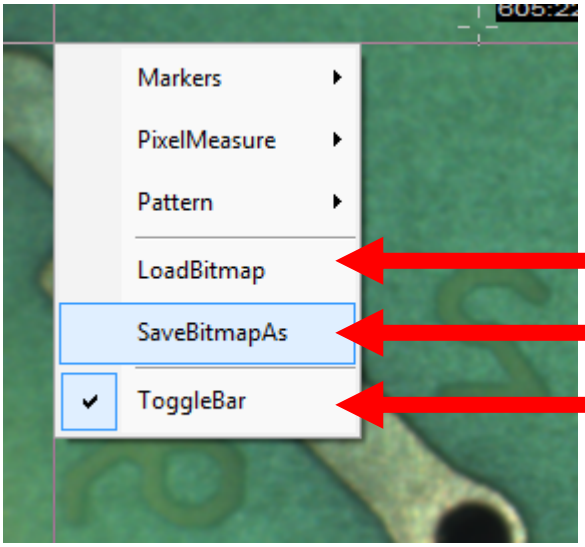
Für den Typ **Kamera** wirkt lediglich der Bildtrigger. Die Schaltflächen **Left/Right/Pause/Play** Sind deaktiviert. Industriekameras werden über ein **TTL/LVDS –Signal** getriggert. Darauf ist **PatControl** ausgerichtet. **Webcams/Scanner** sind Dauerläufer hier wirkt der Trigger zum blocken der andauernden Bilderzeugung. Für alle Vorgänge kann der **Time** gesteuerte **Trigger** verwendet werden, dieser lässt neue Bilder erst dann durch, wenn ein **TimeOut** in Millisekunden verstrichen ist. Diese Option kann insbesondere nützlich sein, wenn eine Physikalisch getriggerte Kamera zusätzlich durch den **Timeout -Trigger** geblockt wird. Dies bremst die Protokoll -Kommunikation mit einer **Prozessleitstelle** soweit ab, das die gesamte **Produktion** dadurch im Einzelschritt ablaufen kann. Je nach Kamera haben diese einen **Optionen -Dialog** dazu gehören alle Kameras von **Allied Vision** sowie die von **Tucsen**, **Twain** und sämtliche über **VFW** eingebunden Geräte

3.1.0 Bild Speichern/Laden

PatControl unterscheidet generell 2 Arten von Bildern, **Prozess-Bilder** und **Raw-Bilder**. **Prozess-Bilder** enthalten bereits erzeugte Grafiken und Schriften. Diese Bilder werden immer im ***.jpg** Format gespeichert. **Raw-Bilder** entsprechen der **Bildquelle** und dürfen nicht verändert werden um die Möglichkeit zu haben gespeicherte Bilder unverändert erneut der Bewertung zu unterziehen. **Raw-Bilder** können pro Bild sehr schnell 10**[MB]** Speicher auf dem Datenträger belegen. Insbesondere **32Bit RGBA** hochauflösende Farbkamerabilder sind **Speicherlastig**. **Prozess-Bilder** werden bei aktiviertem Webserver ständig erzeugt und auch versendet. Oft ist die Dateigröße kleiner 100**[Kb]**.

Zum Speichern des aktuellen Messbildes werden immer zwei Versionen gespeichert, das auslösen der Speicherung eines Messbildes erfolgt mit der rechten Maustaste auf das **PopUp Menü** des sichtbaren **Messbildes**.

Messbild PopUp Menü:



Überladen des aktuellen Bildes

Speichern des aktuellen Bildes

Verstecken der Toolbar

Default wird der Dateiname **Shot** angezeigt der beliebig verändert werden kann, und den Namen der **Raw-Bild** Version bezeichnet. Automatisch dazu wird eine zweite Version im ***.jpg** Format gespeichert mit der Textverlängerung „**Dat**“ in dem Fall : **ShotDat.jpg**
Die zweite Version enthält die Messinformationen und Texte aus dem Messbild.

Die automatische Speicherung von **Prozess-Bildern** und/oder **Raw-Bildern** wird unter Prozesslogbuch-Speicherung beschrieben.

3.1.2 Videospeicherung

Jede Bildquelle kann automatisch für jede Bilderneuerung als unkomprimierter Videostream gespeichert werden. Dabei entstehen beträchtliche Dateigrößen die ab 2**[GB]** aufgesplittert werden in dem jede weitere >2**[GB]** Datei eine numerische Kennung im Dateinamen erhält.

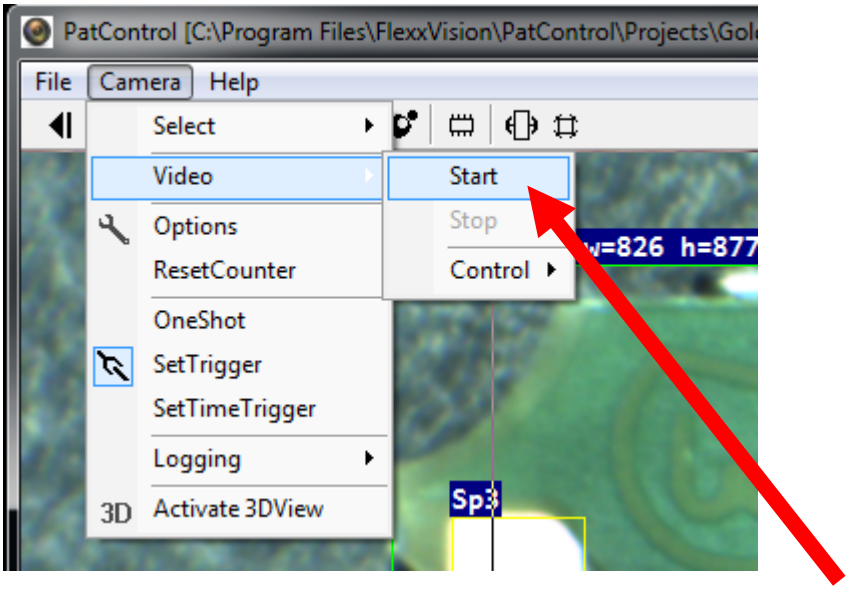
Ausschließlich erfolgt die Speicherung im ***.avi** Format dies kann später nachbearbeitet werden, dient aber im Rahmen der **Messbildverarbeitung** hier als **Ur-Datenquelle** die inhaltlich nicht verändert behandelt wird. Alle Videokompressoren reduzieren zwar die Dateigröße aber auch beträchtlich die Qualität was hier einem Datenverlust gleichkommt.

Der Dateiname wird aus dem entsprechenden Kamerakonfigurationsnamen gewonnen, auch das Datenziel ist dasselbe Verzeichnis in dem auch die **Kamera-Initialisierungsdatei** liegt. Beispiel: **C:\Programme\FlexxVision\PatControl\config\Kamera.xml**

So wird das Video den Dateinamen **Kamera001.avi** im selben Verzeichnis erhalten.

Zum Starten der Aufzeichnung kann jederzeit der **Menüpunkt Camera/Video/Start** ausgelöst werden. Zum Anhalten der Aufzeichnung wird **Camera/Video/Stop** ausgelöst.

PatControl Menü:



Menü Video Start/Stop

Das Abspielen erfolgt mit einem **Mediaplayer** oder über **PatControl** wie oben dargestellt.

Ein bereits ablaufendes Video kann nicht als Video erneut gespeichert werden.

4.0.0 Pattern Editieren

In einem zu untersuchenden Messbild ist nicht grundsätzlich das gesamte Foto von Interesse zumeist gibt es bestimmte **Bereiche** in denen eine genaue Untersuchung der Bildinformation erfolgen soll. Eine Bildbereichsuntersuchung kann erhebliche Prozessor lasten erzeugen, es kann vorkommen das jeder Pixel mit anderen in eine Berechnung einbezogen wird.

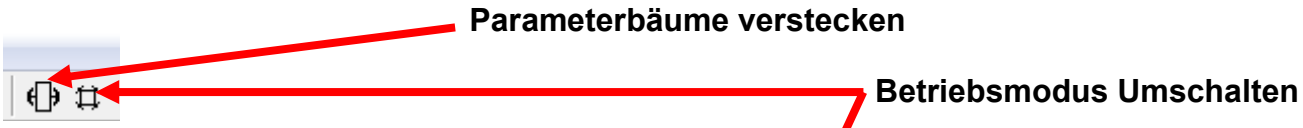
Der Bezug auf Patterns spiegelt sich auch in der Namensgabe der Software wieder. Ein **Pattern** ist ein durch den Prüfplaner erzeugtes Rechteck mit einem eindeutigen **Namen** und einem Set von Standardparametern die in der Prüfplanung zur Parameterübertragung in den Prüfplan dienen. Diese Parameter sind also Bild-Lokal es gibt auch Benutzerdefinierte Bild-Globale Parameter auf die im Thema Benutzerdefinierte Parameterbäume eingegangen wird.

Alle in einem Pattern vorkommenden Variablen erhalten nur vom Prüfplaner im **Lua-Skript** eine Bedeutung. **PatControl** verwendet keinen dieser Parameter für eigene Vorgänge.
Der Zweck dieser Parameter die zu jedem Pattern geführt werden ist, das im **Prüfplan** zwischen den einzelnen Bildbereichen Daten zur Hand sind die nur für diesen Bereich gelten.

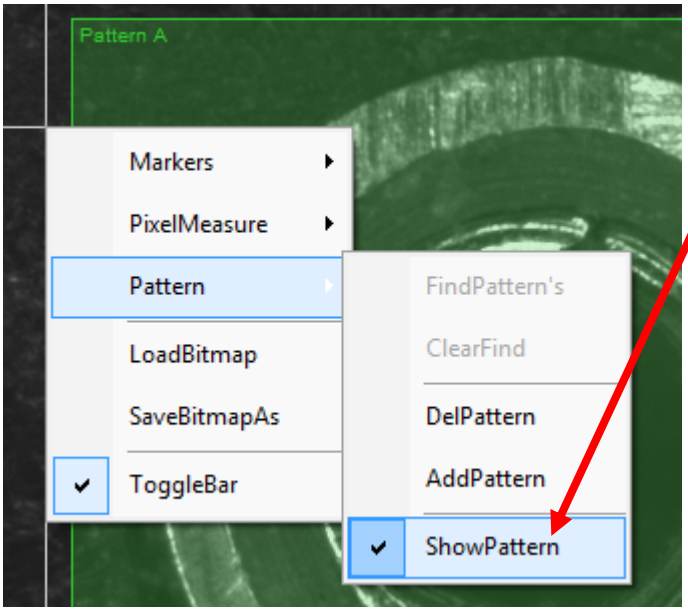
Das Messbild enthält zwei Betriebsarten den **Darstellungsmodus** und den **Editormodus**. Nur im Editormodus können Pattern erzeugt, gelöscht oder in der Größe verändert werden.

Zum umschalten in den Editormodus verwenden sie im Messbild die rechte Maustaste um das **PopUp-Menü** hervorzubringen.

Toolbar Buttons



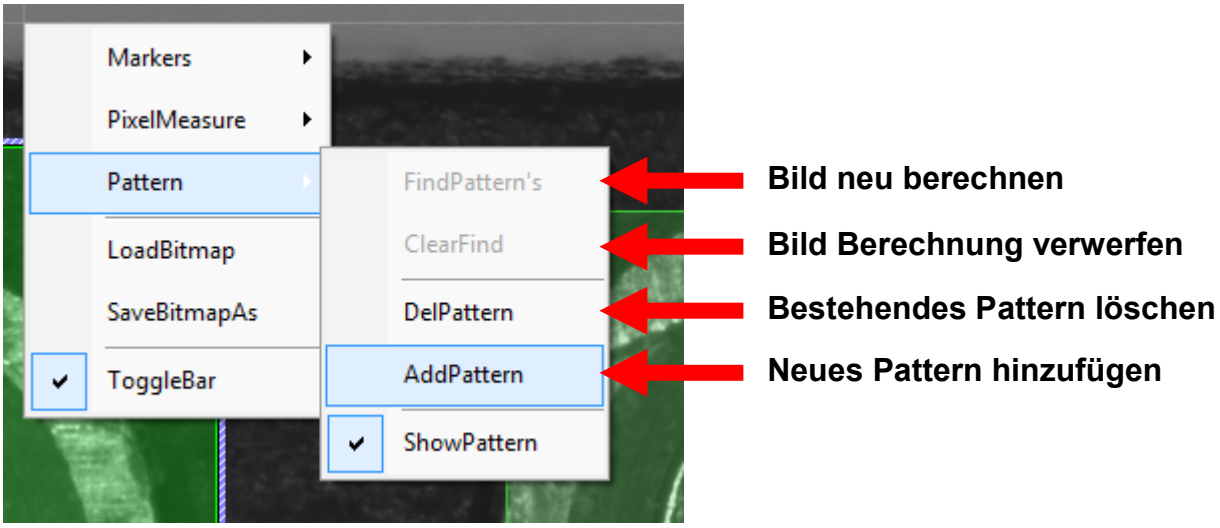
Messbild PopUp Menü:



4.0.1 Pattern Anlegen

Um ein Pattern anzulegen ist es sinnvoll wenn bereits ein aktuelles Kamerabild angezeigt wird, die Pattern Bereiche sind durchsichtig und ermöglichen eine genaue Überlagerung mit dem Bildbereich den man „einfangen“ möchte. Zum Anlegen eines neuen Patterns verwenden sie im Messbild die rechte Maustaste um das **PopUp-Menü** hervorzubringen.

Messbild PopUp Menü:



Messbild Neues Pattern:

Das neue **Pattern** wird als kleines Rechteck dargestellt das nun mit der Maus in Position und Größe dem gewünschten Messbereich angepasst wird. Dazu werden mit der Maus die Standard -**Kontaktpunkte** die das Pattern umranden bewegt, Bzw. insgesamt verschoben, wenn der Mauscursor durch anklicken des Bereiches zu einem 4 Richtungszeiger wird.

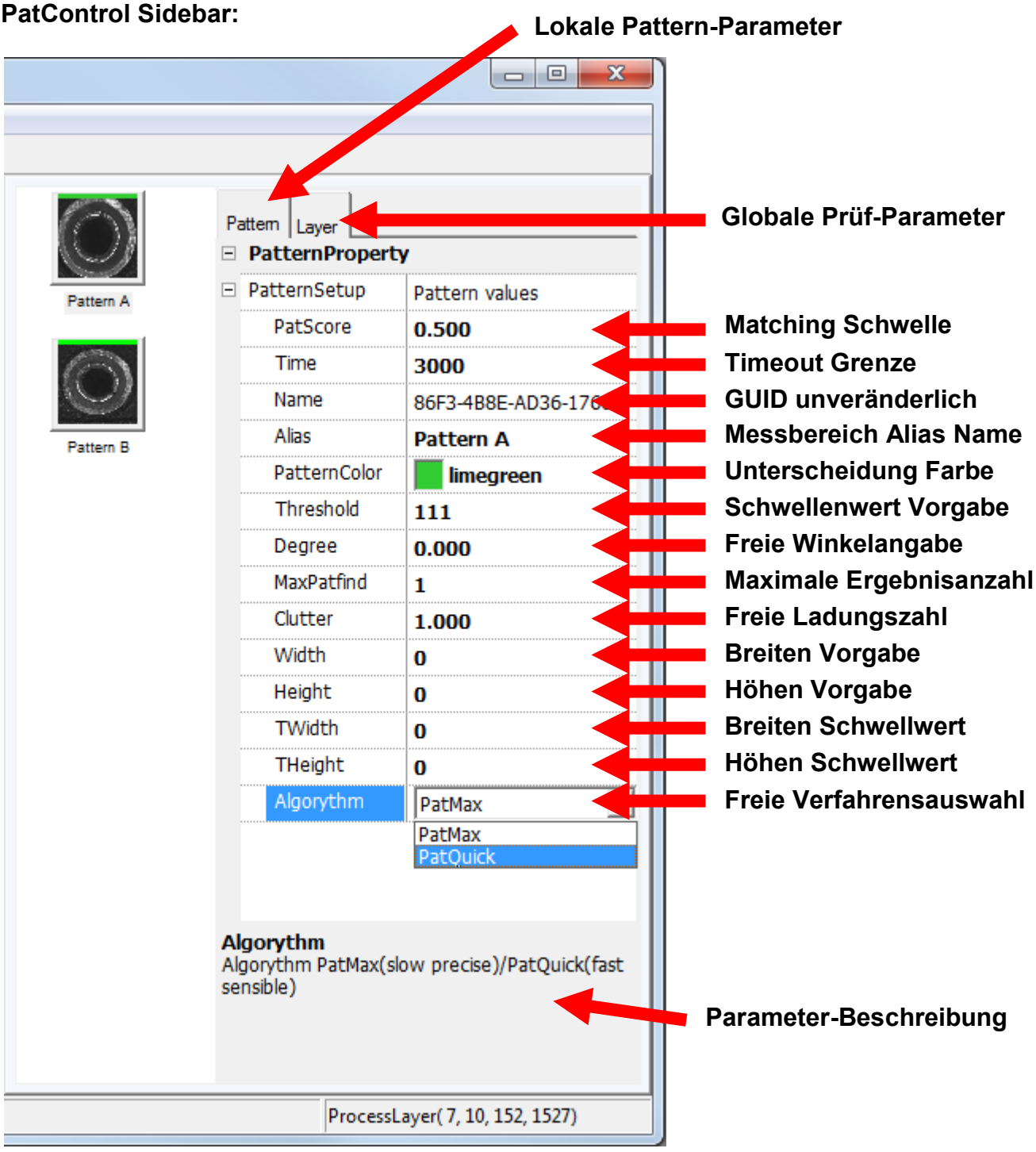
Das **Pattern** hat automatisch einen neuen Namen erhalten, und eine unveränderliche **GUID** Das Namensfeld lautet **Alias** dieser **Aliasname** kann frei verändert werden und wird in der weiteren Prüfplanung als einziger **Bezeichner** zur Unterscheidung der vorhanden Messbereiche dienen.

Zum löschen eines Pattern wird immer das gerade aktive Pattern über das Menü gelöscht

4.0.2 Pattern Parameter Tree

Der feste Satz an Standard-Parametern hat sich im Laufe der Zeit entwickelt und deckt verschiedene Information ab, die ein benutzerdefinierter Messbereich benötigt.

PatControl Sidebar:



Jede Veränderung der Parameter löst im **LUA-Prüfplan** ein Event aus, das alle Inhalte an das LUA-Skript überträgt, die Aufnahme der Daten wird unter LUA-Prüfplanung beschrieben.

4.0.3 Pattern Parameter Datei

Die Parameter jedes **Pattern** finden sich in der Prüfplandatei ***.set** für jedes angelegte Pattern getrennt aufgelistet wieder, diese können dort mit einem Texteditor verändert werden.

Parameter Auszug aus der Datei ***.set**

```
[86F3-4B8E-AD36-1768]
Alias=Pattern A
PatScore=5.000000e-001
Time=3000
Rgb=3329330
Threshold=111
Degree=0.000000e+000
MaxFind=1
Clutter=1.000000e+000
Width=0
Height=0
TWidth=0
THeight=0
Algorythm=0
```

Die vom Prüfplaner vorgegebene Dimension jedes Patterns ist unter dem Schlüssel **[Rect]** in der Datei ***.set** gespeichert und sind in der gleichen Reihenfolge zu den Parametern der Patterns angeordnet.

Parameter Auszug aus der Datei ***.set**

```
[Rect]
Pos(0)=560;1008;136;664,20;460;144;612
```

5.0.0 Benutzerdefinierte Parameter Bäume

Eine **Bildauswertung** benötigt weiterführende vom **Bediener/Operator** eingeb bare Informationen die **Global** für den gesamten Messablauf gültig sind. Hierzu gehören **Bemaßungsangaben**, **Längen**, **Breiten**, **Warngrenzen**, **Eingriffsgrenzen**, obere **OT** und untere Grenzen **UT.** Es werden **Schwellwerte** und **Aktionsschalter** sowie **Auswahlboxen** für unterschiedlichste Verfahren benötig die der Prüfplaner für das **LUA-Interface** implementieren kann. Diese Prüfplanbedingten **globalen** Parameter sind für jede Messaufagbe unterschiedlich.

PatControl ermöglicht zur Prozesslaufzeit Datenfelder hinzuzufügen oder zu verändern. Jeder Prüfplan führt neben seiner ***.set** Datei eine gleichnamige Datei mit der Endung ***.xml** Hier befinden sich die **Steueranweisungen** für den **Parameterbaum-Interpreter**.
Die verschiedenen Parameter können in Gruppen zusammengefasst werden.

Eine Gruppe wird durch das Schlüsselwort **<Root** eingeleitet und durch **/Root>** beendet. Jedem Parameter und jedem Gruppeneinteiler kann eine **Beschreibung** beigefügt werden die im Informationsbereich eines Parameterbaumes angezeigt wird wenn das jeweilige **Item** mit der Maus angewählt wird.

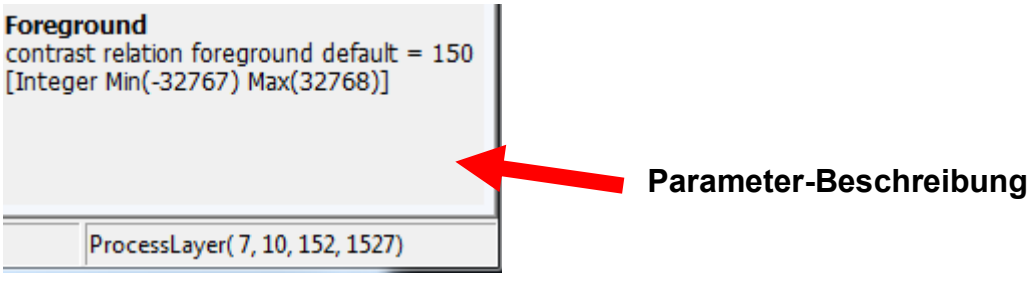
Beispiel eines **Edit** –Eingabefeldes vom Type **Integer** mit einem **Informationstext** der im unteren Bildbereich des Parameterbaumes angezeigt wird.

XML Steueranweisung für ein Edit-Integerfeld:

```
<Edit
Format=int
Name=Foreground
Value=130
Type=edit
InfoText="contrast relation foreground default = 150"
/>
```

Erscheinungsform des integer Feldes aus der XML Steueranweisung :

Darstellung der Feld Information im Fußbereich des Parameter-Baumes:



5.0.1 Mögliche Parameter Felder

Folgende Benutzerdefinierte Felder werden unterstützt:

```
1) Auswahlboxen    (Combobox)
2) Schalter        (Button)
3) Text            (Edit)
4) Zahlen          (Numedit)
5) Gruppierer      (Seperator)
```

5.0.2 Auswahlboxen (Combobox)

```
<Combo
Format=int
Name=Verstärkung
Value=0
Type=combo
Index=Eintrag A,Eintrag B,Eintrag C
InfoText="Feld Information für den Benutzer"
/>
```

Das Feld **Name** enthält den angezeigten Feldnamen. Der Datentype **Format** ist **int**, der **Type** des Control's ist **combo** und der Default Eintrag **Value**=0 lässt den ersten „Eintrag A" als aktives Feld erscheinen. Hinter dem Eintrag **Index** folgt durch Komma getrennt jeder weitere in der Liste aufgeführte Parameter. Das Control liefert an den Prüfplan die **Indexnummer** des gewählten Listen -Eintrags dass vom Benutzer ausgewählt wurde. Der Eintrag **InfoText** enthält die Information die dem Benutzer nach der Feldauswahl gezeigt wird.

5.0.3 Schalter (Button)

```
<Button
Format=int
Name=Verstärkung
Value=0
Type=button
InfoText="Feld Information für den Benutzer"
/>
```

Das Feld **Name** enthält den angezeigten Feldnamen Der Datentype **Format** ist **int**, der **Type** des Control's ist **button** und der Default Eintrag **Value**=0 ist Null. Das Control liefert an den Prüfplan eine **1** für gerade gedrückt, und eine **0** für nicht gedrückt. Der Eintrag **InfoText** enthält die Information die dem Benutzer nach der Feldauswahl gezeigt wird.

5.0.4 Text (Edit)

```
<Edit
Format=text
Name=Verstärkung
Value=Textnachricht
Type= edit
InfoText="Feld Information für den Benutzer"
/>
```

Das Feld **Name** enthält den angezeigten Feldnamen. Der Datentype **Format** ist **text**, der **Type** des Control's ist **edit** und der Default Eintrag **Value**=Textnachricht. Das Control liefert an den Prüfplan einen Text. Der Eintrag **InfoText** enthält die Information die dem Benutzer nach der Feldauswahl gezeigt wird.

5.0.5 Zahlen (NumEdit)

```
<Edit
Format=int
Name=Verstärkung
Value=10
Type= edit
InfoText="Feld Information für den Benutzer"
/>
```

Das Feld **Name** enthält den angezeigten Feldnamen. Der Datentype **Format** ist **int**, der **Type** des Control's ist **edit** und der Default Eintrag **Value**=10. Das Control liefert an den Prüfplan ein integerer Wert. Der Eintrag **InfoText** enthält die Information die dem Benutzer nach der Feldauswahl gezeigt wird.

5.0.6 Gruppierer (Seperator)

```
<Root
Format=text
Name=Verstärkung
Value=0
Type= colapse
InfoText="Feld Information für den Benutzer"
/>
```

Das Feld **Name** enthält den angezeigten Gruppen –Namen. Der Datentype **Format** ist **text**, der **Type** des Controls ist **colapse**. Für eingeklappt oder **expand** für ausgeklappt, der default Eintrag **Value**=0 bleibt unbeachtet. Das Control liefert an den Prüfplan keinen Wert. Der Eintrag **InfoText** enthält die Information die dem Benutzer nach der Feldauswahl gezeigt wird.

```
</Root>
```

Beendet die Gruppierung für alle Controls dazwischen.

6.0.0 Vermessungs und Kalibriervorrichtung

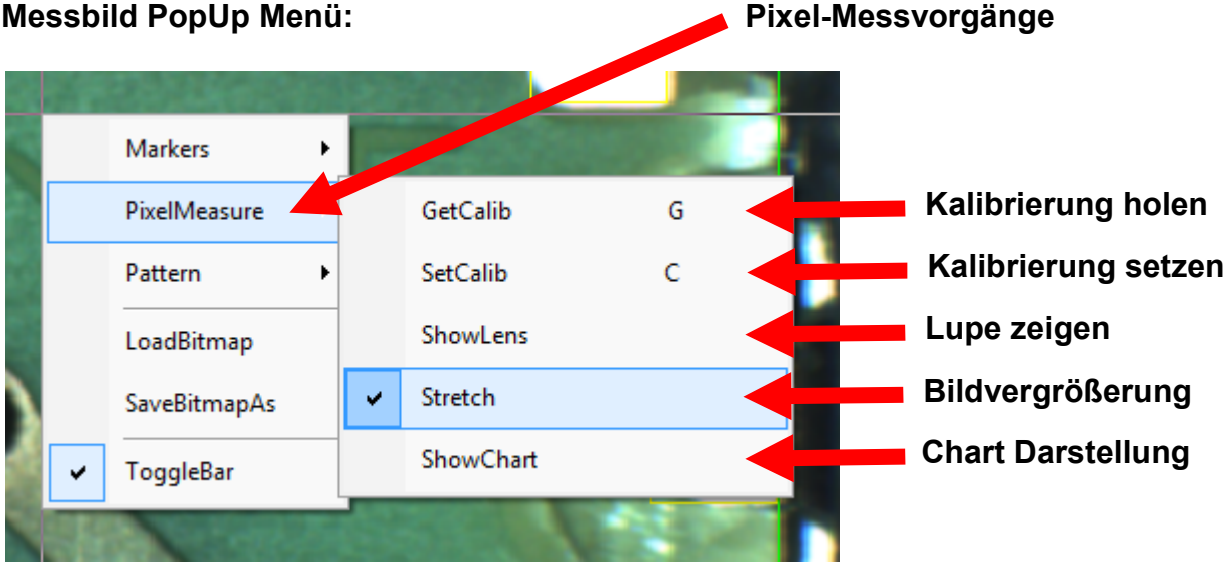
Jeder Pixel eines Bildes entspricht einer Anzahl von Mikrometern, hat ein Kamerasensor eine Pixelauflösung 3**[µm]** und eine optische Vergrößerung von 1:1 entsprechen 1000 Pixel : 1000**[Pixel]** x 3**[µm]** = 3000**[µm]** = ~3**[mm].**

PatControl unterstützt Benutzerdefinierte Kalibrierungen, dazu wird ein Band mit Start und Zielmarker über ein sichtbares Objekt gespannt dessen Länge bekannt ist. Dieser Wert wird in einer Historie -Eingabebox gespeichert und kann später durch auswählen wieder aktiviert werden. Daraus ergibt sich für jede Anzahl von Bildpunkten ein Multiplikationsfaktor der durch die gesamte Prüfplanung gereicht wird, und auch Bestandteil der Hardwarekommunikation ist. Alle Messwerte werden als Ur-Werte in Pixel übertragen plus den vom Prüfplaner festgelegten Kalibrierfaktor.

6.0.1 Bildvergrößerung und Get/Set Calib

Das Messbild kann in zwei Darstellungen angezeigt werden aufgezogen oder Original, ein aufgezogenes Bild wird von der Bildhardware auf die Fenstergröße gezogen und entspricht nicht mehr der tatsächlichen **Bildformatierung**. Für alle Kalibrier und Messvorgänge mit der Maus muss der **Modus Stretch** beendet werden. Kalibrierinformationen und Ziehbänder werden im Modus **Stretch** nicht dargestellt. Zum Anzeigen der Vergrößerungs-Umschaltung und des Kalibrier-Eingabe Dialogs verwenden sie im Messbild die rechte Maustaste um das **PopUp-Menü** hervorzubringen.

Messbild PopUp Menü:

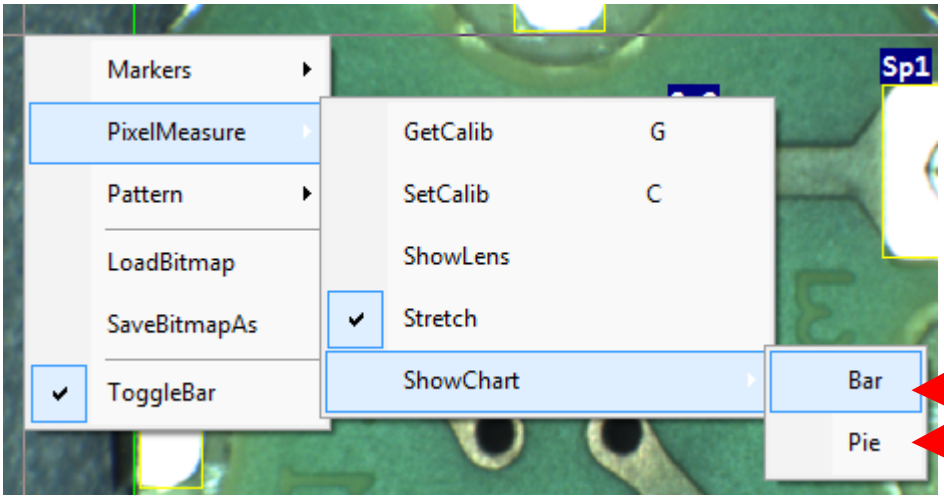


Der Menüpunkt Stretch wird jetzt abgeschaltet.

6.0.2 Histographische Messbild-Darstellung

Das **Messbild** kann alle Informationen auch in einer Balken/Torten –Darstellung abbilden wenn der **Prüfplaner** hierfür Zusatzinformationen mit **SetPatResult** überträgt das unter **LUA**-Prüfplanung näher erklärt wird. Zum Anzeigen der histographischen **-Messbilddarstellung** verwenden sie im Messbild die rechte Maustaste um das **PopUp-Menü** hervorzubringen.

Messbild PopUp Menü:

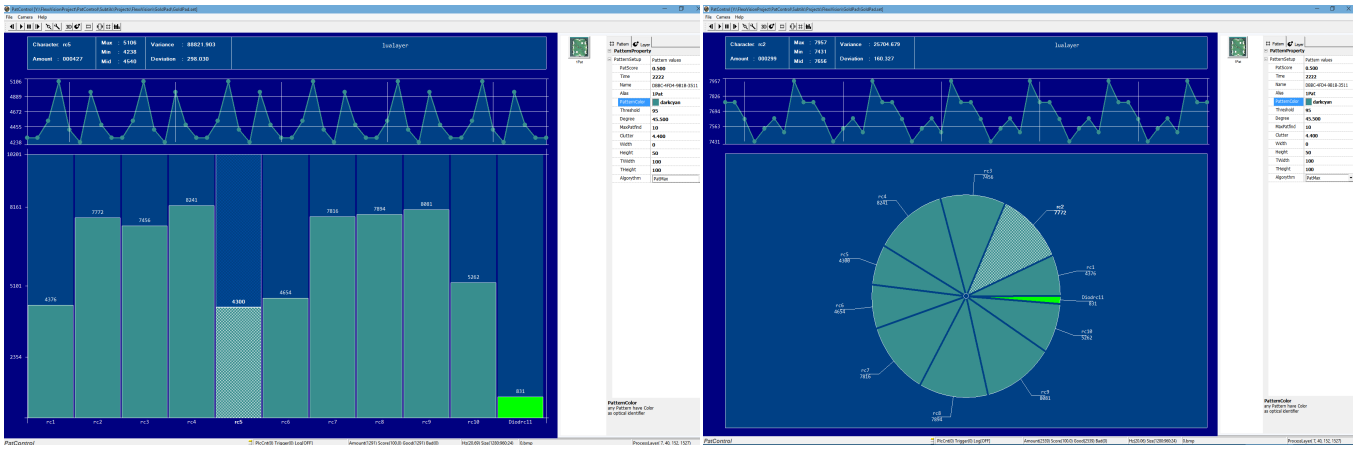


Balken Darstellung

Torten Darstellung

Statistische Merkmal -Informationen

Balken/Torten -Darstellung:



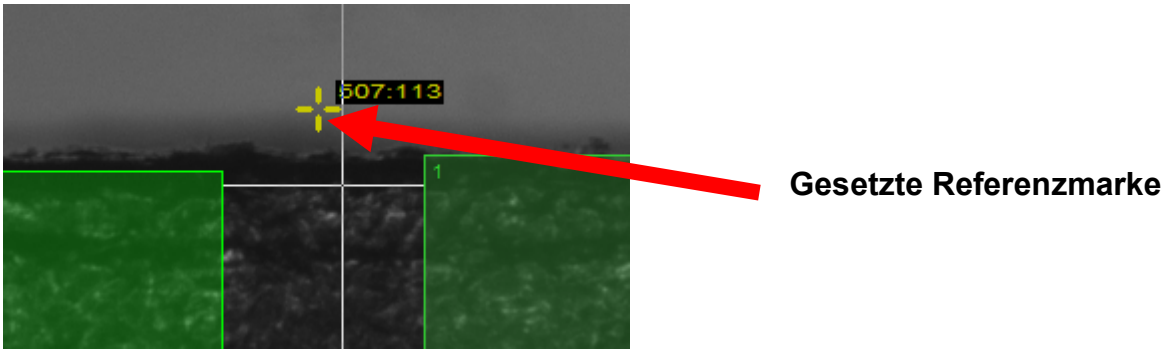
Wählbare Messmerkmale

6.0.3 Pattern Referenz Marker

Für ein Messbild und all seinen Patterns kann im **Edit-Modus** des Messbildes ein Referenz - Marker gesetzt werden. Alle später erkannten Objekte haben zu dessen Position eine Distanz die in der Prüfplanung als Entfernung zum **Referenz-Marker** gewertet werden kann.

Um den **Referenz-Marker** an beliebiger Stelle zu setzen, doppelklicken sie auf einen nicht durch Patternbereiche überlagerten Bildbereich. Sollte ein Pattern die Position überlagern muss dieses vorübergehend zur Seite geschoben werden um den Marker setzen zu können.

Edit Bereich des Messbildes:

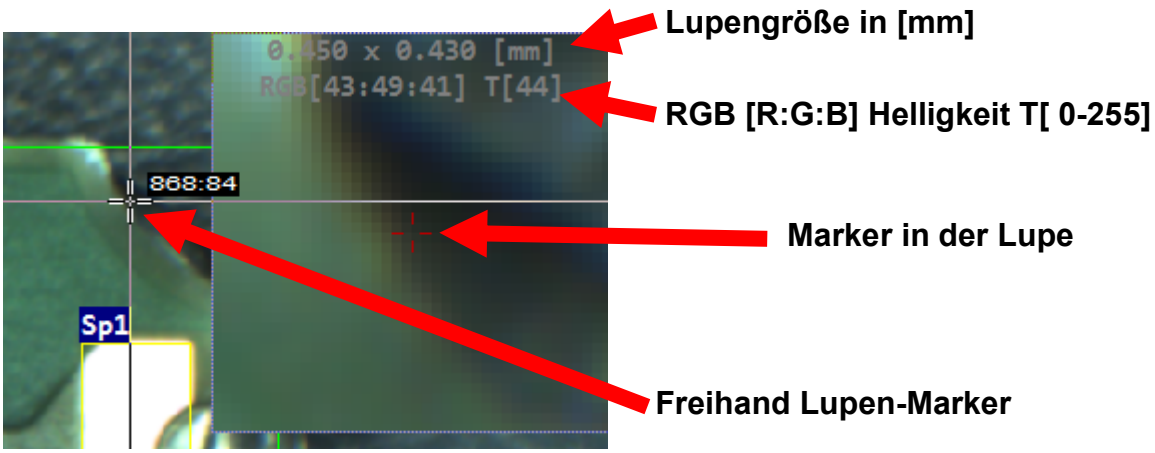


Wie die Referenzposition in der Prüfplanung erhalten wird, ist unter **LUA-Skript** in der **Event** -Beschreibung **OnUsrDat** dargestellt.

6.0.4 Freihand Lupen Marker

Ist die Lupenvergrößerung über das Messbild **PopUp Menü Punkt** PixelMeasure/ShowLens Aktiv, kann der Marker mit der Maus frei verschoben werden um die Fokuslupe pixelgenau auch mir den Messbändern zusammen Synchron zu bewegen.

Lupenbereich des Markers:

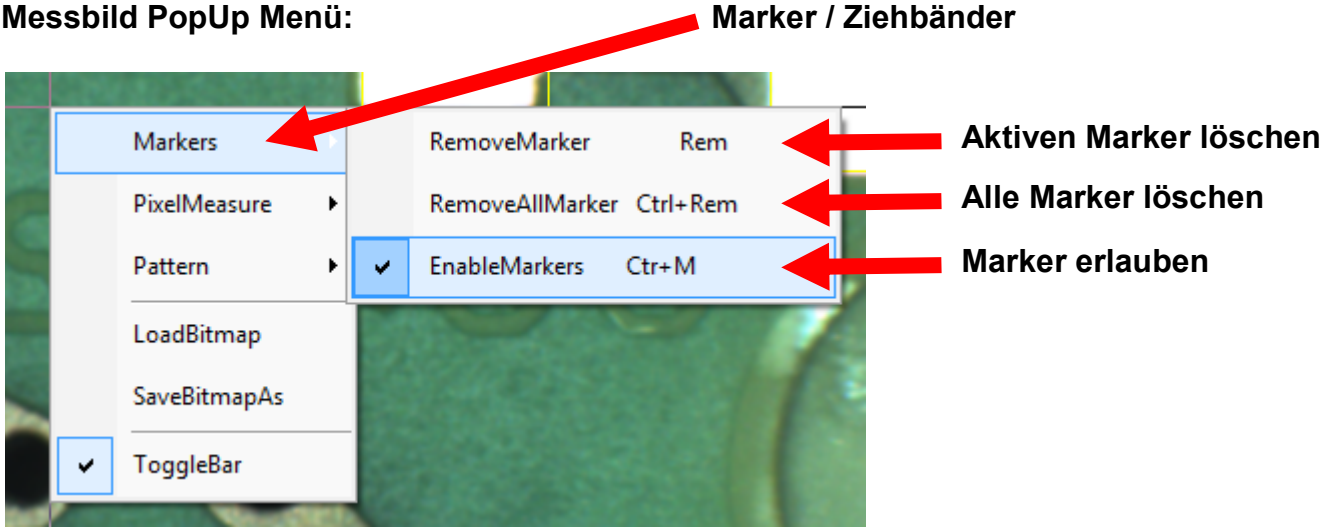


Wie die Lupenposition in der Prüfplanung übergeben wird, ist unter **LUA-Skript** in der **Event** -Beschreibung **OnImage** dargestellt.

6.0.5 Marker Ziehbänder

Ziehbänder werden mit der Maus vom Benutzer/Operator oder Prüfplaner verwendet um mit der Hand **Längenbestimmungen** im **Messbild** herstellen zu können. Der **LUA-Skript** Prüfplan kann mit bestimmten Funktionen Marker und Kalibrierungen automatisch herstellen. Dies wird unter **Lua-Prüfplanung** noch genauer dargestellt. Zum aktivieren der Ziehbänder muss der Bildmodus ohne Vergrößerung aktiv sein, außerdem müssen Ziehbänder und Marker explizit aktiviert werden Default (aktiv). Verwenden sie im Messbild die rechte Maustaste um das **PopUp-Menü** hervorzubringen.

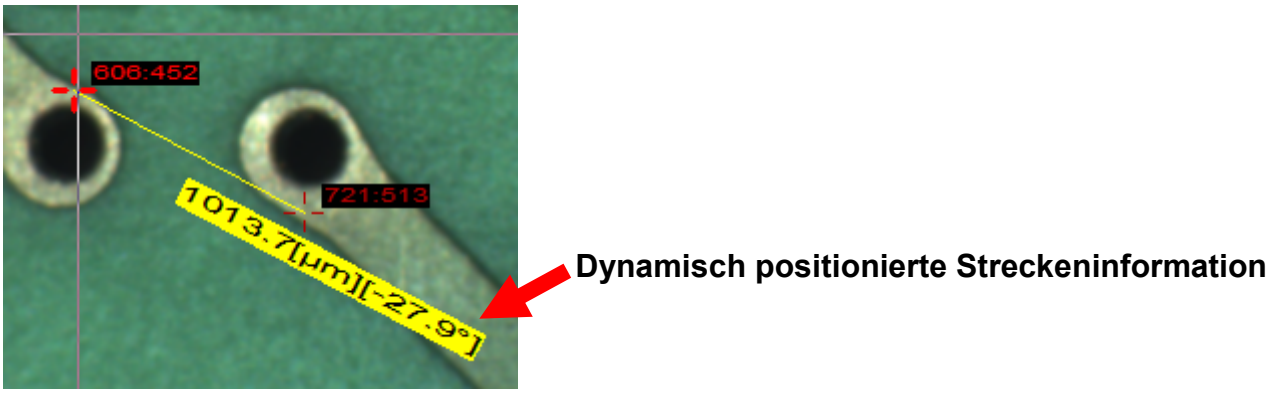
Messbild PopUp Menü:



Nachdem Marker eingeschaltet sind, und die Vergrößerung (**Stretch**) abgeschaltet, kann mit einem Doppelklick ein erster **Marker** positioniert werden. Jeder weitere Doppelklick fügt einen neuen Markierungspunkt hinzu der mit einer Messlinie den Vorgänger verbindet.

So können **Mess-Gerüßte** errichtet werden um in einem Messbild verschiedene Positionen miteinander vergleichen zu können. Wird ein **Marker** mit der Maus über einen anderen bewegt und losgelassen verbinden sich beide zu einem gemeinsamen Messpunkt, so kann ein **Mess-Gerüßt** geschlossen werden. Ein Marker kann aktiviert werden wenn dieser angeklickt wird.

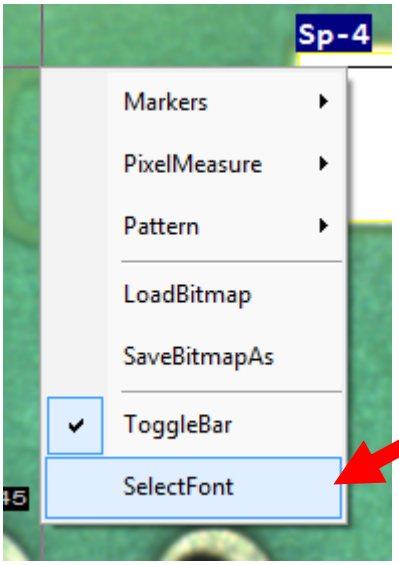
Beispiel : zweier Markierungen mit Angabe der Winkellage und der Entfernung in [µm]



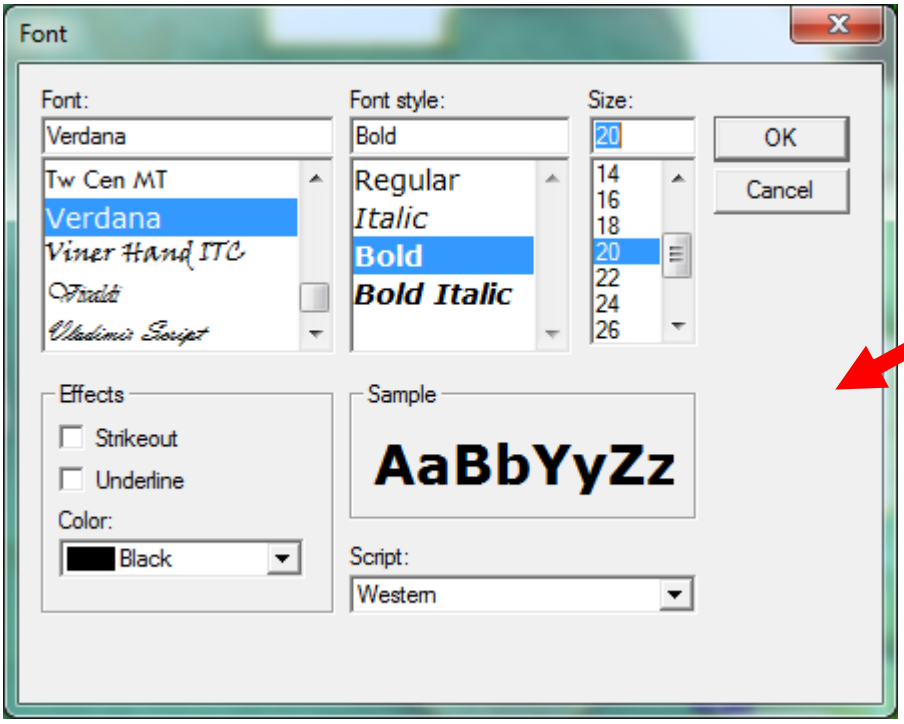
6.0.6 Messbild Font Auswahl

Das Messbild unterstützt alternative Fonts, insbesondere wenn sehr viele Ergebnisse im Messbild angezeigt werden sollen bietet sich die Verkleinerung der Fonts an. Für die Betrachtung des Messbildes mit großem Abstand zum Monitor ist es nützlich die Fonts zu vergrößern, oder wenn Mehrere Ergebnistexte mit Umbruch dargestellt werden kann der **New Currier** Font zur Anwendung kommen. Default ist **Consolas**. Der Ausgewählte Font wird mit dem Speichern des Prüfplanes in der lokalen **Windows Registry** hinterlegt und ebenso beim Laden dort gelesen.

Verwenden sie im Messbild die rechte Maustaste um das **PopUp-Menü** hervorzubringen.



Font Auswahldialog aufrufen

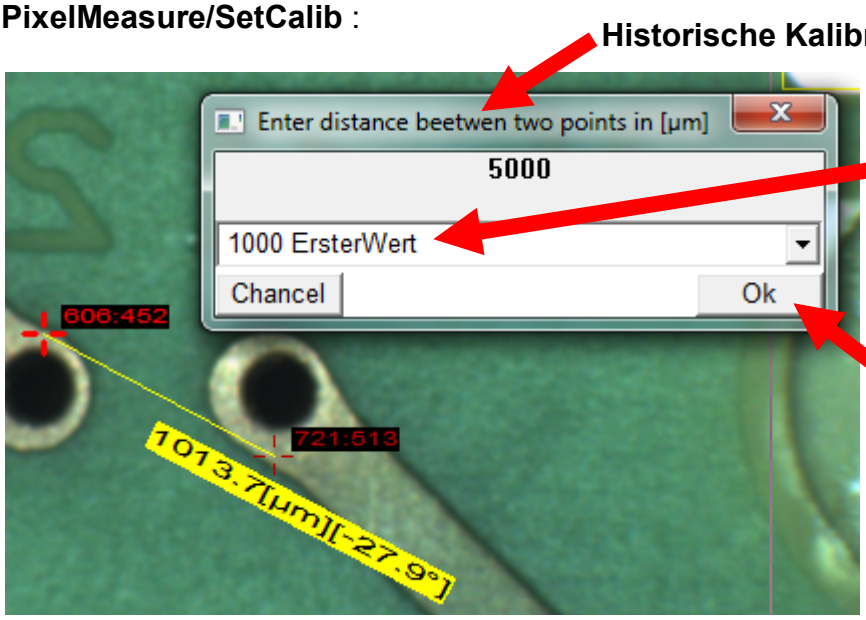


Standard Font -Dialog

6.0.7 Messstreckenkalibrierung

Nachdem zwei Punkte wie im Bild oben zu sehen markiert sind, kann dieser Strecke eine bekannte Anzahl Mikrometer zugewiesen werden. Verwenden sie den **PopUp-Menüpunkt**

PixelMeasure/SetCalib :



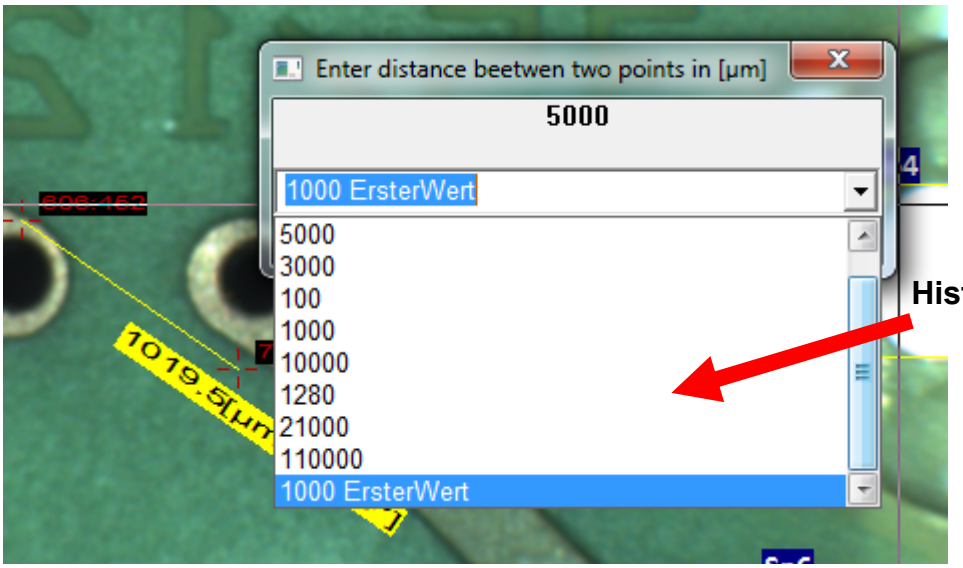
Nach der Zuweisung werden alle Darstellungen sich auf diese Messgröße beziehen. Die aktuelle Kalibrierung wird in der Prüfplandatei ***.set** nach der Speicherung des Prüfplans hinterlegt.

Auszug aus der Datei *.set

```
[FrameInfo]
Calib=1.000000e-001
```

Um eine **bestehende** Kalibrierung zu aktivieren Verwenden sie den **PopUp-Menüpunkt**

PixelMeasure/GetCalib :



7.0.0 LUA Skript Prüfplanung

Der wesentliche Teil der Prüfplanung holt aus einem Messbild Informationen hervor die an die Prozessleitstelle zeitnahe weitergereicht werden.

Die modifizierbare Prüfplanerstellung wurde implementiert, um im Klartext aus dem Messbild die benötigten Informationen zu gewinnen. Es stellt sich heraus dass immer dieselben Verfahren mit unterschiedlichen Betriebsparametern auf das Messbild wirken.

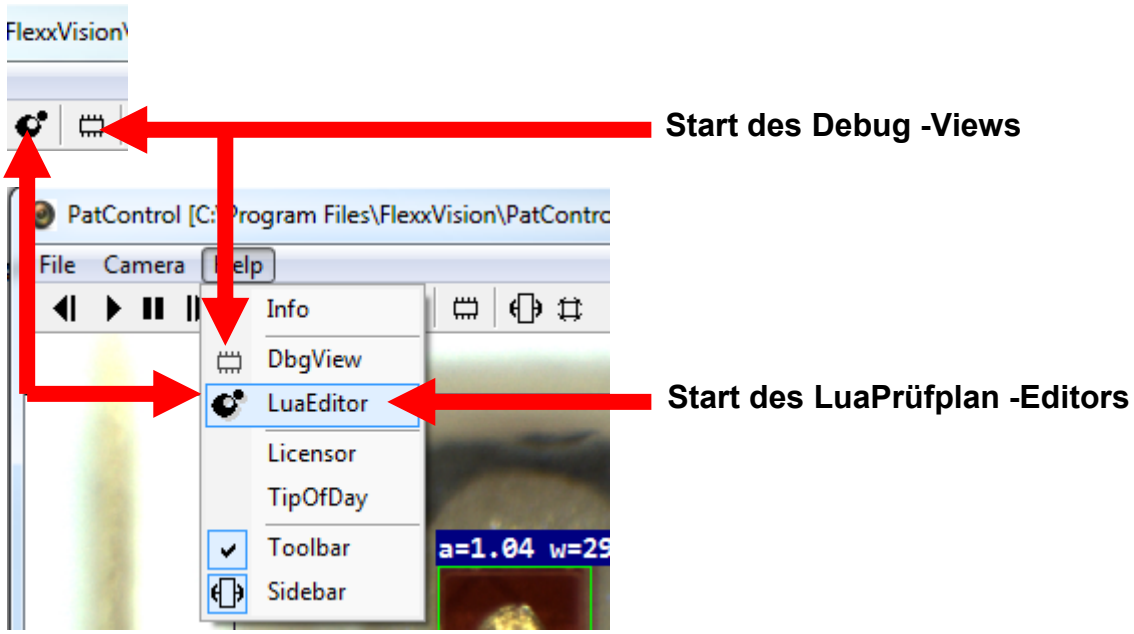
Diese Verfahren der Bildbereichsauswertung werden Bildpunkt/Pixel –Sensoren genannt. Die Skriptsprache LUA ist ein auf C basierender Lexikalpaser der aufgrund seiner minimalen Semantik schnelle Ablaufergebnisse liefert.

PatControl hat einen eigenen Editor der mit einem Haltepunkt –Debugger die Kontrolle und Fehlerdiagnostik über die Prüfplanentwicklung gewährt. **PatControl** gibt sämtliche Fehlersituationen über eine **Debug-Konsole** aus, dazu gehören Skriptfehler sowie auch alle anderen Fehlersituation der gesamten Software, damit sind Zusammenhänge bei Störungen nachvollziehbar. Fehlermeldungen werden durch umfangreiche Klartext-Benachrichtigungen ausgegeben. Eine **Error-Code** Suche entfällt.

Der **LUA-Prüfplan** erhält während des Programmablaufs Benachrichtigungen über Ereignisse.

```
1) Wenn ein neues Bild geliefert wurde.
2) Der Benutzer ein Datenfeld verändert.
3) ein Pattern ausgerechnet werden soll.
4) ein PixelSensor zusätzliche asynchrone Informationen liefert.
5) Eine LUA alternative TCP oder RS232/432 Kommunikation Ereignisse meldet.
```

PatControl Menü/Toolbar:



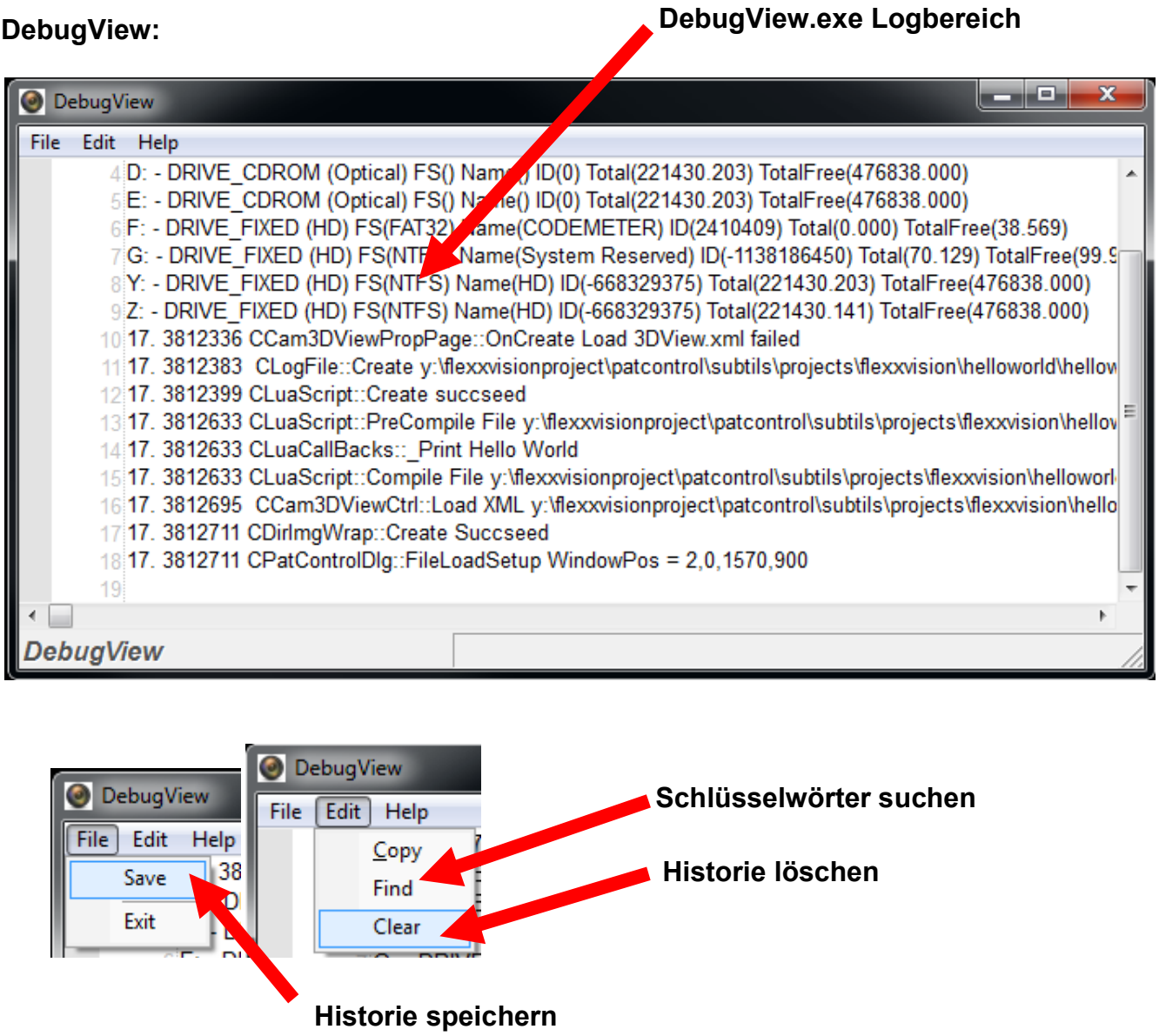
7.0.1 Debug Konsole (DebugView)

Sämtliche Fehler und Status –Ausgaben der gesamten Software werden auf einem Separaten **Debug-Terminal** ausgegeben.

Während der Prüfplanung und Prozesskommunikation wird empfohlen diese Konsole parallel zum Hauptprogramm geöffnet zu halten.

Die Ausgaben sind Systemglobal auch über externe Event-Logger empfangbar. Jedes LUA Skript wird den Status der Kompilierung, und ggf. Fehlermeldungen mit Zeilennummern als **System-Event** Senden die mit dem Programm **DebugView.exe** eingesehen und auch gespeichert werden können. **DebugView** fügt jedem Ereignis eine Zeit in Millisekunden an, um den zeitlichen Ablauf der Messvorgänge einsehen zu können. Hiermit lässt sich also auch das Laufzeitverhalten z.B. bei der **SPS –Kommunikation** nachweisen. Nach einem Programmstart von **PatControl** werden zuerst Informationen über die Laufwerke des Systems angezeigt sowie über den Status der Programm-Initialisierung.

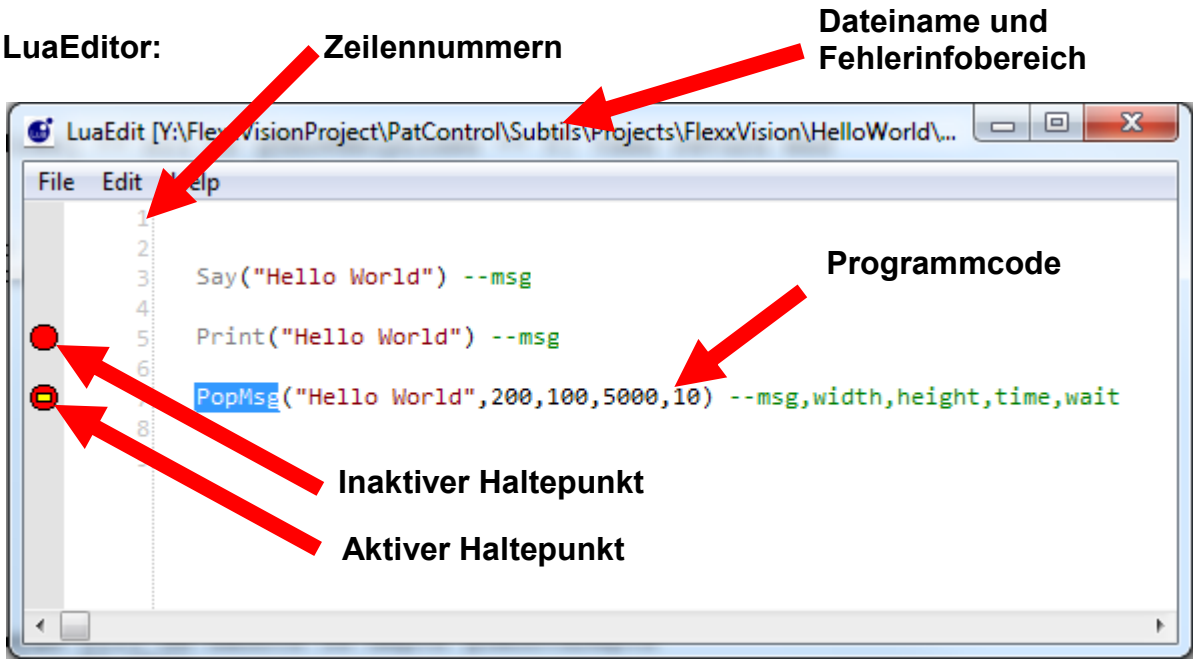
DebugView:



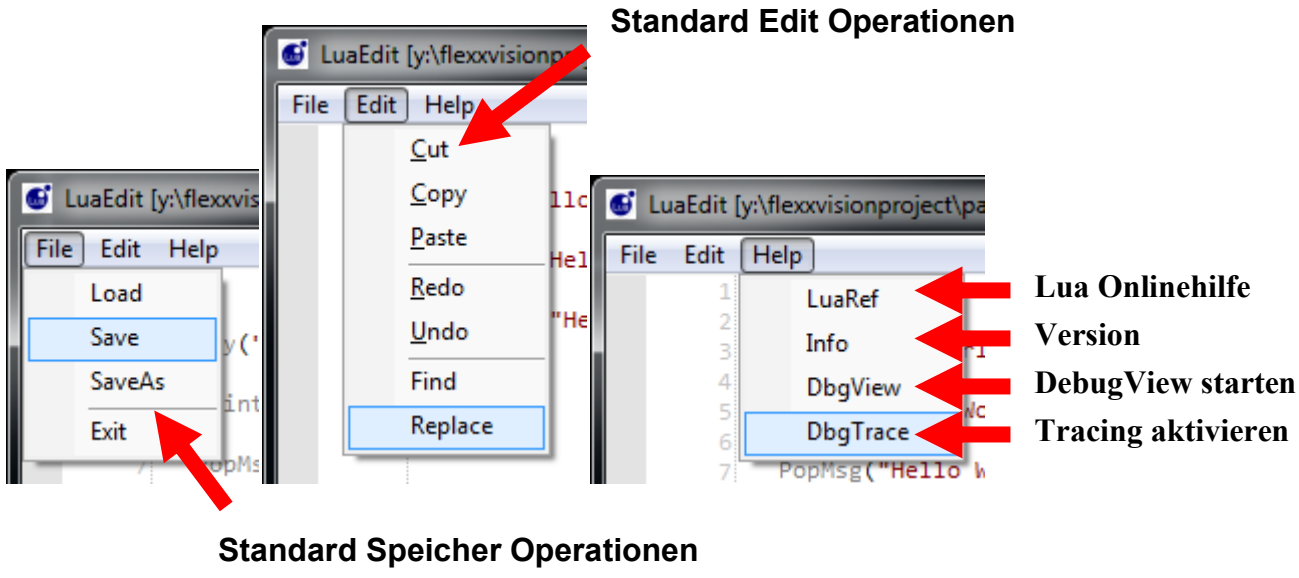
7.0.2 LUA-Prüfplaneditor

Der integrierte **LUA-Editor** ist direkt mit dem aktuellen Prüfplan gekoppelt. Das zu einem Prüfplan gehörende **LUA-Skript** hat immer den Dateinamen des Prüfplans, jedoch ist die Dateiendung nicht ***.set** sondern ***.lua** Der Menüpunkt **Save** wirkt direkt auf den aktiven Prüfplan und löst eine Kompilierung des Textlichen Inhaltes aus.

Es wird empfohlen vor der Veränderung eines Skriptes dieses mit dem Menüpnkt **SaveAS** über das EditorMenü zu sichern. Danach wird eine Kopie des Skriptes erzeugt, der Editor beschreibt dann die Kopie und wirkt nicht mehr auf das Original, es erfolgt dann keine Kompilierung, da der neue Dateiname nicht dem des Prüfplans entspricht.



7.0.3 LUA-Prüfplan Menü



7.0.4 LUA-Prüfplan Haltepunkte

Wenn ein Prüfplan abläuft, also einkommende Bilder den Prüfplan anstoßen/triggern ist es möglich einen Haltepunkt auf einer Programmzeile zu positionieren.

Dazu wird mir der Maus eine Programmzeile **selektiert**, und die Funktionstaste **F9** gedrückt. Am linken Bildschirmrand des Editors erscheint eine rote **Markierung**. Wenn der Interpreter diese Zeile verarbeitet wird dieser dort den Programmablauf stoppen, damit wird auch die Beschaffung neuer Bilder und sämtlicher Aktionen der Messsoftware angehalten.

Mit dem Mauscursor ist es an dieser Stelle möglich den Inhalt lokaler Variablen anzeigen zu lassen indem der Mauscursor über eine lokale Variable bewegt wird.

Mit der Taste **F5** wird der Programmablauf wieder freigegeben bis der Interpreter den markierten Haltepunkt erneut erreicht. Es ist möglich mehrere Haltepunkte zu setzen.

Im Prüfplaneditor kann der Programmablauf Zwangs gestoppt werden, indem
Eine Ausnahme ausgelöst wird durch gleich zeitiges drücken der Tasten

LCONTROL + LSHIFT +C

7.0.5 LUA-Prüfplan Tracing

Um einen historischen Durchlauf aller Variablen im **DebugView** ausgeben zu lassen, kann unter erheblicher Rechenlast der Menüpunkt **Help/DbgTrace** aktiviert werden.

Alle Variablen -Inhalte die während des durchlaufen des Prüfplanes aufkommen werden in der Eventausgabe des **DebugView** angezeigt. Alternativ dazu können zusätzlich Haltepunkte den Durchlauf an diesen Stellen anhalten, und mit **F5** fortgesetzt werden.

7.0.6 LUA-Prüfplan SyntaxHighlight

Im Programmverzeichnis der **PatControl** Installation befindet sich eine Steuerdatei für den Syntaxparser des Editors mit dem Namen: **LuaEdit.xml** hier ist es möglich die Farbgebung und schematische Darstellung der Programmtexte ggf. anzupassen.

Alle bekannten Funktionen die vom **LUA-Skripthost** unterstützt werden sind dort aufgeführt. Das löschen der Datei oder die Fehleingabe von Direktiven kann erhebliche Programmstörungen bewirken. Änderungen sollten nur unter Vorbehalt der Originalkopie durchgeführt werden.

7.0.7 LUA-Meldungen

Der Prüfplaner kann Meldungen im **DebugView** ausgeben, oder eine **PopUp** Nachricht erscheinen lassen, sowie Messergebnisse an den **VisionServer** senden, der diese Visualisiert. Außerdem ist es möglich **Texte Warnungen** oder **Hinweise** mit der Sprachsynthese auszugeben, um z.B. bei der Einrichtung einer Kamera Informationen akustisch zu melden ohne den Bildschirm sehen zu müssen. Meldungen an den **VisionServer** erfolgen insbesondere durch die Ergebnisübermittlung mit **PatResult** und **PatStatus** diese Ergebnisübermittler übertragen auch grafische Markierungs-Informationen.

```
1) Print          (Textausgabe auf dem DebugView)
2) Say            (Akustische –Textausgabe)
3) PopMsg         (Animiertere Text -Schildnachricht auslösen)
4) SetPatResult   (Ergebnis an den VisionServer senden)
5) PatStatus      (Status an den VisionServer senden)
6) SetPatCalib    (Messkalibrierung an den VisionServer senden)
```

7.0.8 LUA-Callbacks

Callbacks sind Aufrufe des **VisionServers** an das **Lua-Skript** diese gliedern sich in 8 Gruppen:

```
1) Neues Bild trifft ein.
2) Pattern soll berechnet werden.
3) Operator ändert Daten in den Pattern-Parameterbaum.
4) Operator ändert Daten im Benutzer-Parameterbaum.
5) Ein PixelSensor meldet eine Staffel von Ergebnissen.
6) Parallele TCP-Verbindung liefert Daten.
7) Parallele RS232/432 Verbindung liefert Daten.
8) Zählerstand abfragen für Good/Bad/Amount
```

7.0.9 LUA-Pixelsensoren

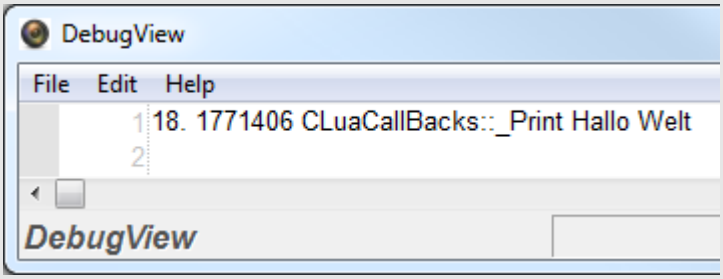
PixelSensoren wirken direkt auf das Messbild und geben numerische Informationen über die Eigenschaften von Bereichen die frei wählbar sind, und auch insbesondere auf einen vom Prüfplaner eingestellten **PatternBereich** ausgerichtet werden können.

```
1) LabelImage       (Liefert zusammenhängende PixelObjekte)
2) FingerImage      (Statistische Vektor -suche nach Übergängen)
3) WormImage        (Kantenverfolgung zur Geometrie Vektorisierung)
4) SigmaImage       (Schärfenwert Sigma -Rauschen eines Bildteils)
5) ThresholdImage   (Helligkeit eines Bildbereiches)
6) StartDfs         (Pyramidial Tiefenpixelanalyse)
```

Parameter die in rechteckigen Klammern angegeben sind, sind optional.
Wird ein optionaler Parameter angegeben, müssen alle optionalen Parameter übergeben werden. Im Folgen wird die Parametrisierung aller Funktionen tabellarisch dargestellt.

8.0.0 LUA Textausgabe Funktionen

Um während der Prüfplanung den Inhalt von Daten oder Aktionen zu signalisieren ist die Funktion **Print()** integriert. Hiermit werden Textnachrichten im **DebugView** ausgegeben.

Print	Parameter	Rückgabe
Print("Hallo Welt") 	`msgtext (string)`	(nil)

Für Tonausgaben während der Prüfplanung oder der Kameraeinrichtung ist es nicht immer möglich den Bildschirm einsehen zu können **Say()** Erzeugt eine Audioausgabe, die den Übergabe- Text in Phoneme wandelt, und akustisch über das Soundsystem ausgibt.

Say	Parameter	Rückgabe
Say("Hallo Welt")	`msgtext (string)`	(nil)

Hinweise oder Ereignisse die dem Betrachter des Messbildes erscheinen sollen werden mit **PopMsg** Dargestellt. Hierbei kann Breite, Höhe und Meldungstext sowie die Anzeigezeit in Millisekunden angegeben werden plus eine Verzögerung bis das **PopMsg** erscheint.

PopMsg	Parameter	Rückgabe

PopMsg(“Hallo Welt“,320,200,1000,0)
 
	`msgtext, (string)`
	`width, (number)`
	`height, (number)`
	`showtime, (number)`
	`delaytime (number)`
		(nil)

8.0.1 LUA Prozessinformations Funktionen

SetPatResult ist eine von zwei möglichen Versionen einen **Prozessstatus** auf das Messbild zu geben, als auch auf die Hardware zu leiten. Hierfür werden verschiedene Modifizierer als Parameter verwendet um die Darstellungsform auf dem Messbild zu bestimmen, und den **Index** und die **Wertigkeit** einer Messinformation in die Transferstruktur eintragen zu lassen.

Die **Prozessdatenstruktur** ist oben unter Kommunikation aufgeführt. Diese Transferstruktur dient ausschließlich der Daten-Übertragung an die Peripherie und kann bis zu 128 Einträge pro Messbild enthalten.

Der Eintrag **Data**[0] der Datenübertragungsstruktur soll dem Prozessstatus vorbehalten sein, und kann nur mit der Funktion **SetPatStatus** gesetzt werden. Das bedeutet dass mit **SetPatResult** nur die Einträge **Data**[1-127] beschreibbar sind. Überschreitung oder Unterschreitung des Parameter **Index** löst eine Fehlermeldung aus.

Die Funktion dient auch dazu, ein Rechteck und einen Text zu übermitteln der auf dem Messbild angezeigt wird, hierzu kann für das Rechteck und die Farbe bestimmt werden. Zusätzlich kann das Rechteck verwendet werden, um damit einen Kreis zu zeichnen oder Start und Endposition zweier Marker zu übertragen die dann mit der Messstrecken-Vorrichtung eine Spannlinie mit Längenangabe und Winkelgrad erzeugt. Außerdem kann für die Histographische Darstellung im alternativen Messbild der Name des Messsegmentes bezeichnet werden ohne diesen Namen wird kein Segment in dieser Darstellung gezeigt.

Jedes **SetPatResult** wirkt auf ein **Pattern** das zuvor im **MessbildEditor** angelegt worden ist. Dessen **Alias-Name** wird dem Prüfplan mitgeteilt, und muss vom Prüfplaner vorgehalten werden oder als Konstante im Parameter **alias** des **SetPatResults** eingetragen sein, damit die Messdaten mit diesem Pattern intern verbunden werden können.

Beispiel einiger RGB Farben:

```
RED    = 0x0000ff
GREEN  = 0x00ff00
YELLOW = 0x00ffff
BLUE   = 0xff0000
GRAY   = 0xaaaaaa
```

Mögliche Darstellungen:

```
PAT_BOX       = 0     (Zeichnet eine Box)
PAT_ANGLEDRAW = 8     (Zeichnet eine Winkelfunktion)
PAT_LINE      = 24    (Zeichnet eine Linie zwischen den PatResults)
PAT_CROSS     = 32    (Setzt einen rotes Markierungskreuz)
PAT_CIRCLE    = 64    (Zeichnet einen Kreis)
PAT_LFTTXT    = 256   (Textausrichtung zur Darstellung)
PAT_RIGTXT    = 512   (Textausrichtung zur Darstellung)
PAT_TOPTXT    = 1024  (Textausrichtung zur Darstellung)
PAT_BOTTXT    = 2048  (Textausrichtung zur Darstellung)
PAT_MEASURE   = 4096  (setzt einen Messstrecken Markierung)
```

8.0.2 SetPatResult

SetPatResult hat verschiedenste Kombinationen von Parametern ein Standardfall kann wie folgt aussehen, es ist an dieser Stelle hilfreich in die Prüfpläne der Beispiele nachzusehen welche anderen Aufrufversionen für die jeweilige Messaufgabe am besten geeignet scheint.
Das Messwerte-Rechteck geht hier von **100,100** bis **200,200** und ist 100**[Pixel²]** groß.
Der Messwert ist **4711** und soll im **Index** 1 der Transferstruktur abgelegt werden.

SetPatResult	Parameter	Rückgabe
SetPatResult("1Pat",100,100,200,200,PAT_BOX, "Hallo Welt","Balken1",1,4711,GREEN) 		
SetPatResult("1Pat",100,100,200,200, PAT_ANGLEDRAW,"Hallo Welt","Balken1", 1,4711, GREEN) 	alias, (string) left, (number) top, (number) right, (number) bottom, (number) status, (number) infomsg, (string) infoname, (string) index, (number) infoval, (number) color (number)	(nil)
PatResult("1Pat",100,100,200,200,PAT_CROSS, "HalloWelt","Balken1", 1,4711, GREEN) 		

8.0.3 SetPatStatus

SetPatStatus signalisiert den Generalstatus der Messung, dieser kann im oberen Teil des Messbildes eine Textausgabe darstellen die zumeist **IO** oder **NIO** lautet :
(In Ordnung/Nicht in Ordnung oder **PASS** Bzw. **FAIL)**.

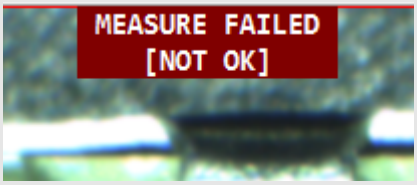
Die Farben für die Texthintergründe werden automatisch erzeug, für den **NIO** Fall rot, und für den **IO** Fall grün.

Der Generalstatus einer Messung gibt Auskunft darüber ob eine Messung überhaupt erfolgreich war und nachfolgende Werte gültige Messwerte enthalten. Würde ein Objekt den Messbereich verdecken, könnten die pixelbasierenden Sensoren Maximale Ausschläge liefern die nicht mehr in einer statistischen Verarbeitung verwendet werden dürfen.

Der Fehlercode wird in der Transferstruktur im Element **Data**[0] abgelegt, ein Wert größer Null erzeugt eine rote Textausgabe sonst eine grüne. Somit kann nicht nur übermittelt werden ob das Teil außerhalb der Toleranz liegt sondern auch ob die Messung überhaupt gültig ist, diese muss vom Prüfplaner mit der Prozessleitstelle abgesprochen werden.

Der Parameter **errmsgstr** bezeichnet den Namen des Fehlers, der Parameter **description** bezeichnet den Status **OK / NOT OK / NIO / IO** als Text.

SetPatStatus	Parameter	Rückgabe

SetPatStatus(“1Pat”,4711," MEASURE SUCCSEED "," [OK] ")
 
SetPatStatus(“1Pat”,4711," MEASURE FAILED"," [NOT OK] ")
 
	`alias, (string)`
	`errcode, (number)`
	`errmsgstr, (string)`
	`description (string)`
		(nil)

8.0.4 SetPatCalib

SetPatCalib Setzt einen neuen Kalibrierfaktor für einen Pixel, das bedeutet das nach zwei Messpunkt Erkennungen deren Strecke bekannt ist z.B. 122**[µm]** die Pixelstrecke zwischen linke obere und rechte untere Ecke kann wie folgt aussehen:

```
left   = 528
top    = 478
right  = 650
bottom = 478
```

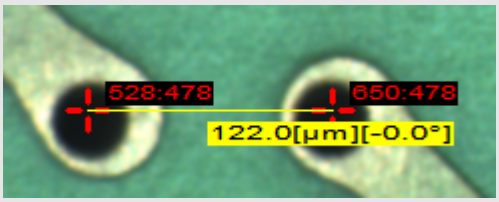
Durch folgenden Rechengang beschafft wird die Messstrecke:

```
Xlen = right-left
Ylen = bottom-top

Pixelstrecke = Math.sqrt(  xlen*xlen + ylen*ylen )
```

calibnumber = 122**[µm]** / **Pixelstrecke** ergibt den neuen Kalibrierwert pro Pixel.

Die so ausgelöste Kalibrierung überschreibt direkt die vom Benutzer/Operator manuell Ausgewählte **Kalibrierung** für die Laufzeit der Programmausführung. Wird der Prüfplan gespeichert wird diese neue Kalibrierung beibehalten bis der Benutzer/Operator manuell erneut eine andere Kalibrierung auswählt. Daher ist es sinnvoll einen Kalibrier -Vorgang nicht ständig auszulösen, sondern in den Benutzerdefinierten Parameterbaum einen **Button/Schalter** dafür anzulegen und einen Hinweis anzuzeigen wenn die Kalibrierung erfolgreich war. Das Speichern des Prüfplans kann nur der Anwender manuell ausführen.

Das Schlüsselwort **Math.sqrt** ist eine der **LUA** - spezifischen Funktion aus deren Mathelibrary. Sie können über die **LUA -Programmierung** auch andere Librarys hinzufügen und alle bekannten **LUA -Optionen** verwenden.

SetPatCalib	Parameter	Rückgabe
SetPatCalib(“1Pat”,528,478,650,478,calibnumber) 	alias, (string) left, (number) top, (number) right, (number) bottom, (number) calibnumber (number)	(nil)

9.0.0 LUA-Callback Events

Um einen Messbild -synchronen Programmablauf zu erzeugen wird der Prüfplan vom **VisionServer** für jede benötigte Aktion angestoßen / Getriggert . Die Ereignisse kommen entweder vom Benutzer durch Eingabe von Daten in den Parameterbäumen, von der Bild – Akquirierung oder einem im **LUA-Skript** -Datentransfer, der von einer Kommunikation oder einem Pixelsensor ausgelöst wurde. Datentransferrückrufe sind immer asynchron zur aufrufenden Funktion, während Bilderneuerungen und Benutzereingaben synchron zum Prüfplanablauf erfolgen. Der **VisionServer** ruft die Standard -**Callbacks** in folgender Reihenfolge pro neues Messbild auf, und wartet bis diese beendet werden.

```
1) OnImage     Meldet neues Messbild ist eingetroffen
2) OnPat       Meldet das dieses Pattern verarbeitet werden soll
3) OnPatCnt    Meldeabschluss und Abfrage der Erfolgszähler
```

9.0.1 OnImage

Ist das erste **Event** das nach dem Eintreffen eines neuen Bildes ausgelöst wird, es wird die Größe des Bildes übertragen wobei **left** und **top** derzeit immer Null ist, **right-left** liefert die aktuelle Breite des neuen Bildes und **bottom-top** die aktuelle Höhe. Der Wert **state** ist Null. **LUA** verwendet globale Variablen, es ist nützlich die Bildbreite und Bildhöhe zu merken.

OnImage	Parameter	Rückgabe
OnImage(0,0,1024,768,Mx,My,0)	left, (number) top, (number) right, (number) bottom, (number) xoff, (number) yoff, (number) state (number) (NULL)	(nil)

9.0.2 OnPat

Jeder Prüfplan verfügt über mindestens ein vom Prüfplaner angelegtes Pattern, ohne dem wird kein Skript ausgeführt. Ein Prüfplan kann mehr als ein Pattern enthalten und jedes hat einen eindeutigen **Alias -Namen** der unter **Pattern-Parameter** beschreiben ist. Dies ist der einzige **Bezeichner** der die Patterns unterscheidet. Für jedes angelegte Pattern wird dieses **Event** einmal aufgerufen, hier ist die Stelle wo alle **PixelSensoren** für dieses Pattern ausgelöst werden, so kann individuell auf jeden Parameterstamm und jeden **Bildbereich** explizit eine Auswertung erfolgen. Die für jedes Pattern benötigten Variablen können vorher bei den **Parameter Events** in globalen Variablen vorgehalten, und mit dem Namen den **alias** liefert indiziert werden.

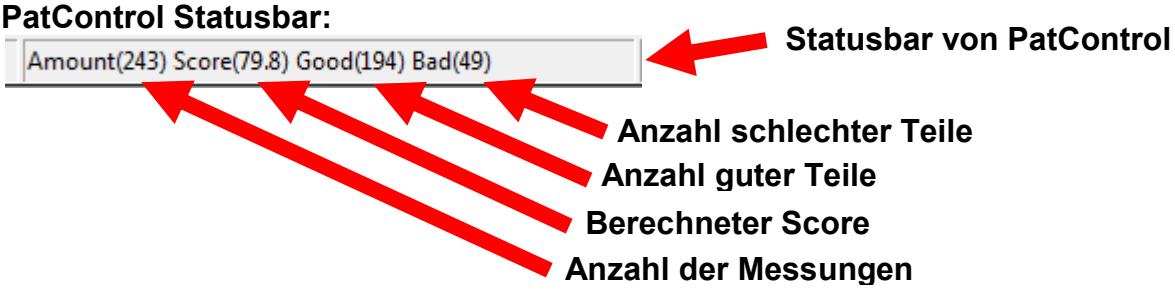
OnPat	Parameter	Rückgabe
OnPat (“1Pat”)	Alias (string)	(nil)

9.0.3 OnPatCnt

Nachdem alle Pattern durchlaufen wurden ergibt sich ein Verarbeitungszähler. Für jedes neue Bild kann ein **Summen-Zähler**, für jede erfolgreiche Bewertung ein **Gut- Zähler** und für jedes schlechte Teil ein **Schlecht-Zähler** inkrementiert werden. Aus dem Verhältnis **Anzahl / gut-schlecht** ergibt sich der **Score** der gesamten Messung über die Dauer der Beobachtung. Dieser **Score** wird unter dem Messbild angezeigt.

Der **Event** markiert auch das Ende der gesamten Bewertung, nach diesem **Event** folgt wieder **OnImage.** Die Ausnahme ist wenn ein Benutzer neue Daten in den Parameter-Bäumen eingibt, diese Events können immer eintreffen.

PatControl Statusbar:



Der Parameter **state** ist normal immer Null, wenn der Operator im Menü von **PatControl** den **MenüPunkt** Camera/ResetCounter auslöst werden alle Zähler zurückgesetzt ausgenommen ist der **Logbuch-Zähler** der unten beschrieben wird. Der **Kamera-Trigger** wird Hardware und Softwareseitig ebenso zurückgestellt.

PatControl Menü:

OnPatCnt	Parameter	Rückgabe
OnPatCnt (1)	`state (number)`	SumCnt (number), BadCnt (number), GoodCnt (number),

9.0.4 OnChangePat

Dieser **Event** tritt ein wenn der Benutzer/Operator Daten in dem **Pattern-Parameterbaum** verändert, der Prüfplan **geladen/gespeichert**, der **Editmodus** des Messbildes verlassen, oder ein **LuaSkript** des Prüfplans gespeichert wird. Somit ist sichergestellt das dieses **Event** zuerst Daten an den Prüfplan liefert bevor andere **Events** aufgerufen werden.

Alle im **Parameterbaum** vorkommenden festen Variablen werden an das **Event** übertragen. Es ist an dieser Stelle erforderlich für jeden **Alias-Namen** die spezifischen Daten in ein globales Array abzulegen so das später mit dem **Alias-Namen** der in **OnPat** übergeben wird die Daten wieder zur Verfügung stehen.

OnChangePat	Parameter	Rückgabe
OnChangePat (“1Pat”, 10,10, 100,100, 1000, 10, 45, 30, 30, 25, 25, 128, 90.5, 17.4, 0xffffff, 1)	**alias, (string)** **left, (number)** **top, (number)** **right, (number)** **bottom, (number)** **time, (number)** **maxfind, (number)** **angle, (number)** **width, (number)** **height, (number)** **maxtw, (number)** **maxth, (number)** **threshold, (number)** **score, (number)** **clutter, (number)** **rgb, (number)** **algorithm (number)**	(nil)

Nicht immer werden alle Informationen benötig ein Ablegen in ein globales Array kann über den **Alias-Namen** wie folgt „gemerkt“ werden:

```
gPatLeft[alias]      = left
gPatTop[alias]       = top
gPatRight[alias]     = right
gPatBottom[alias]    = bottom
gPatThreshold[alias] = threshold
gRgb[alias]          = rgb
```

Das Abrufen der Variablen über den Indizierer -Eintrag **alias** z.B. in **OnPat(alias)** erfolgt durch Zuweisung der Daten in lokale Variablen :

```
local PatLeft   = gPatLeft[alias]
local PatTop    = gPatTop[alias]
local PatRight  = gPatRight[alias]
local PatBottom = gPatBottom[alias]
local rgb       = gRgb[alias]
```

9.0.5 OnUsrDat

Dieser **Event** tritt ein wenn der Benutzer/Operator Daten im **Benutzer-Parameterbaum** verändert, der Prüfplan **geladen/gespeichert**, der **Editmodus** des Messbildes verlassen, oder ein **LuaSkript** des Prüfplans gespeichert wird. Somit ist sichergestellt das dieses Event Daten an den Prüfplan liefert bevor andere Events aufgerufen werden.

Der **Event** erhält zwei Strings den Variablen -Namen und den Variablen –Wert. Es empfiehlt sich direkt als erstes den Parameter **value** in eine Zahl zu wandeln via:

```
local numVal   = tonumber(Value)
```

im weiteren wird jeder Variablen –Name mit einer **if then else** Kaskade auf Zutreffen geprüft, denn das **Event** wird für jeden einzelnen Parameter einmal aufgerufen, bei sehr langen Parameterbäumen können also durchaus mehr als ein Dutzend Aufrufe eintreffen die jedoch im Millisekunden- Bereich abgearbeitet werden.

Beispiel: Den Prüfplankalibrierfaktor abzuprüfen, und in eine globale zu Variable übertragen:

```
if(Name == "Calib")  then gCalib = numVal end
```

Diese Abfragen -Kaskade wird solange für jede einzelne benutzerdefinierte Variable ausgeführt bis die Iterationen durchlaufen sind, dann hat jede globale Variable ihren Wert aus dem **Benutzer-Parameterbaum** erhalten.

OnUsrDat	Parameter	Rückgabe
OnUsrDat (“Threshold”,“128“)	`name,  (string)` `value  (string)`	(nil)

Beispiel einer Prüf-Kaskade :

```
local numVal  = tonumber(Value)

if(Name == "Calib")              then gCalib          = numVal
elseif(Name == "ThresLeft")      then gThresLeft      = numVal
elseif(Name == "ThresWorm")      then gThresWorm      = numVal
elseif(Name == "SmoothDepth")    then gSmoothDepth    = numVal
elseif(Name == "ShowHelpLines")  then gShowHelpLines  = numVal
elseif(Name == "XRef")           then gXRef           = numVal
elseif(Name == "YRef")           then gYRef           = numVal
end
```

Die Koordinaten für den **Pattern -Referenz** Marker lauten **XRef** und **YRef** , über den Namen **Calib** kann der aktuelle Multiplikator für die **Pixel- Kalibrierung** nach Mikrometer erhalten werden.

10.0.0 PixelSensoren

PixelSenoren wurden bereits mehrmals erwähnt und stellen den **Kern** der Operationen dar. Der Aufruf eines **PixelSensors** kann ein **Event** auslösen das jedoch synchron zum Aufruf erfolgt um Ergebnisse an den Prüfplan übermitteln zu können. Daher werden diese **Events** getrennt, und passend zur jeweiligen **Sensor-Funktion** beschrieben.

Alle Pixeloperationen wirken auf eine **GrayScaleBitmap.** Farbbilder werden vor der Prüfplanzuführung in ein **8Bit Grauwertbild** gewandelt. Wenn es erforderlich ist Farben zu unterscheiden kann mit der virtuellen Kamera **VisualFilter** die gewünschte Information vor verstärkt werden, oder der PixelSensor **ColorImage** verwendet werden.
(Im Ram sind alle Bilder grau)

Jedem Sensor kann ein bestimmter Bildbereich zur Untersuchung zugewiesen werden, diese Bereiche werden mit **left,top,right,bottom** bezeichnet, oft ist das Rechteck eines **Patterns** der Rahmen in dem ein Pixelsensor wirken soll.

Die meisten Bildpunktoperationen sind in Assembler erstellt und greifen direkt auf dem Bildspeicher zu, der Skripthost verhindert durch Parameter -Überprüfung das keine ungültigen Bildgrenzen wie z.B. negative Werte oder Ordinaten außerhalb der Bildschirmgröße erfolgen. Dennoch ist es zu vermeiden das Variablen die einem **PixelSensor** übergeben werden ungültige Bereiche ansprechen, dies wird stets unerwünschte **Resultate** liefern.

Mit Hilfe der der Funktion **SetPatResult** kann auf dem Messbild gezeichnet werden um Ergebnisse oder Übergaben für Debug -zwecke dort anzuzeigen. Es wird empfohlen Zielrechtecke und Ergebnisrechtecke optional im Messbild anzeigen zu lassen, damit eine **Prüfplanwartung** leichter möglich ist. Es empfiehlt sich für diesen Zweck eine **ComboBox** im Benutzer definierten Parameterbaum anzulegen die es ermöglicht **Hilfsausgaben** ein oder ausschalten zu können.

Pixeloperationen sind rechenlastig, es ist darauf zu achten das möglichst nur die Bereiche zur Untersuchung herangezogen werden die auch in Frage kommen, die Patternausrichtung im Editor –Modus des **Messbildes** ermöglicht eine **Vorselektierung** bestimmter Ausschnitte die für eine Untersuchung in Frage kommen. Ein **Labeldetektor** der zur Aufgabe hat alle Bildpunkte eines bestimmten Helligkeitsbandes zu Objekten zu verbinden wird auf einem gänzlich weißen Bild das alle Verbindungskriterien enthält **totlaufen**, hier kann nur noch ein **timeout** behilflich sein, damit eine derartige Operation ein zeitnahes Ende findet.

Das ein Sensor nicht mehr reagiert kommt nicht vor. Im Normalfall wo das Bild natürliche Übergänge enthält, werden die **Pixeloperationen** erhebliche Rechenleistungen erreichen.

Wichtig: Alle **PixelSensoren** erzeugen in einer Hilfs -Bitmap für den Verlauf der jeweiligen Operationen Linien oder Punkte, nur wenn diese angezeigt werden, ist eine Verfolgung der der Arbeitsweisen auf dem Ur-Bild möglich. Ist der Schalter **ShowHelpLines** im benutzer-definierten Parameterbaum angelegt, werden alle **PixelSensoren** Ihre Operationen puffern, und im Messbild durch eine weiße Linie darstellen wenn der Schalter auf „on“ steht. Der Vorgang ist dann Rechenlastig, aber für die Positionierung und Parametrisierung der Sensoren unablässig.

10.0.1 LabelImage

Dem sogenannten Labeln ermöglicht es Bildpunkte die sich in einem **Helligkeitsbereich** befinden und sich gegenseitig berühren zu sammeln bis eine Gruppe erstellt ist die keine weiteren Verbindungen zu den Nachbarn aufweist. Angewandt auf ein Bild können tausende von rechteckigen Bereichen gefunden werden die als **Inseln** im Bild vorkommen. Diese Funktion ähnelt in gewisser Hinsicht dem Farbeimer eines Mal- Programms in umgekehrter betrachtungs- Abfolge, so das **Objekte** im Bild erkannt werden, und eine **Unterscheidung** zwischen **Pixelbrücken** und **Hintergrund** gefunden wird. Der Labeldetektor ist in der Lage hunderttausend Objekte pro Sekunde zu identifizieren. Diese Datenmengen kommen schnell zusammen wenn größere Substrate wie **Wafer** in Bezug auf **Lotbumb's** untersucht werden.

Das **Labeling** ist eine der Grundfunktionen auf die eine Bildverarbeitung basiert. Es geht nicht unbedingt darum ähnliche Objekte aufwendig mit der Bild in Bild suche zu identifizieren da dies viel zu lange dauern würde, und nicht unter allen Umständen erfolgreich ist.

Das kleinste mögliche **Labelobjekt** hat 2 Pixel, ein Bild mit vielen **Pigmenten** würde sehr viele Ergebnisse liefern, dies möchte man im **Vorfeld** unterdrücken durch abgestimmte Prozess Parameter die eine Mindestmenge an Bildpunkten beschreiben soll. Ein wichtiger Parameter zur suchmengen- Optimierung ist **minwight** es handelt sich um das **Gewicht** eines Objektes je mehr Pixel es zusammengefasst hat desto „schwerer" ist es. Ein weiteres Kriterium ist die Mindestbreite und Mindesthöhe, damit sich das Gewicht auf eine Dimension beziehen kann. Diese Parameter werden **minwidth** und **minheight** bezeichnet. Um eine weitere Geometrie einzubringen kann man den Durchmesser des Objektes angeben, genannt **mindiameter.** Die vierte Dimension kann sinnvoll wirken um die Suche frühzeitig abbrechen zu können, dazu wird eine zeitliches **timeout** in Millisekunden übergeben, damit eine Suche bei totalem Pixelschluss aller Bildpunkte nicht erst nach einigen Dutzend Sekunden ein Ende findet, sondern nach mindestens x Millisekunden abbricht. Außerdem muss der **Helligkeitsbereich** angegeben werden der beschreibt welche Bildpunkte in Betracht gezogen werden, in einer Grauwertebitmap ist der dunkelste Pixel = 0 und der hellste = 255, eine Suche nach Bildpunkten zwischen 128 und 255 ignoriert einen ggf. dunkeln Hintergrund was erheblich zur **Beschleunigung** der Untersuchung beiträgt. Die dafür verwendeten Parameter sind **thresholdmin** und **thresholdmax.**

Erst eine zum Bild passende **Parameterauswahl** kann optimale Ergebnisse liefern, es ist sinnvoll durch Voruntersuchungen die Helligkeitsschwellen zu messen dafür gibt es weitere Funktionen die unten erläutert werden. Die Parameter für das **Labeln** von Bildbereichen sollten im Benutzerdefinierten Parameterbaum vorkommen, damit es **Einwirkmöglichkeiten** von außen gibt ohne Programmieren zu müssen.

Oft kommt es vor, das ein Bereich zersplittert vorliegt, ein Schraubenkopf durch seinen Schlitz z.B. als zwei Objekte erkannt wird. Es gibt einen umfangreichen weiteren Parameter, ist dieser **> 0** wird versucht alle gefunden Nachbarrechtecke miteinander zu verschmelzen wenn die **Außenkanten** nicht weiter entfernt sind als durch den Parameter **spc** (spacing) vorgegeben wurde. Ist der Wert gesetzt wird optional ein **Event** ausgelöst das jedes Rechteck und ein Nachbarrechteck an das **LUA-Skript** übergibt, damit mit der **Prüfplan** entscheiden kann, welche Rechtecke miteinander verschmolzen werden und welche nicht.

Die Funktion **LabelImage** liefert zwei Parameter zurück der erste ist die **Anzahl** gefundener Objekte, der zweite ist eine **Liste** von Werten die über die Eigenschaften eines jeden Objektes Auskunft gibt. Diese Liste muss in einer Schleife solange durchlaufen werden bis alle Daten erhoben wurden. Ist die Liste leer liefert die Anzahl den wert **nil** zurück.

Beispiel:

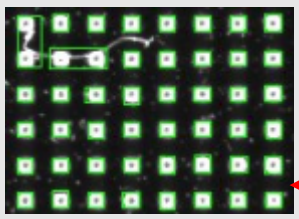
```
cntlab,tablab = LabelImage(0,0,100,100,128,255,3,6,8,4,4,0,5555)

if(cntlab == nil) then
  return
end
```

Liefert **cntlab** jedoch einen Wert ungleich **nil** enthält die Liste immer **5 Werte** für ein Objekt die wie folgt angelegt sind:

```
local i;
for i=1, cntlab, 5 do
     left      =  tablab[i+0]    linke  Rechteckseite
     top       =  tablab[i+1]    obere  Rechteckseite
     right     =  tablab[i+2]    rechte Rechteckseite
     bottom    =  tablab[i+3]    untere Rechteckseite
     wight     =  tablab[i+4]    Gewicht des Rechtecks
end
```

10.0.3 Labeling Parameterdarstellung

LabelImage	Parameter	Rückgabe
LableImage (0,0, 100, 100, 128, 255, 3, 6, 8, 4, 4, 0, 5555 9999999)	left, (number) top, (number) right, (number) bottom, (number) threshold, (number) thresholdmax, (number) [mindiameter], (number) [minwight], (number) [minwidth], (number) [minheight], (number) [spc], (number) [timeout] (number) [maxwight] (number)	(number), (list)



Jedes Objekt wurde erkannt (grüne Rechtecke)

10.0.4 Labeling Rechteckverschmelzung

Ist der Parameter **spc** (spacing) angegeben findet eine einfache Verschmelzung der Nachbarrechtecke an den Seiten eines berechneten Labelobjektes statt das sich dadurch steht's vergrößert. Die zersplitterten Teile werden durch diesen Vorgang aus der Liste gefundener Objekte entfernt. Es kann jedoch vorkommen, dass so eine einfache Verschmelzung den Anforderung nicht genügt. Dazu kann im **LUA**-Prüfplan – Skript ein **Event** angelegt werden, wird dieses vorgefunden findet eine Iteration aller Rechtecke statt, wobei dem **Event** immer zwei Rechtecke die in Frage kommen, übergeben werden. Returniert der **Event** eine **1** werden beide Rechtecke verschmolzen und das zweite verworfen, dieser Vorgang ist auch in einigen Beispielen enthalten. Eine **0** Rückgabe löst keine Verschmelzung aus. Bei sehr vielen Rechtecken kann dieser aufwendige Vorgang die Verarbeitungszeit negativ beeinflussen, eine genaue Bereichsberechnung geht derartigen Operationen voraus. Es ist oft vorteilhaft einen Bereich mehrmals zu labeln mit immer feineren Bereichsangeben, auch kann die Bildhelligkeit für diese Bereiche dynamisch ermittelt werden mit der Funktion **ThresholdImage** die eine weitere Basis der Bildverarbeitung darstellt und unten beschrieben wird.

OnRcCheck	**Parameter**	**Rückgabe**
OnRcCheck (spc, 0, 0, 10, 10, 11, 11, 22, 22)	**rspc, (number)** **r1left, (number)** **r1top, (number)** **r1right, (number)** **r1bottom, (number)** **r2left, (number)** **r2top, (number)** **r2right, (number)** **r2bottom (number)**	(number)

Damit es möglich ist im Prüfplan festzustellen das die Iteration beendet ist wird nach diesem Durchlauf ein weiteres Event ausgelöst das das Ende der **Verschmelzung** signalisiert und die Dimension der Abmaße vom ersten bis zum letzten Rechteck überträgt. Über dieses **Event** ist es möglich Hilfsvariablen im Prüfplan wieder zu „**Reseten**". Beide Events sind Optional und müssen nicht Bestandteil der Prüfplanung sein.

OnRcReset	**Parameter**	**Rückgabe**
OnRcReset (width (number) height (number))	**Width, (number)** **Height (number)**	(nil)

10.1.0 FingerImage statistische Konturerkennung

In einem Messbild kann schon eine Bereichseingrenzung durch die Patterns definiert werden, diese können aber nur im groben umschreiben wo sich ein Objekt befinden wird, in einigen Fällen kann so ein Bereich den gesamten Bildschirm betreffen. Objekte im **FieldOfView** (Sichtfeld der Kamera) müssen in Ihrer Lage vor der genauen Untersuchung der Merkmale zuerst gefunden werden. Die Objekte haben vorab eins gemeinsam, sie heben sich durch ihren ersten Außenumriss vom Hintergrund ab. Diese Übergänge können von einem Finger (**Abtastsensor**) gefunden werden. Diese **Suchstrahlen** können in vier Richtungen abgesendet werden, von **links nach rechts**, **von rechts nach links**, **von oben nach unten** und von **unten nach oben**.

Damit ist es möglich, ein zu erwartendes Objekt in seiner Lage zu erfassen und als erstes Ergebnis eine genaue Position des Prüflings festzulegen auf dem alle weiteren Messungen basieren. Dies gehört zum Konzept der immer feiner werdenden Bereichseingrenzung gern auch **LevelOfDetail** in der **3D–Welt** bezeichnet, überträgt sich das auch auf **2D** Bilder.

Die **Suchstrahlen** haben eine zueinander liegende Abstands -**Granularität** und eine Suchbreite sowie eine Suchlänge. Aus der **Granularität** der nebeneinander ausgesendeten Strahlen und der Suchbreite ergibt sich auch die Anzahl der Kontaktpunkte an einer **Übergangs-Kannte**. Da diese ggf. rau oder wellig ist, wird das statistische Mittelmaß der Kontaktierung mit dem Übergang als Messwert geliefert.

Die vier Suchrichtungen für den Parameter **direction** lauten:

```
UP              = 0x00
RIGHT           = 0x01
DOWN            = 0x02
LEFT            = 0x03
```

Der Suchbereich **left,top,right,bottom** beschreibt ein Rechteck indem die Suche durchgeführt wird, entsprechend der Suchrichtung ist dieses Rechteck zu positionieren.

Die zu erwartenden Helligkeitsmerkmale des Überganges werden durch **thresholdmin** und **thresholdmax** eingegrenzt und können vorher dynamisch ermittelt werden.

Der Parameter **linesz** ermöglich explizit die Stärke eines Fingers anzugeben und damit auch auf die Anzahl der ausgesendeten Strahlen bezüglich der gesamten Sensorbreite. Der Wert **steps** bezeichnet die Lücke zwischen den Suchstrahlen, mit dem Parameter **sigma** wird die Empfindlichkeit bestimmt ein Wert von 3.0 ist ausreichend.

Zu allen Parametern empfiehlt es sich einen im Benutzerdefinierten Parameterbaum angelegten Variablen-Block für Zuschläge der verwendeten Parameter anzulegen, damit eine Optimierung der Erkennung auch durch den Benutzer/Operator ohne Programmierkenntnis erfolgen kann.

Die Funktion **FingerImage** liefert zwei Parameter zurück, der erste ist die **Anzahl** der Einträge, der zweite ist eine **Liste** von Werten die über die Eigenschaften der Kantenkollision Auskunft gibt. Ist die Liste leer liefert die Anzahl den wert **nil** zurück.

Beispiel:

```
cntfng,tabfng = FingerImage(0,0,500,500,128,255,UP,1,0.3)

if(cntlab == nil) then
  return
end
```

Liefert **cntfng** jedoch einen Wert ungleich **nil** enthält die Liste immer **4 Werte** für ein Objekt die wie folgt angelegt sind:

```
mw     =  tabfng [0]    Mittelwert der der gefunden Pixelkante
sigma  =  tabfng [1]    Sigma für die Signalstärke der Pixelkante
max    =  tabfng [2]    Maximaler Strahlen -Ausschlag
min    =  tabfng [3]    Minimaler Strahlen -Ausschlag
```

Insbesondere ist immer der Wert **mw** von Interesse, dieser beschreibt an welcher Position der Übergang im definierten Rechteck am besten war. Die Werte **max** und **min** bezeichnen lediglich das einer der Strahlen seine Erste oder Letzte Kollision an diesen Positionen fand. Das **sigma** liefert die Signalstärke aus der der Mittelwert abstammt, ist dieser Wert schwach, kann ggf. eine **Nachuntersuchung** fällig werden.

10.1.1 FingerImage Parameterdarstellung

FingerImage	Parameter		Rückgabe
FingerImage (0, 0, 500 500 128 255 0x00 8 16 3)	left, top, right, bottom, thresholdmin, thresholdmax, direction, [linesz], [steps], [sigma]	(number) (number) (number) (number) (number) (number) (number) (number)	(number), (list)

Statistische Position des Übergangs (gelbe Linie)

10.2.0 WormImage Geometrie -Vektorisierung

Die Konturverfolger -Funktion **WormImage** ist ein komplexer Pixeloperator der sich dem Namen entsprechend durch das Bild bewegt und nach einem Kontakt sucht dem er folgen kann. Für dieses Verhalten benötigt der Sensor ein Startpunkt und ein Zielpunkt, ist der Raum zwischen beiden Punkten leer läuft der Sensor direkt zum Zielpunkt und liefert keine geometrischen Informationen.

Im Kollisionsfall entsteht an einer Kontaktstelle der Entscheidungsnotstand für den **PixelSensor** eine neue dem Ziel entsprechende Richtung aufzugreifen und gleichzeitig der Übergangskante im Bild zu folgen. Die Lösung des Problems ist der **Pledge-Algorithmus.** Entdeckt haben soll diesen Algorithmus ein zwölf jähriger Junge namens John Pledge deshalb nennt man ihn den **Pledge-Algorithmus**. Der Rechenweg beschreibt das Vorgehen von der „Maus im Labyrinth“ um aus einer offenen oder geschlossenen Geometrie möglichst effektiv entweichen zu können. **FlexxVision** entwickelt auf dieser Basis parallel neue Sensoren die in dieser Fassung noch nicht integriert sind. Umfangreiche Literatur und Verfahren sind im Internet nachlesbar.

Der Sensor erhält als Parameter die **startx**, **starty** Koordinate, und für das Fahrziel **endx**, und **endy** als Ziel -Koordinate zu dem sich der Sensor bewegen soll. Außerdem wird eine maximal Lauflänge **runlen** vorgegeben an der der Sensor die Suche abbricht, da dieser imstande ist, dauerhaft einer geschlossenen Geometrie zu folgen. Für jeden Schritt den der Sensor im Bild ausführt wird ein **Event** ausgelöst an dem eine Liste von Parametern zum aktuellen Status der Weg suche übergeben wird. Returniert dieses **Event** einen Wert **< 0** ist die Suche beendet. Wird eine **1** Zurückgegeben bedeutet dies, dass der aktuelle „Fahrweg“ durch eine Helligkeitsstufe blockiert ist. Bei **0** Rückgabe ist der Weg frei und der Sensor nimmt den direkten Weg zum Ziel wieder auf. Die Feststellung das der Weg blockiert ist entsteht dadurch, dass der Sensor sein nächstes Pixelziel aus dem Koordinatensystem des Bildes an das **Event** übergibt, es obliegt dem Prüfplanentwickler an dieser Stelle mit der Funktion **ThresholdImage** diese Koordinate zu prüfen um die Entscheidung einer Kollision festzulegen.

Der Sensor kann sich in 8 Richtungen bewegen die Namen für diese Vektoren entsprechen den Himmelsrichtungen:

North,NorthEast,East,SuedEast,Sued,SuedWest,West,NorthWest

Folgt der Sensor einem Vektor wird für jeden Fortschritt auf der Achse ein Zähler in einem internen Vektorarray erhöht. Nach der Umfahrung einer Geometrie kann aus dem Zählerstand der 8 Vektoren die numerische Geometrie des Objektes entnommen werden, die wie folgt lauten kann.

60 North, 22 East , 71North 102 West Jeder Zahlenwert entspricht also die Lauflänge auf der Achse und somit die Pixelstrecke.

Nach dem Auslösen des Sensors läuft die Eventverarbeitung, erst danach kehrt die Funktion zum Aufrufer zurück. Alle Informationen werden im **Eventbereich** festgestellt.

10.2.1 Worm Event

OnWorm	Parameter	Rückgabe
OnWorm (tabelle)	`tabelle (number)`	(number)

Beispiel einer Eventbasierenden Konturoperation:

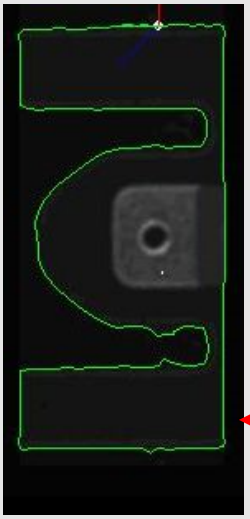
```
function OnWorm(direction)

  --direction counters
  local n    = direction[1]  --NorthCounter
  local ne   = direction[2]  --NorthEastCounter
  local e    = direction[3]  --EastCounter
  local se   = direction[4]  --SuesEastCounter
  local s    = direction[5]  --SuedCounter
  local sw   = direction[6]  --SuedWestCounter
  local w    = direction[7]  --WestCounter
  local nw   = direction[8]  --NorthWestCounter
  local x    = direction[9]  --TestposX
  local y    = direction[10] --TestposY
  local len  = direction[11] --RunLenCounter
  local col  = direction[12] --CollisionCounter
  local vx   = direction[13] --Kopfposition X
  local vy   = direction[14] --Kopfposition Y


  if(ThresholdImage(x-2,y-2,x+2,y+2,0) > 128) then
   return 1 -Collision measured
  end

  return 0 --Freeway
end
```

ThresholdImage wird hier auf jeden Bildpunkt in einer 5x5 Matrix großen Helligkeitsbeschaffung angewandt, ist der Wert größer als 128 wird eine Kollision auf diesen Koordinaten vermutet, das nächste Event wird eine andere Position zur **Auswegssuche** liefern. Sollte der **Suchkopf** festgefahren sein steigt die Anzahl **RunLenCounter**, während die Koordinaten nicht denselben Zuwachs erleben.
ThresholdImage wird weiter unten dargestellt.

Es ist möglich anhand der Zählerstände die Untersuchung zu beenden oder die Zählerstände an globale Variablen zu übertragen. Weniger Operationen im **Event** erhöhen die Ablaufgeschwindigkeit erheblich. Der Einsatz des Sensors erfolgt nach einer vorausgehenden Bereichsuntersuchung um möglichst nicht irrelevante Bildbereiche kostspielig zu analysieren.

10.2.3 WormImage Parameterdarstellung

WormImage	Parameter	Rückgabe
WormImage(0, 0, 100, 100, 1000)	startx, (number) starty, (number) targetx, (number) targety, (number) runlen (number)	(nil)



Nummerischer Objektumriss (grüne Linie)

Worms zeichnen sich dadurch aus, extrem scharfe **Konturen** berechnen zu können wie es mit allgemeinen Filtertechniken nicht machbar ist, durch genaue Auswertung und passende Parametrisierung können so sehr feine Strukturen im Messbild herausgearbeitet werden.

Eine Suche kann mit mehreren **Worms** erfolgen die aus verschiedenen Richtungen zu einem Zentrumspunkt wirken. Nicht selten konnten auf diese Weise schwierigste **Merkmalserkennungen** zuverlässig durchgeführt werden. Die Anwendung dieses freien Sensors erfordert eine genaue Untersuchung der Möglichkeiten die sich mit diesem Messmittel bilden. Es ist empfehlenswert die dazu verwendeten Parameter in dem Benutzerdefinierten Parameterbaum vorzuhalten, um die Entwicklung eines **Lösungsweges** wesentlich zu erleichtern und dem Benutzer/Operator Zugriff auf die Einstellungen zu ermöglichen, ohne Programmierkenntnis einsetzen zu müssen.

10.2.4 Image Recognition

In der **Bild in Bild** Erkennung werden in einem Untersuchungsvorgang alle Helligkeitsübergänge und Konturen als **Merkmale** des Bildpunkteverlaufes gesammelt und gespeichert.

Ein weiterer Prozess reduziert anschließend die Menge der **Merkmale** und überführt diese in einem skalierten Merkmalsraum. Hier liegen die Daten in Gruppen klassifiziert vor z.B. unterteilt in Mengengewichte und Fließrichtung. Mit der Funktion **ImageRecognition** kann ein Bildausschnitt z.B. **(1024x768)** mit bis zu **65[Hz]** hochgerechnet werden.
Im Anschluss wird eine Liste aller identifizierten **Merkmale** geliefert.

Die Suche nach gleichartigen Merkmalsgruppen ermöglicht die Erkennung von ähnlichen **Objekten** im Bild. Außerdem kann durch die Charakterisierung der Konturen hochgenau eine **Lage -Bestimmung** von Kanten und Übergängen ermittelt werden.

Die Funktion **ImageRecognition** kennt 2 zugrunde liegende Verfahren um Daten zu liefern. In Punktform mit Fließrichtung und in Rechteckform als Zusammenfassung von Vektorisierten Merkmalen.

Die Merkmalserhebung kann durch einen dynamischen Threshold automatisch erfolgen unter Ausschluss aller Bildpunkte die kleiner als Min-Threshold und größer als MaxThreshold sind, damit kann die Verarbeitungsleistung drastisch gesteigert werden wenn im Vorfeld ein Helligkeitsband angegeben wird.

Zudem ist es möglich die Automatische Threshold -Untersuchung zu umgehen und nur Bildpunkte zu Verarbeiten die in diesem Helligkeitsband liegen, um das zu erreichen wird dem Verarbeitungstype ein Schalter „nicht“ übergeben **RECO_DYN.**

```
RECO_LAB = 0x01 Modus Recheck – Sampling
RECO_VEC = 0x02 Modus Vector -Sampling
RECO_DYN = 0x80 Modus Dynamic –Sampling
```

Nach der Auslösung der Funktion wird eine Liste von Informationen geliefert die je nach Schalterstellung für **RECO_VEC** Punktdaten enthält in der Form **x , y, d** wobei **d** für die Ausrichtung des Vektors steht der die möglichen Werte von D ~= 0-7 annehmen kann und die Verlaufsrichtung wie folgt angibt North = Oben :

North,NorthEast,East,SuedEast,Sued,SuedWest,West,NorthWest

Die Werte **x,y** bestimmen die Position des Vektors im Bild. Für das Verfahren **RECO_LAB** werden Rechtecke geliefert die eine Breitenstärke von 1 selten überschreiten, und je nach Ausrichtung über die Lage des markierten Überganges Auskunft gibt. Die Rechteckdaten enthalten außerdem das Pixelgewicht das die Dimension bestimmte. Die Daten liegen
In dieser Folge vor:

left , top , right , bottom , wight

10.2.5 ImageRecognition Parameterdarstellung

ImageRecognition	Parameter	Rückgabe
ImageRecognition (0, 0, 1024, 768, RECO_DYN+ RECO_LAB)	left, (number) top, (number) right, (number) bottom, (number) thresmax, (number) thresmin, (number) type (number)	(number), (list)
		

Modus RECO_LAB

Modus RECO VEC

Beispiel RECO_LAB:

```
cntlab,tablab = ImageRecognition(0,0,1024,768,255,10,RECO_LAB+RECO_DYN)
if(cntlab == nil) then
return
end
```

RECO_LAB Liefert **5 Werte** für ein Objekt, für **RECO_VEC** werden 3 Werte geliefert.
Ist der Rückgabewert cntlab == nil wurden keine Daten erhoben.

```
local i;
for i=1, cntlab, 5 do
left   = tablab[i+0] linke Rechteckseite
top    = tablab[i+1] obere Rechteckseite
right  = tablab[i+2] rechte Rechteckseite
bottom = tablab[i+3] untere Rechteckseite
wight  = tablab[i+4] Gewicht des Rechtecks
end
```

Beispiel RECO_ VEC:

```
for i=1, cntlab, 3 do
x   = tablab[i+0] X Position des Vektors
y   = tablab[i+1] Y Position des Vektors
dir = tablab[i+2] Dir Ausrichtung des Vektors
end
```

10.3.0 ThresholdImage

Dies ist der wohl wichtigste Sensor, der jedoch keinerlei Auskunft über die **Geometrie** eines Messbereiches liefert, sondern Auskunft über dessen **Helligkeit**.

Die **Bildbereichs -Helligkeit** ist für sämtliche Messoperationen insbesondere zur dynamischen Schwellwert –Berechnung ausschlaggebend. Über die Helligkeit eines Bereiches oder eines Pixels können die anderen **PixelSensoren** in Ihrer Parametrisierung Verbessert werden, so dass alle Eingaben aus dem Benutzerdefinierten Parameterbaum als Zuschlag dienen für die Werte, die **ThresholdImage** in der Bildbereichs -Voruntersuchung lieferte.

Die Parameter für diese Funktion **left**,**top**,**right**,**bottom** beschreiben den zu messenden Bereich alle Pixelhelligkeiten werden in diesem Bereich addiert und durch die Anzahl der Additionen dividiert, somit liefert die Funktion den Mittelwert der **Bereichshelligkeit.** Durch verkleinern des Bereiches auf 0,0,1,1 kann ein einzelnes Pixel abgefragt werden, das jedoch in der Praxis kaum einen Sinn ergibt. Der letzte Parameter ist ein Hilfsmittel um Helligkeiten z.B. einen schwarzen Hintergrund auszuschließen und wird als **thresholdmin** übergeben, dieser Wert ist meist **Null**.

10.3.1 ThresholdImage Parameterdarstellung

ThresholdImage	Parameter		Rückgabe
ThresholdImage (0, 0, 100, 100, 0)	**left,** **top,** **right,** **bottom,** **thresholdmin**	**(number)** **(number)** **(number)** **(number)** **(number)**	(number)

Die Funktion arbeitet sehr schnell so das mehrmals ganze Bilder damit abgefragt werden können, dennoch empfiehlt es sich den Messbereich auf den Wirkbereich folgender Operationen zu begrenzen. Die **Generalhelligkeit** eines gesamten Bildes kann jedoch für die **Voruntersuchungen** eine wichtige **Grundlage** der Wertebestimmung und Parametrisierung sein. Die Funktion kann auch dazu dienen eigene Untersuchungsmethoden aufzubauen wie es für die **Scheitelpunkt Dunkelfeld Analyse** bei schräger Beleuchtung der Fall ist. Dies wird im Beispielprojekt **RectDetect** und im Anhang dargestellt.

10.4.0 ColorImage

In der maschinellen Bildverarbeitung werden Grauwerte verarbeitet, dies hat auch verschiedene interne Vorteile. Nachdem eine Kontur jedoch erfasst worden ist, besteht die Möglichkeit die Farbe eines Bildbereiches zu ermitteln, außerdem wird mit der Funktion Der Name der Farbe in English und in Deutsch als Klartext zurück geliefert, die Funktion Say kann diese also Phonetisch ausgeben.

Die Parameter für diese Funktion **left**,**top**,**right**,**bottom** beschreiben den zu messenden Bereich alle Pixelfarben werden in diesem Bereich addiert und durch die Anzahl der Additionen dividiert, somit liefert die Funktion den Mittelwert der **Bereichsfarbe.**

Die Rückgabe der Funktion liefert 7 Werte, der erste Wert ist ein Text der Bereichsfarbe in Englisch der zweite Wert beschreibt die Farbe in Deutsch die drei folgenden Rückgaben Erhalten die Farbintensität in der Folge: **R , G , B** gefolgt von dem Winkel der Farbscheibe.

10.4.1 ColorImage Parameterdarstellung

ColorImage	Parameter		Rückgabe

ColorImage (
0,
0,
100,
100
)
	`left,`
	`top,`
	`right,`
	`bottom,`
		`(number)`
		`(number)`
		`(number)`
		`(number)`
			(string)
			(string)
			(number)
			(number)
			(number)
			(number)

Es wird für diese Funktion das Beispiel-Projekt **ColorSay** empfohlen. Dort wird nicht nur dargestellt wie die Farbe mit der Say Funktion akustisch ausgegeben werden kann sondern auch wie mit LUA RGB Werte in Hex und umgekehrt verarbeitet werden können.

In der Textrückgabe werden 144 Farben namentlich in 2 Sprachen unterschieden.

10.5.0 Sigmalmage Schärfensensor

Die **Bildschärfe** eines rechteckigen Bereiches wird dadurch berechnet das der Helligkeitsunterschied von jedem Pixel zu jedem anderen eine Summe liefert aus der ein Rauschsignal durch statistische Operationen gewonnen wird. Das Signal aus Sigma wird in Prozent zurück geliefert. Ist der **Rauschabstand** aller Bildpunkte maximal, liefert der Sensor einen Wert >= **100%** ein einfarbiger Bereich ergibt ein Signal von **0%** .

Mit diesem **Messmittel** ist es möglich die Höhe von Objekten unter der Kamera zu messen wenn diese sich z.B. auf der **Z-Achse** entfernen oder annähern. Auf einer **Prüfvorrichtung** wie sie z.B. Mikroskope bieten, und steht's ein gleichartiges Teil gemessen wird, kann so eine Mikrometer genaue **Höhenmessung** erfolgen wie Sie kein anderes optisches Messmittel so hervorbringen kann. Dies bezieht sich auf den **Mikroskopischen** Bereich.

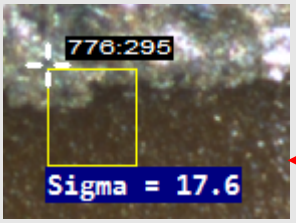
Das Verfahren der Schärfenbestimmung kommt auch beim **DeepFokusStacking** zum Einsatz. *Das Thema wird weiter unten aufgegriffen.*

Für die Parametrisierung wird eine **left**, **top** Koordinate angegeben, und von dort aus die Bereichsgröße **width**, **height**. Für kleine Bereiche ist die Rechenlast gering, steigt jedoch exponentiell an mit Vergrößerung des Messbereiches.

10.5.1 Sigmalmage Parameterdarstellung

Sigmalmage	Parameter	Rückgabe

Sigmalmage (
0,
0,
100,
100
)
	`left, (number)`
	`top, (number)`
	`width, (number)`
	`height (number)`
		(number)



Summenschärfe 17.6% (gelbes Rechteck)

10.6.0 StartDfs DeepFocusStacking

Hinter dieser Methode steht das stapeln von Eingangsbildern unterschiedlicher Schärfe. Insbesondere in der Mikroskopie kann das Verfahren zur **Objekthöhenbestimmung** und für die 3D-Visualisierung verwendet werden, es erzeugt ein tiefenscharfes Bild. Auch unter dem Namen Tiefenschärfenerweiterung in der Messtechnik seit 1972 bekannt.

Das Verfahren verlangt zur Urbilder -Erzeugung mit einer Kamera die sich auf der Z-Achse bewegt, und in einem bestimmten Intervall auf der Fahrstrecke pro (n) Mikrometer getriggert wird, z.B. über einen optischen oder Induktiven Längen – Messsensor.

Die so fotografierten Bilder werden in einem **Speicher-Stapel** übertragen. Nach einer solchen Stapelaufnahme (stacking) kann aus allen Bildern (maximal 255) ein neues tiefenscharfes **32Bit RGBA** Bild erzeugt werden in dem nur die scharfen Bildpunkte übertragen wurden. Damit wird ein Bild erzeugt das physikalisch so nicht mit einer festen Optik herstellbar ist.

Durch interne nachträgliche Anwendung des Schärfensensors werden im Alphakanal des 32Bit –Bildes für jeden Bildbereich Ebenen-Nummern hinterlegt die beschreiben aus welcher Stapelhöhe der Ausschnitt stammt. Damit ist es möglich sehr genaue Höhenvermessungen zu gewinnen. Außerdem entsteht bei dem Verfahren ein **3D-Netzobjekt** das die räumliche Geometrie des Objektes in der **Draufsicht** wiedergibt.

Der erste zu übertragende Parameter **SectorSize** ist die Sektorengröße die eine Gruppe von Bildpunkten beschreibt, dessen Schärfe die Ebene der Z-Fahrt für den Alphakanal des **32Bit RGBA** Bildes bestimmt. Dieser Wert ist z.B. **9** oder für große Bilder **24** Bildpunkte²

Der Parameter **Type** entscheidet über Qualität des Ergebnisbildes **Type** =0 basiert auf eine Mittelwert und **Type** =1 auf eine Maximalwertberechnung.

Für die Approximation der Bildbereichssegmente wird ein Iterationsfilter verwendet der Rauschspitzen unterdrückt. Der Parameter **Filter** korreliert mit der Anzahl der Bilder die aufgenommen werden sollen und sollte **15%** der Anzahl entsprechen. Dies optimiert lediglich die Darstellung im **3D-View** der unten noch beschrieben wird.

Die Parameter **Foreground=130, Brightness=110, Background=100** beschreiben die optische Erscheinung des Ergebnisbildes und sind in obiger Dimensionierung gut.

ImageCnt gibt die Anzahl der zu erwartenden Bilder an, schnell kann der gesamte Arbeitsspeicher mit zu hohen Bildanzahlen belastet werden, bis hin zum Abbruch des Verfahrens. Für eine Höhenmessung mit einem **FieldOfView** von wenigen Millimetern werden nicht mehr als 10-50 Bilder benötigt. Gute Ergebnisse liefern bereits 10 Bilder. Wird die Bildanzahl erreicht, werden keine weiteren Bilder mehr angenommen und der Prozess beginnt im Hintergrund mit den verfügbaren Prozessoren die Rechenschritte durchzuführen. Am Ende der Operationen löst die Funktion das System -Bildanzeigeprogramm aus und zeigt das Ergebnis auf dem Desktop. Die **32Bit RGBA** Ergebnisdatei heißt immer **Region.bmp** und liegt im Verzeichnis aus dem der Prüfplan gestartet wurde, jedes weitere Aufrufen der Funktion löscht dieses Bild wieder. Die Berechnungszeit ist vom Datenvolumen abhängig.

Der Parameter **Action** entscheidet wie die Funktion die Bilder beschaffen soll:

```
DFS_ACTIONRECORD = 1 --record
DFS_ACTIONRECALC = 2 --recalc
DFS_ACTIONREGEN  = 3 --stop
DFS_ACTIONMANUAL = 4 --record manual
```

Das Action-Flag **DFS_ACTIONRECORD** überträgt alle Parameter und die Funktion sammelt die Bilder im Hintergrund bis die Anzahl erreicht ist.

DFS_ACTIONRECALC berechnet den Stapel neu z.B. mit anderen Parametern für **Foreground, Brightness, Background**.

DFS_ACTIONREGEN Stoppt vorzeitig eine zuvor mit **DFS_ACTIONRECORD** gestartete Aufnahme und berechnet den Stapel mit den bis hierhin vorhanden Bildern.

DFS_ACTIONMANUAL ist nützlich für Mikroskope und freilaufende Kameras ohne Trigger-Möglichkeit. Es wird nur das gerade aktuelle Bild auf den Stapel gelegt, z.B. durch drücken eines **Buttons** im Benutzerdefinierten Parameterbaum.

10.5.1 StartDfs Parameterdarstellung

StartDfs	Parameter	Rückgabe
StartDfs (24, 0, 4, 130, 110, 100, 32, 10, DFS_ACTIONRECORD)	SectorSize, Type, Filter, Foreground, Brightness, Background, ImageCnt, Increment, Action	(nil)

Nur unten scharf

Nur oben scharf

Oben und unten scharf

Nutzen sie die Projektbeispiele zur FokusFusion um die Vorgänge nachstellen zu können.

11.0.0 Alternative Kommunikation

Neben der Prüfplankommunikation mit der Prozess -Datagramm Übertragung via **SetPatResult** ist der **LUA**-Skripthost fähig parallel Verbindungen über unterschiedliche Interfaces die LUA- Spezifisch angewendet werden können wie **TCP**, **RS232** oder **ODBC** Datenbankkommunikation Verbindungen herzustellen, oder eigene Dateioperationen auf dem **Filesystem** durchzuführen.

Der **VisionServer** bildet zwei alternative Kommunikationen an, um ohne weiterführende **LUA**- Spezial Kenntnisse für **TCP** und **RS232/432** eine Kommunikation mit anderen Geräten herzustellen. Insbesondere ist das Sinnvoll um einen Mikrocontroller **z.B.** die Schrittweite zu übermitteln in der dieser die Kamera triggern soll, wenn eine **Z-Achse** überwacht wird.

Zum bisherigen Verhalten das nur **Events** eintreffen wenn ein **Bild** getriggert wurde machen die beiden alternativen Kommunikationen einen Unterschied, weil nach der Initialisierung einer Kommunikation zu jeder Zeit Daten eintreffen können die asynchron zum Prüfplan ablaufen.

Es ist pro Prüfplan nur eine Verbindung für **TCP** und dazu eine **RS232/432** Verbindung möglich. Der Versuch noch eine weitere Verbindung zu eröffnen trennt die erste. Für komplexe Mehrfachverbindungen bietet das LUA- Interface alternative Bibliotheken an auf die hier nicht näher eingegangen wird.

Für die Kommunikation mit dem Mikrocontroller **ADuC7020** ist ein **C** -Programm im **SDK -** Bereich enthalten das auf den ARM Prozessor übertragen werden kann um mit hoher Baudrate dem Prüfplan Informationen zu übertragen oder versenden zu können. Außerdem wurde für **Linux** -Kleincomputer z.B. **Raspian** ein Beispiel in **C++** beigelegt das die protokollierte Kommunikation mit den Datagramm aufzeigt, dieses kann auch für die freie Verwendung umgestaltet werden, es wurde mit **CodeBlocks** Kompiliert.

In den Beispiel-Prüfplänen zum **DeepFocusStacking** kann eingesehen werden, auf welche Weise eine einfache Verbindung zu einem Achsen-controller hergestellt wird um die Inkrementschrittweite zu übertragen.

Nachdem eine Kommunikation aufgebaut worden ist, werden eintreffende Daten an ein zur Übertragung notwendiges **Event** übergeben, dies erhält eine Anzahl und eine Liste von Integer Werten. Handelt es sich um Text findet sich in jedem Arrayelement der **ASCI I**-Code der individuell mit einer **LUA**- Stringoperation zu einer Zeichenkette gewandelt werden kann. Dies erfordert natürlich eine Absprache mit dem Datensender.

11.0.1 ComWrite RS232/432

Die Funktion **ComWrite** stellt eine Verbindung her oder sendet Daten je nach Anzahl der übergebenen Parameter. Das Protokoll ist immer auf **8 Data bits None Parity bit 1Stop bit** Ausgerichtet, also eine **8n1** Kommunikation.

Für den Fall der Verbindungsherstellung ist der erste Parameter **port** der numerische Wert des Com -Port's der angesprochen werden soll, der zweite Parameter **baud** ist die Baudrate die frei wählbar ist. Es ist empfehlenswert zuerst mit einem Terminalprogramm die Verbindungen zu testen und danach die Verbindung auch wieder zu trennen.

Beispiel : `ComWrite(2,115200)  (verbindet den Port COM2)`

Zum Senden von Daten kann **ComWrite** ein Text übergeben werden indem nur ein Parameter der Funktion verwendet wird.

Beispiel: `ComWrite("Hallo Welt") (sendet einen Text)`

Der Rückgabe Wert ist **0** für Fehler und **1** für Erfolg. Um Details der Verbindung im Fehlerfall zu sehen, soll der **DebugView.exe** geöffnet sein.

11.0.2 ComWrite Parameterdarstellung

ComWrite	Parameter	Rückgabe
ComWrite(2, 19200)	**port,** **(Text oder number)** **[baud]** **(baudrate** number)	(number)

11.0.3 OnCom Event

Das Event liefert Daten des verbunden Klienten, **cnt** enthält die Anzahl der Listeneinträge, **tab** enthält die Einträge für jedes empfangende Zeichen.

OnCom	Parameter	Rückgabe
OnCom(2,[])	**cnt,** (number) tab (list)	(number)

In den Beispielen zur FokusFusion kann die Verwendung der Kommunikation eingesehen werden. Ein Zeichen aus der Liste **tab** kann wie folgt einen String verlängern:

```
local msg = ""
for i=1, cnt, 1 do
   msg = msg..string.char(tab[i])
end
```

11.1.0 TcpWrite TCP

Die Funktion **TcpWrite** stellt eine Verbindung her oder sendet Daten je nach Anzahl der übergebenen Parameter. Das Protokoll ist immer TCP und stellt eine Verbindung über die Ethernet -Netzwerkkarte her

Für den Fall der Verbindungsherstellung ist der erste Parameter **ip** der Text einer IP Adresse die angesprochen werden soll, der zweite Parameter **port** ist der Port auf dem ein Server verbunden werden soll. Es ist empfehlenswert zuerst mit einem Terminalprogramm die Verbindungen zu testen und danach die Verbindung auch wieder zu trennen.

Beispiel : `TcpWrite("192.168.2.31",777)  (verbindet den Port 777)`

Zum Senden von Daten kann **TcpWrite** ein Text übergeben werden indem nur ein Parameter der Funktion verwendet wird.

Beispiel: `TcpWrite("Hallo Welt") (sendet einen Text)`

Der Rückgabe Wert ist **0** für Fehler und **1** für Erfolg. Um Details der Verbindung im Fehlerfall zu sehen, soll der **DebugView.exe** geöffnet sein.

11.1.0 TcpWrite Parameterdarstellung

TcpWrite	Parameter	Rückgabe
TcpWrite(**"192.168.2.31"**, 777)	**ip,** **(string)** **[port]** **(portnummer)** (number)	(number)

11.1.2 OnTcp Event

Das **Event** liefert Daten des verbunden Servers, **cnt** enthält die Anzahl der Listeneinträge, **tab** enthält die Einträge für jedes empfangende Zeichen.

OnTcp	Parameter	Rückgabe
OnTcp(2,[])	**cnt,** (number) tab (list)	(number)

In den Beispielen zur FokusFusion kann die Verwendung der Kommunikation eingesehen werden. Ein Zeichen aus der Liste **tab** kann wie folgt einen String verlängern:

```
local msg = ""
for i=1, cnt, 1 do
   msg = msg..string.char(tab[i])
end
```

12.0.0 Process-Logbuch

Die Data –Log Textdatei erhält für jeden einkommenden Bildtrigger die Messergebnisse die mit **SetPatResult** und **SetPatStatus** übertragen wurden, linear Zeilenweise durch Komma getrennt. Der Dateiname einer Logbuchdatei ist der Name des Prüfplans mit der Endung ***.txt**

Die Datei wird In dem Verzeichnis erzeugt aus dem der Prüfplan stammt, oder aus dem Verzeichnis das der Benutzer/Operator explizit auswählte.

Im Zielverzeichnis wird ein Logbuch Wurzel-Verzeichnis angelegt das folgendes Format hat:

Log-12-17-08-41 (M:T:h:m)

Wird das Programm beendet, und neu gestartet, wird ein neues Wurzel- Verzeichnis angelegt mit einem aktualisierten Zeitstempel.

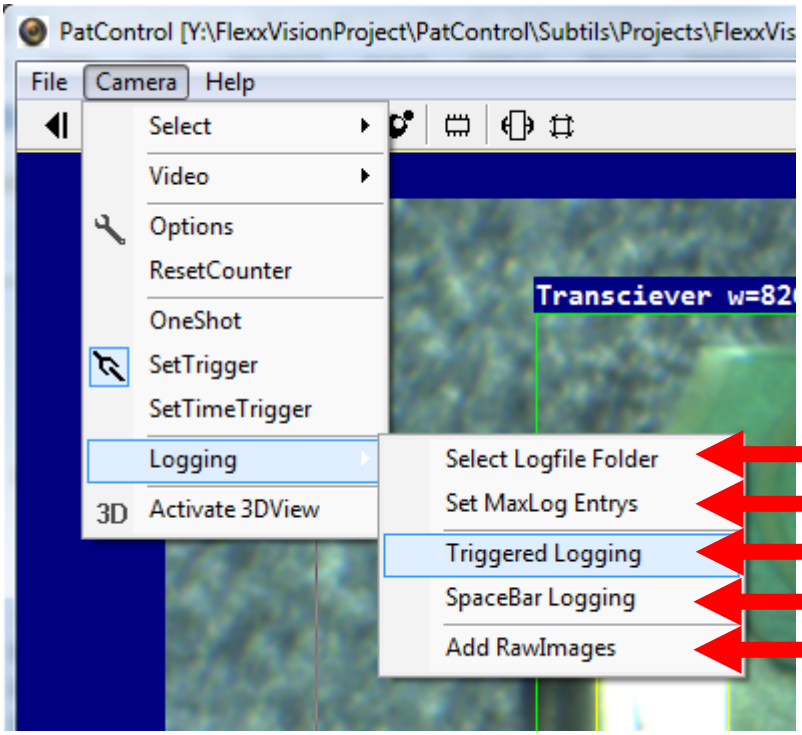
Jede Zeile eines Logbucheintrages enthält einen Zeitstempel, die Nummer des Logbuch-Eintrags, dem Bildtrigger, und der Prüfplannummer gefolgt vom Prozessstatus und den anschließenden Merkmalswerten.

12.0.1 Beschreibung einer Logbuchzeile mit 11 Merkmalen:

```
08.41.13,    (Zeitstempel h:m:s)
870,         (Millisekunde)
00000020,    (Logbuch Id)
00004440,    (Trigger)
001,         (Prüfplannummer)
000000,      (Prozessstatus 0=Gültig sonst Fehlercode)
005306,      (Merkmalswert 001)
007510,      (Merkmalswert 002)
007347,      (Merkmalswert 003)
008341,      (Merkmalswert 004)
004978,      (Merkmalswert 005)
004365,      (Merkmalswert 006)
007790,      (Merkmalswert 007)
008999,      (Merkmalswert 008)
008376,      (Merkmalswert 009)
004703,      (Merkmalswert 010)
000820       (Merkmalswert 011)
```

Zu jeder **Logbuchzeile** wird grundsätzlich das dazugehörige **Messbild** gespeichert, das als ***.jpg** grafische Mess-Informationen enthält. Die Bilder werden in zwei Unterverzeichnissen namens **Good** und **Bad** einsortiert, entsprechend dem **Prozessstatus 0** für Gut und **>0** für schlecht. Somit ist gewährleistet jedes Merkmal grafisch zu Referenzieren.

12.0.2 Logbuch Menü:



Ziel Verzeichnisauswahl

Rotationslogbuch Aktivierung

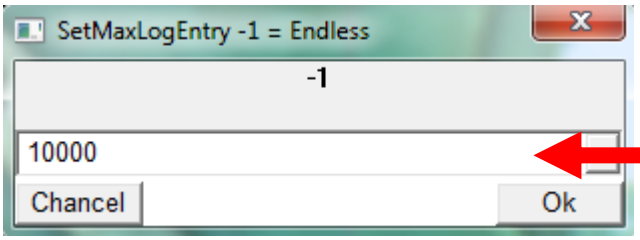
Aufzeichnung über Trigger

Aufzeichnung über Leertaste

Ur-Bilder Speichern aktivieren

Über den Menüpunkt **Select Logfile Folder** kann ein alternativer Speicherort für die Logbuchführung ausgewählt werden, dies sollte vor dem Start der Aufzeichnung erfolgen.

Der Menüpunkt **Set MaxLog Entrys** erlaubt eine rotatorische -Aufzeichnung, wenn der Logbuchzähler den dort vorgegeben Wert erreicht, beginnt das Logbuch wieder bei Eintrag **0**, dies ermöglicht es zu verhindern, dass die Aufzeichnungen die Grenzen des Speicherplatzes auf dem Datenträger überschreiten. Ein Wert von **-1** (Default) speichert bis zum Datenüberlauf.



Anzahl bis Logbuch bei Eintrag 0 beginnt

Der Menüpunkt **Triggered Logging** Startet die Aufzeichnung für jeden Bildtrigger automatisch, während **SpaceBar Logging** nur einen Logbucheintrag auslöst wenn die Leertaste gedrückt wird.

Soll zu jedem Eintrag das **Ur-Bild** gespeichert werden, muss der Menüpunkt **Add RawImages** aktiv sein. Das **Raw –Bild** aufzeichnen ermöglicht den gesamten Prozess der Bildverarbeitung zu einem späteren Zeitpunkt über das Abspielen des Verzeichnisses als virtuelle Kamera erneut zu wiederholen. **Diese Möglichkeit ist ausschlaggebend für die Prozessoptimierung.**

12.0.3 Logbuch Eigenschaft merken

Nach dem Speichern des Prüfplanes werden die Daten die im Menü **Logging** gesetzt wurden in der Prüfplan –Datei ***.set** unter dem Schlüssel **[UserData]** gespeichert.

Wenn der Prüfplan später erneut geladen wird, wird mit der Logbuchaufzeichnung in dem Verzeichnis das ausgewählt wurde und der Rotationseigenschaft automatisch fortgefahren, wenn der Schalter **Triggered Logging** oder **SpaceBar Logging** aktiv war.

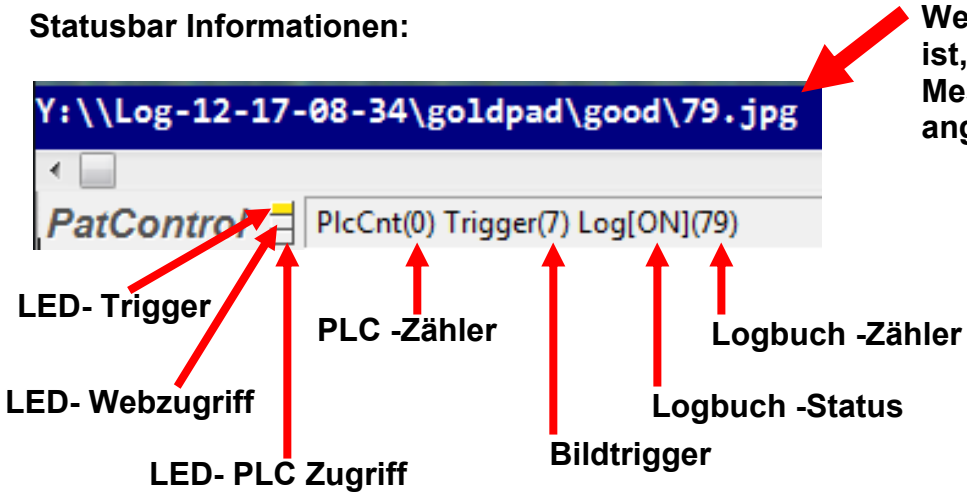
Auszug aus der Datei *.set:

```
[UserData]
Set(1)=1          (Logbuch -Aufzeichnung ist aktiv)
Set(3)=10000      (Logbuch Rotationswert)
Set(4)=c:\\Temp   (Logbuch Ziel -Verzeichnis)
```

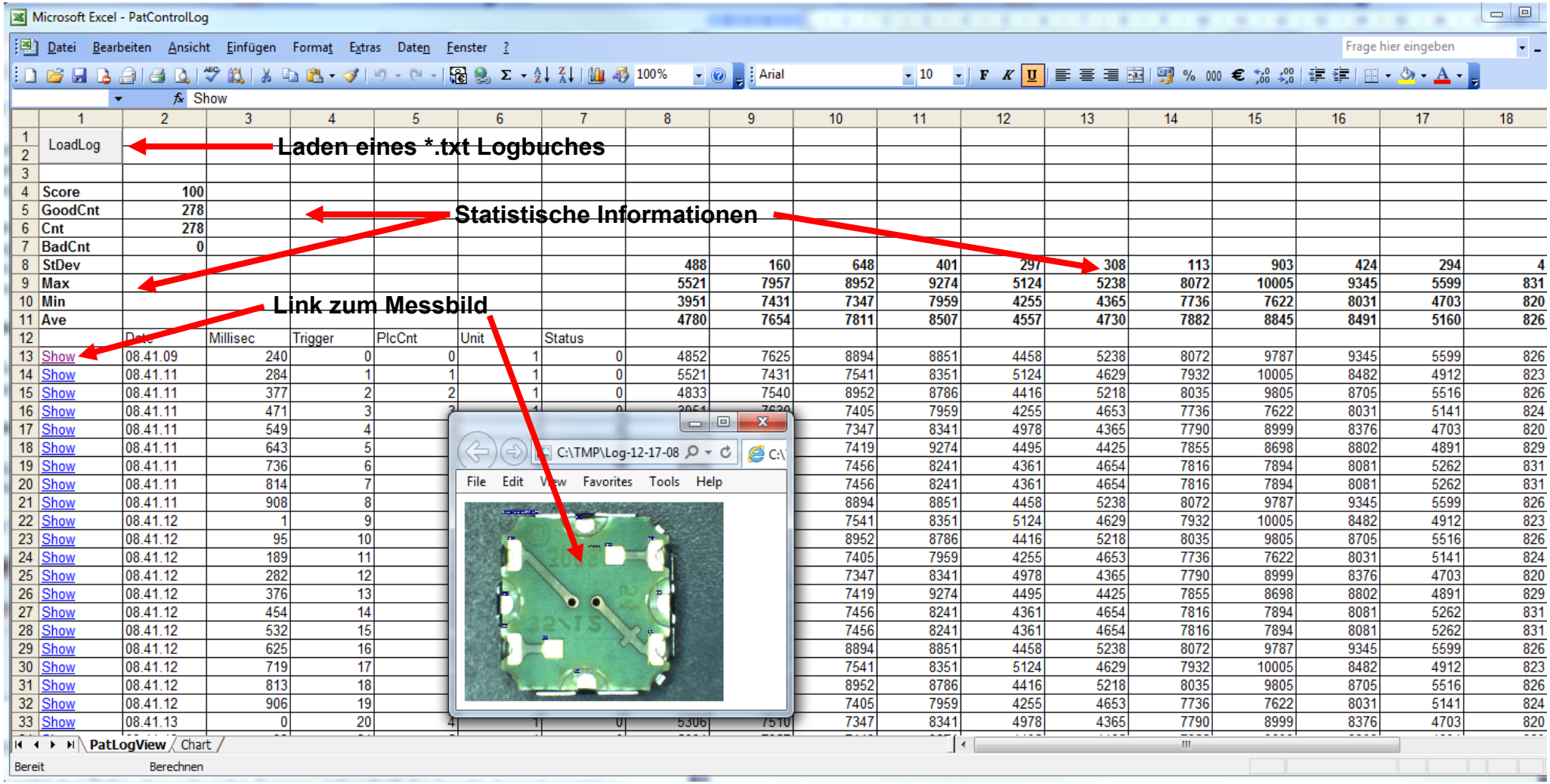
12.0.4 Logbuch Status:

In der **Statusbar** von **PatControl** wird über den Fortgang der Logbuchaufzeichnung informiert, dazu wird die Anzahl der **PLC**(SPS) Übertragungen angezeigt, der aktuelle Bildtrigger, und ein Signal **[ON][OFF]** das signalisiert das die Logbuch-Aufzeichnung aktiv ist. Dem folgt der Logbuch –Zähler, der für jede neue Zeile ansteigt.

Statusbar Informationen:

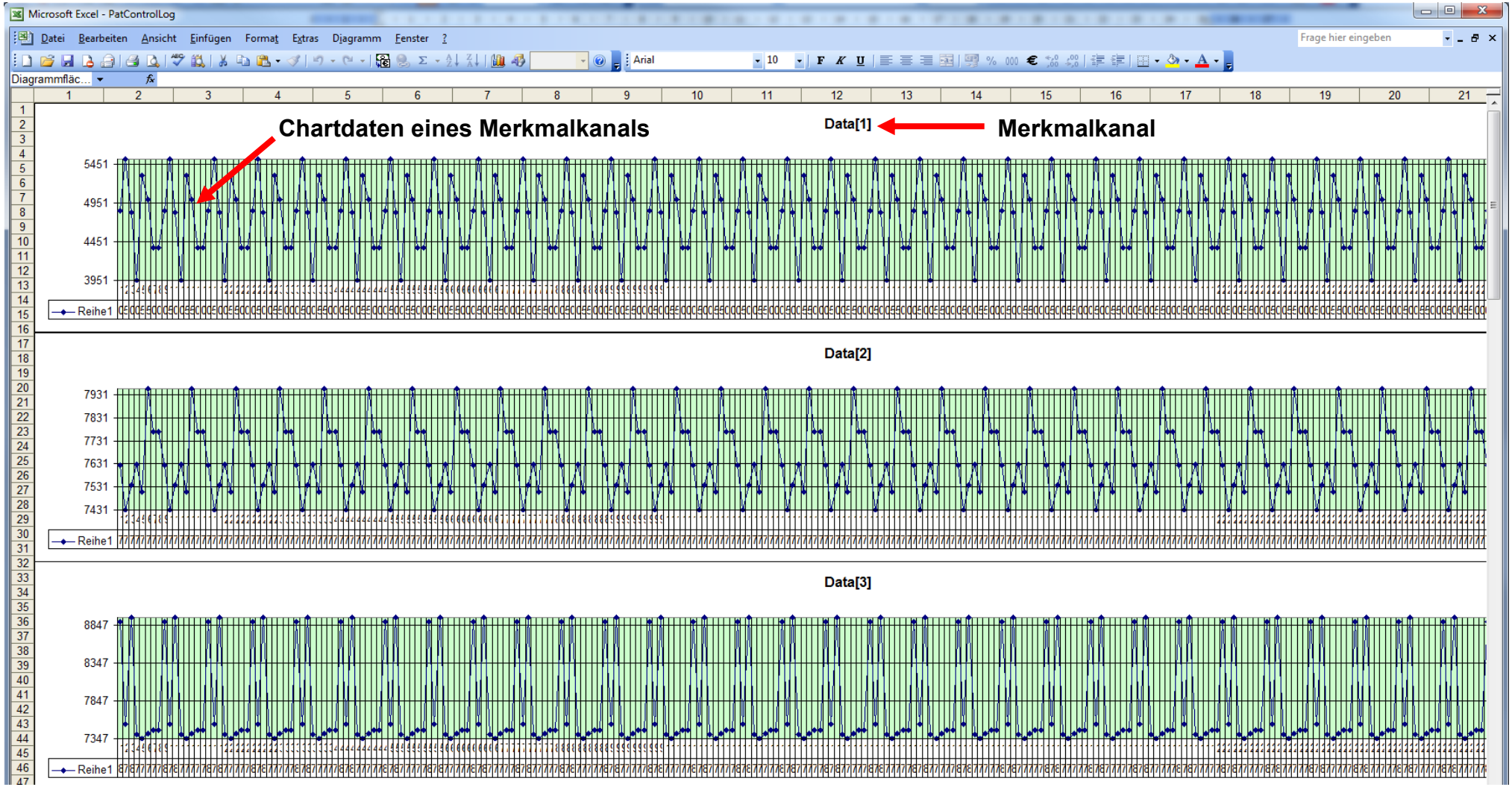


12.0.5 Logbuch Excel Plot-Darstellung:



Im **SDK –Bereich** befindet sich ein Excel –Beispiel **(PatControlLog.xls)** mit dem das Prozesslogbuch visualisiert werden kann.

12.0.6 Logbuch Excel Chart-Darstellung:



Jeder Merkmalskanal wird mit seinen Werten zur Demonstration angezeigt, das **Excel -Skript** kann frei editiert und ergänzt werden.

13.0.0 Rotationsprüfpläne

Die **Prozessleitstelle** kann vor der Bildtrigger -Auslösung eine **Prüplannummer** mit dem **Prozessdatagramm** im Datenfeld **Measure** übertragen. Ist dieser Wert **>0** wird aus dem beim Programmstart geladenen Prüfplan ***.set** einer der Nummer entsprechender alternativer Prüfplan geladen. Damit dies möglich ist wird eine ***.set** Datei erstellt die lediglich die Namen aller „normalen“ Prüfpläne enthält sowie die gemeinsamen Verbindungsinformationen zur Prozessleitstelle. Diese Form der zuerst geladenen Prüfplandatei ***.set** mit den Informationen weiterer gewöhnlicher Prüfpläne unterscheidet sich inhaltlich insofern, dass sie weniger Daten enthält. Es entfällt die Information aller Patterns und Benutzerdaten, denn diese liegen in den Prüfplänen vor die über die Prüfplannummer nachgeladen werden. Der Ladevorgang eines Rotationsprüfplanes dauert ca. 10**[ms]**.

13.0.1 Beispiel eines Rotationsprüfplan :

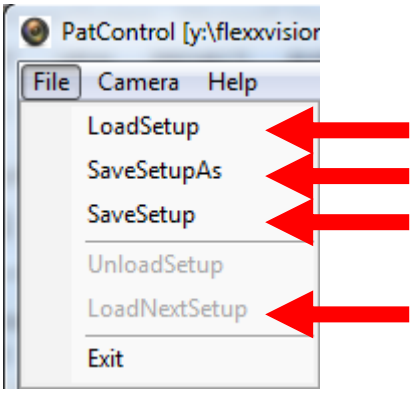
```
[FrameInfo]
SpsChan=201                 (gemeine Verbindungsinformation)
SpsIp=192.168.2.154         (gemeine Verbindungsinformation)
SpsSlot=2                   (gemeine Verbindungsinformation)
Webport=8081                (gemeine Verbindungsinformation)
[SetupStack]                (Schlüssel für die Prüfplanliste)
Setup(0)=TestPlanA.set      (erster  Plan für Nummer 1)
Setup(1)=TestPlanB.set      (zweiter Plan für Nummer 2)
Setup(2)=TestPlanC.set      (dritter Plan für Nummer 3)
```

Rotationsprüfpläne haben gemeinsam, dass sie dieselbe Verbindung zur Prozessleitstelle besitzen so wie sonst die einzelnen Prüfpläne. Die dortigen Verbindungsinformationen sind dann ungültig und werden nicht verwendet wenn sie über die Rotation ausgelöst werden.

Es gilt für alle kommunikativen Prüfpläne, dass eine Verbindung zur Kommunikation nur dann hergestellt wird wenn der Prüfplan über seine **Verknüpfung** gestartet wird, oder wenn **PatControl.exe** als Parameter den zu ladenden Prüfplan erhält.

Der Sinn eines Rotationsprüfplanes ist z.B. wenn eine Kamera auf einer **Achse** von der Prozessleitstelle bewegt wird, und nun ein gänzlich anderes Bild liefert als an **Position A** Dann wird auch ein anderes **Prüfverfahren** nötig.

Menü LoadNextSetup :



Prüfplan laden, es wird keine Verbindung aufgebaut

Prüfplan unter anderen Namen speichern

Prüfplan unter selben Namen speichern

Vorhandene Rotationsprüfpläne können im Menu manuell durchgeschaltet werden

13.0.2 Alternatives Rotations –LUA Skript

Im Fall von Rotationsprüfplänen kann es sinnvoll sein, das selbe **LUA-Skript** für alle beteiligten Prüfpläne zu verwenden, lediglich die Parameterbäume für Pattern und Benutzer definierte Parameter werden bei einem **Prüfplanwechsel** immer mit dem Dateinamen des zugehörigen Prüfplanes geladen.

Der Benutzerdefinierte Parameterbaum unterstützt ein Datenfeld das vom **Prüfplan –Parser** zum laden eines anderen Prüfplanes herangezogen wird, anstatt den Plan zu laden der den selben Namen trägt wie der Prüfplan mit der Endung ***.lua**

Um das zu erreichen muss im Benutzerdefinierten Parameterbaum ein Datenfeld angelegt werden das folgendes Format aufweist:

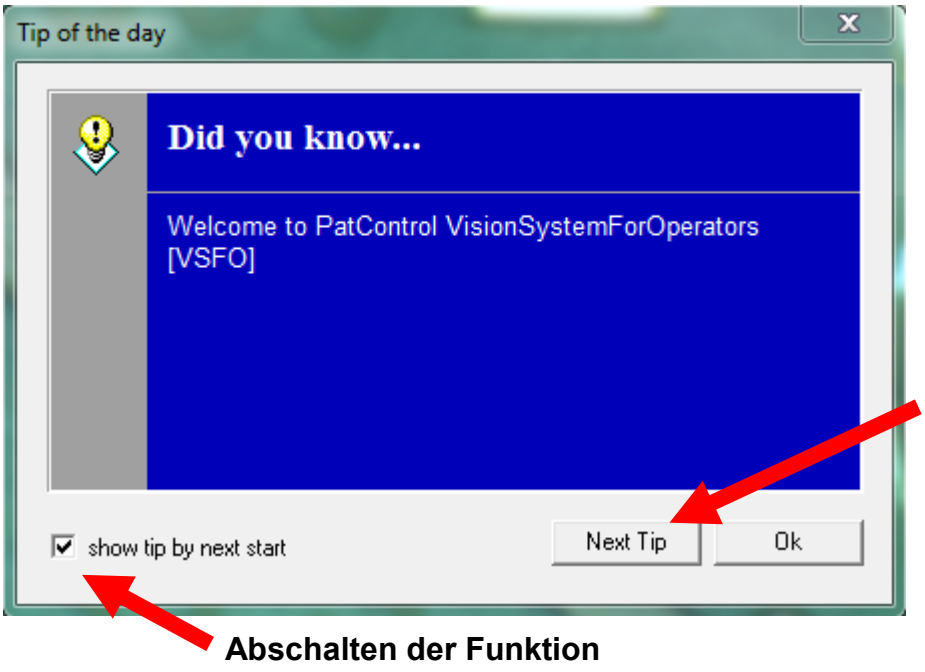
```
<Root2
Format=text
Name=LuaSkript
Value=0
type=colapse
InfoText="Property for Lua"
>

<Edit
Format=text
Name=SkriptPath
Value = y:\FlexxVision\PatControl\TestPlan.lua
Type=expand
InfoText="Alternate LuaSkript"
/>

</Root2>
```

Im Parameterbaum sieht das dann wie folgt aus:

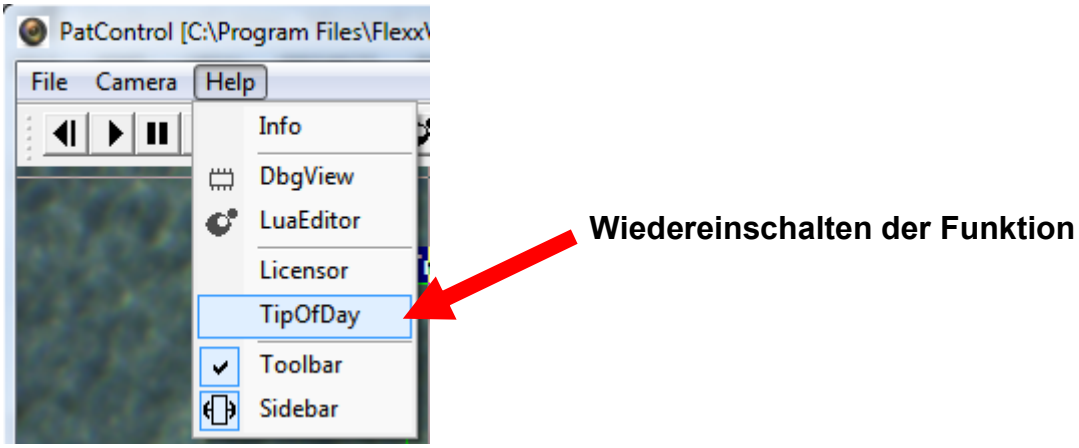
Der Schlüssel **<Root2 bis </Root2>** ist optional und dient der verbesserten Darstellung.

13.0.3 Rotatorische Programm Infos

Möchte man dem Operator/Benutzer der Software zusätzlich Informative Tipps anzeigen die bei einem Programmstart angezeigt werden kann die Textdatei **Tips.tip** im Hauptverzeichnis der Installation mit eigene Einträgen und Hinweisen versehen werden. Die Texte rotieren durch das Anzeigefeld bei jedem Klick auf **Next Tip**.



Sollte die Funktion durch die Checkbox deaktiviert sein, kann diese über das Menü wieder Hervorgebracht werden.



14.0.0 3D-Visualisierung (3D-View)

PatControl enthält eine integrierte 3D-Engine die mit **OpenGL** arbeitet. Die Visualisierung von Messbildern wird grundsätzlich in zwei verschiedenen Betriebsarten durchgeführt.

Modus A) Helligkeit (Histographische –Darstellung).
Modus B) Tiefenpixel (Höhen –Darstellung).

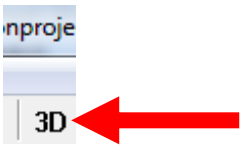
Für beide Verfahren wird für jeden Bildpunkt ein Netzknoten erzeugt, jeder Netzknoten (**Vertice**) verfügt über weiterführende Informationen:

```
1) Lichtabstrahlvektor –Normalvektor [x,y,z]
2) Textur –Koordinate [u,v,n]
3) Farbe  [r,g,b]
4) Position [x,y,z] (wobei Z die Netzhöhe angibt)
```

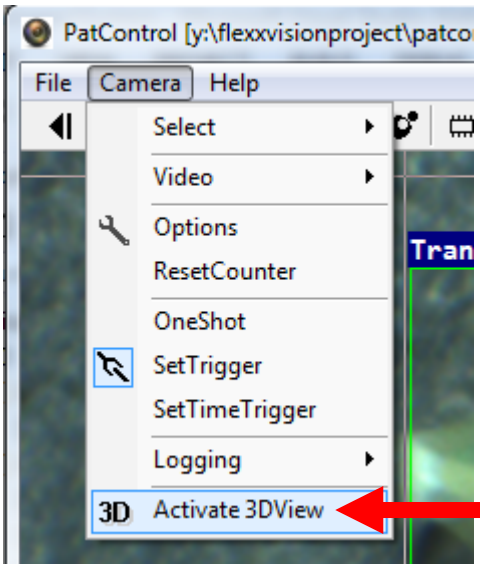
Die Datenmengen die für ein Bild benötigt werden sind beträchtlich, denn jeder Netzknotenpunkt erhält aus obiger Tabelle insgesamt **4 x 3** Werte also ist das Datenaufkommen für ein Bild bis zu **12**-mal größer.

Dennoch wird es durch interne 3D-Treiberoperationen ermöglicht, das ein Bild in Quasi – Echtzeit zur Darstellung kommen kann. Diese Darstellung bremst den gesamten Prozess der Bilderkennung, die im Hintergrund weiterläuft wenn der **3D-View** aktiviert wurde, und eine Echtzeit -Darstellung des Ur-Bildes aktiviert ist. Eine leistungsstarke Grafikkarte wird empfohlen und entschärft das Problem.

14.0.1 3D-View aktivieren



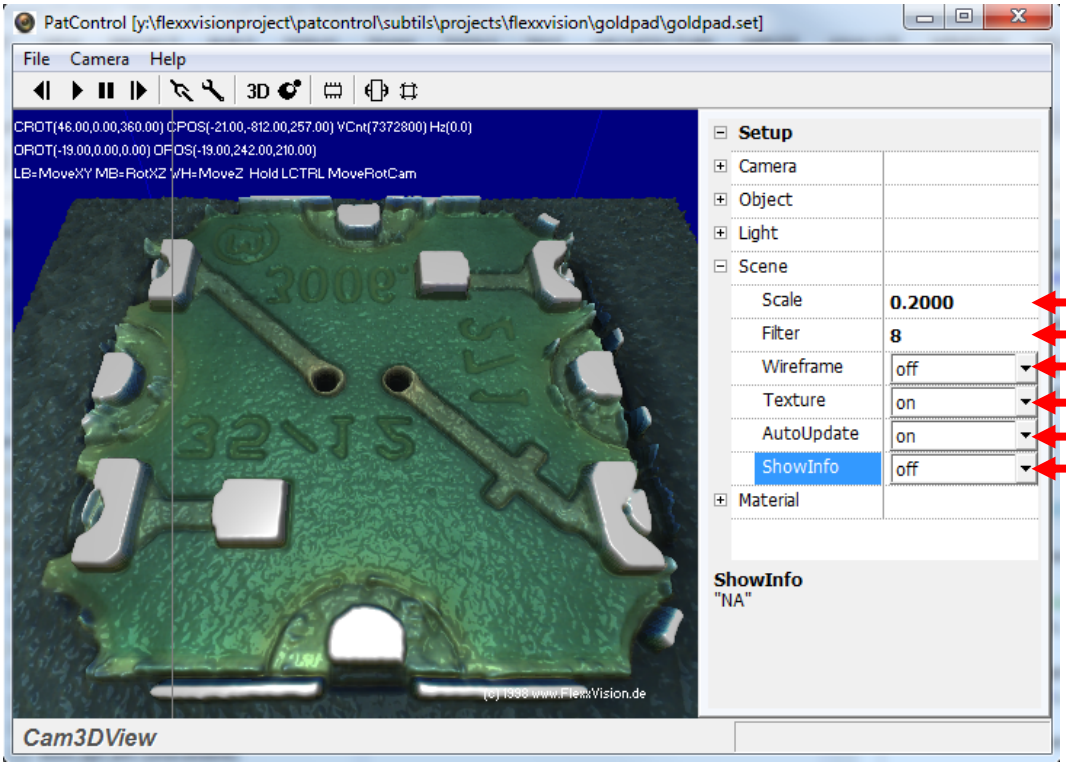
3D Visualisierung über die Toolbar aktivieren



3D Visualisierung aktivieren

14.0.2 3D-View Textur (Helligkeit Darstellung)

Modus A)



In der Darstellung Helligkeitshistogramm kann der Schalter **AutoUpdate** aktiviert werden um jedes neue einkommende Bild in den **3D-View** einzurechnen. Da es durch den Rauschanteil des Bildes zu einer Streuung der Helligkeitswerte kommt, ist es erforderlich einen **Iterationsfilter** auf die Netzpunkte anzuwenden, dieser gleicht die Höhen statistisch gemittelt sehr gut aus, ein Durchlauf von 3 – 10 Iterationen beseitigt die **Nadelspitzen** größere Werte sind sehr rechenlastig da jeder Netzknoten mit seinen Nachbarn in Korrelation gestellt wird.

Das Eingabefeld **Skalierung** gibt den Betrag an mit dem ein Netzknoten multipliziert wird, damit ist die Ausschlags Höhe der Darstellung veränderlich ohne den Messwert der Helligkeit an einem Netzknoten zu verfälschen.

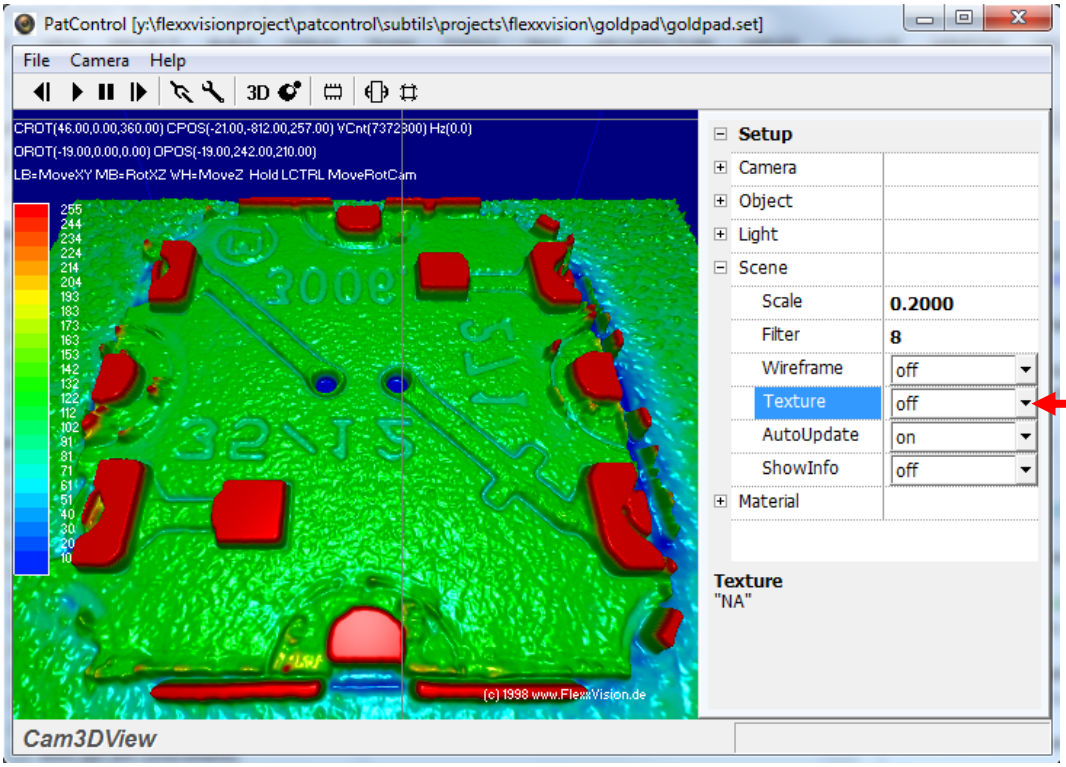
Der Schalter **ShowInfo** blendet ein Koordinaten -Netz ein, um sich in der Ansicht besser orientieren zu können.

Der Schalter **Texture** beschleunigt zum einen die Darstellung da keine Bitmapinformationen übertragen werden, zum anderen wird in den **JetColor** –Farbraum umgeschaltet wo jede Höhe eine Farbe erhält. Unten blau bis Oben rot.

Die verborgenen Schalter sind **OpenGL** Spezifische Variablen die auf das gesamte Erscheinungsbild wirken, diese beschreiben das Material und die Lichtreflexions-Eigenschaften.

14.0.3 3D-View JetColorRoom (Helligkeit Darstellung)

Modus A)



Nach dem umschalten in den **JetColor** –Farbraum werden Netzknoten in der Ausschlags Höhe mit einer Farbe belegt. In der Helligkeit- histographischen Darstellung wird jeder Höhe eine Farbe zugeteilt. Der **Kreuz -Cursor** im Messbild zeigt die augenblickliche Helligkeit an seiner Position mit einem beweglichen **Marker** an. Der Wert kann in der Helligkeit einen Bereich von **0 bis 255** annehmen.

Mit dem **Kreuz –Cursor** kann das Bild „gegriffen“ werden, dazu muss die linke Maustaste gehalten werden, eine Bewegung der Maus richtet die Ansicht entsprechend neu aus.
Im **Modus A)** Zeigt der Cursor die Helligkeit der Position und nicht die Höhe an.

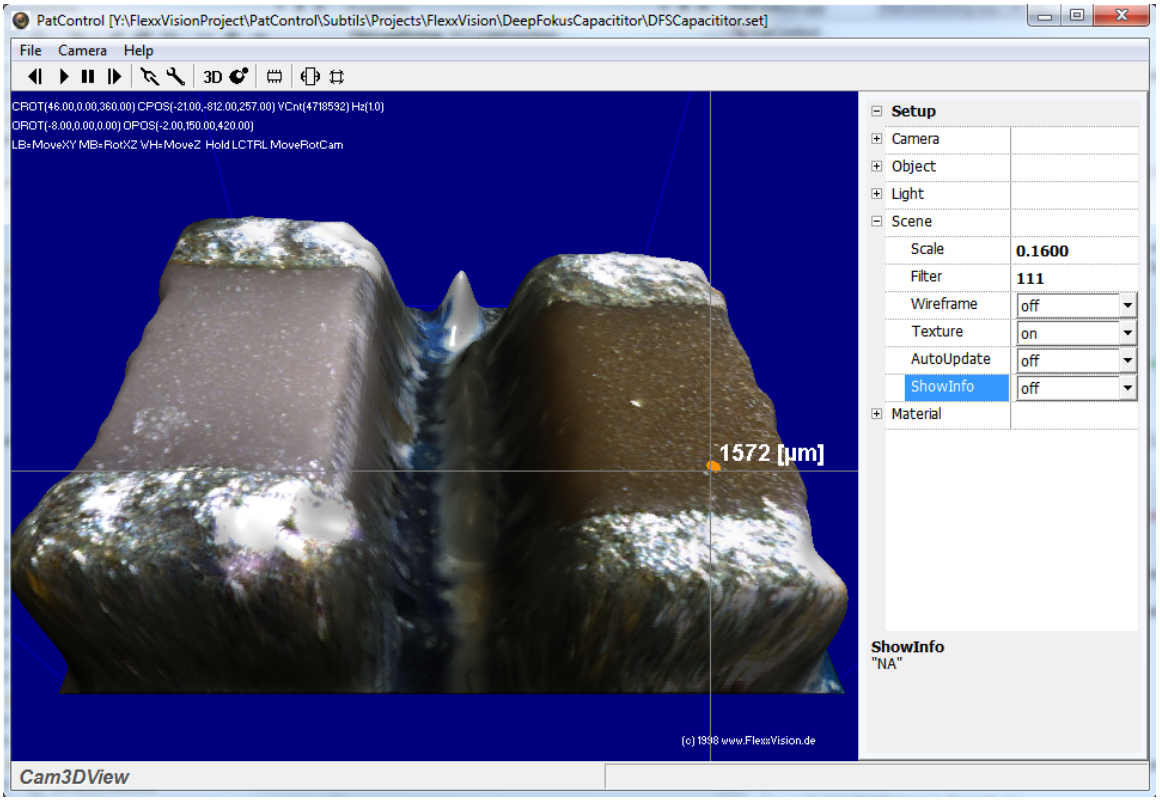
Wird stattdessen die mittlere Maustaste gehalten, wird das Bild über die **Achsen** gekippt.

Dies betrifft die **Objektrotation/Position**, es gibt in der **3D-Ansicht** auch noch die **Weltrotation/Position,** um diese zu bewegen oder zu kippen wird zusätzlich die linke **STRG** Taste gedrückt gehalten.

Die Dreh und Positionskoordinaten für die Welt und Objektansicht werden oben links dargestellt, damit ist es möglich eine genaue Positionierung von Objekt und Welt-Ansicht zu ermöglichen, die numerischen Felder für diese Daten ändern sich synchron im **Parameterbaum** unter der Rubrik **Camera** und **Object**.

14.0.4 3D-View Textur (Höhen Darstellung)

Modus B)



Die Betriebsart **Modus A)** (Helligkeit) unterscheidet sich zur Betriebsart **Modus B)** (Höhenmessung) dadurch, dass jeder Netzknoten die Höhe in **Mikrometer** repräsentiert Und nicht der **Pixelhelligkeit** (0-255).

Dieses Verhalten wird vom **Eingangsbild** gesteuert, ein **32Bit RGBA** Bild das im Alphakanal die Höhe der Ebenen enthält löst die Darstellung in Mikrometer aus.

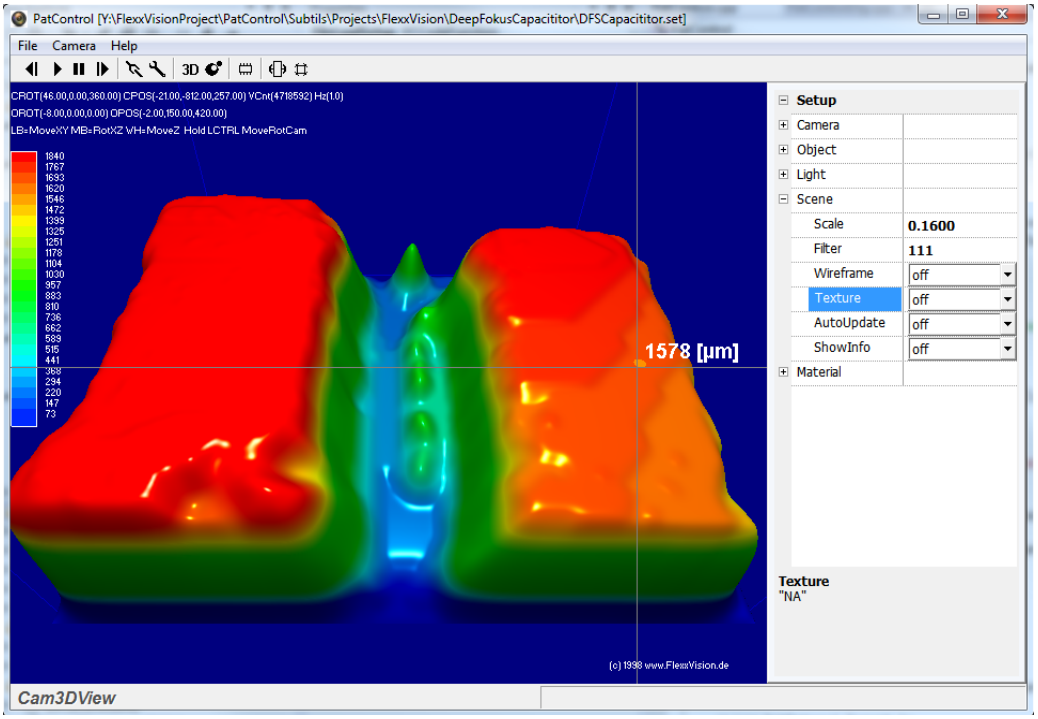
Die Bilder mit Tiefenpixelinformation kommen von der bereits beschrieben DeepFokusStacking- Funktion **StartDfs** die über die Lua-Prüfpläne **FocusFusion** erzeugt worden sind.

Dort wird die berechnete Höhe im Ergebnisbild hinterlegt. Der Schalter „**AutoUpdate**“ wird in dieser Darstellung **ausgeschaltet**, damit nicht das aktuelle **Tiefenpixelbild** mit einer niederwertigen **8Bit** oder **24Bit** Grafik zu überschrieben wird.

Der **Filter** sowie sie Skalierung(**Scale**) wirken auf die optische Darstellung, nicht auf die berechneten Höhenwerte. Die originalen Informationen werden gezeigt wenn **Scale**=1 und **Filter** = 0 ist. Beide Werte können zur Darstellungsverbesserung verändert werden.

14.0.5 3D-View JetColorRoom (Höhen Darstellung)

Modus B)



In der **Höhen -Darstellung** wird jeder Netzknoten, die Messhöhe aus dem Alphakanal multipliziert mit der Schrittweite die für jede Ebene hinterlegt ist, in Mikrometer abbilden.

Im linken Bildbereich wird in dieser Darstellung eine farbliche Legende zum Wertebereich angezeigt, um einen besseren Eindruck der **maximalen** und **minimalen** Messwerte zu erhalten.

14.0.6 3D-View Parameterbeschreibung

Jedem **Prüfplan** wird ermöglicht die Messbilder in der **3d-Ansicht** anzeigen zu lassen, auf dieser Darstellung basieren einige Parameter die direkt auf die 3D-Engine in **OpenGL** wirken, die wichtigsten Parameter wurden bereits oben dargestellt.

Jedem Prüfplannamen wird das Schlüsselwort "**3dView**" mit der Dateierweiterung ***.xml** angehangen um einen eindeutigen Bezeichner für die Parmeterdatei zu erzeugen. Die Datei kann in einem Texteditor bearbeitet werden, und verhält sich wie der Benutzerdefinierte – Parameterbaum , mit dem Unterschied das keine neuen Variablen hinzufügt werden können.

[Camera]
Hier kann die Weltansicht also die Betrachtungskamera eingestellt werden, diese hat eine Ordinate im Raum die durch die Gruppen -Parameter verändert werden kann.

[Object]
Neben der Weltansicht gibt es die Objektausrichtung im Raum, dies entspricht dem Messbild. Die Objekt Koordinaten wirken wie die Weltkoordinaten jedoch auf das Lokale Bildobjekt.

[Light]
Die Lichtquelle hat wie die Camera und die Objekt -Eigenschaft, eine Position im Raum und eine Ziel Ordinate für die Spot -Licht Darstellung. Außerdem kann hier ein Öffnungswinkel **Hotspot** und ein Leistungsabfall **Falloff** angegeben werden, die **Intensity** beschreibt die Abstrahlleistung diese kann durch die Höhe Z besser reguliert werden. Außerdem ist es möglich die Farbe der Lichtquelle anzugeben und die Spot- Eigenschaft an, oder aus zu schalten. *(LightID erzeugt weitere Lichtquellen die aber nicht gespeichert werden)*

[Scene]
Enthält die wichtigsten Parameter für die Darstellung die oben bereits erklärt wurden. **Scale** ermöglicht die Höhe nachträglich mit diesem Faktor zu multiplizieren. **Filter** ist ein Iterations Vorgang der Messspitzen unterdrückt, und viel Rechenzeit verbraucht je nach Anzahl der Wertigkeit. **Wireframe** zeigt das Modell in einer Drahtgitterdarstellung. **Texture** schaltet zwischen „JetColorFarbRaum" und Textur –Mapping um. **AutoUpdate** erneuert die gesamte Darstellung wenn ein neues Kamera –Bild eintrifft. Gerade in der Höhenmessung wo es nur ein **32Bit RGBA** Bild gibt sollte der Schalter **aus** sein, um das Ergebnis nicht augenblicklich durch eine Helligkeitsdarstellung mit einem eintreffenden neuen Bild zu revidieren. **ShowInfo** stellt zur Orientierung ein Koordinatennetz an den Rändern der Szene dar.

[Material]
Jede Oberfläche hat einige Reflexionseigenschaften, die mit **Ambient**,**Specular** und **Diffuse** bezeichnet werden. Diese Farbreflexionswerte können geändert werden. Der Wert für **Shine** beschreibt die Reflexionsstärke und sollte immer **127** für maximale Reflexion enthalten. **Shinest** und **Transparency** haben in dieser Version keine Wirkung. **Emit** dient zur Emission -Unterdrückung und wirkt mit **Shine** zusammen. Die Werte ergeben gute Lichteindrücke wenn **Emit** kleiner als **0** und größer **-1** ist, und **shine** auf **127** steht.

15.0.0 Systemschematische Darstellung

Über die Schematisch aufgeführten Komponenten ist ein Überblick der beteiligten Bestandteile der Software möglich. Nicht alle Einzelteile sind in der Dokumentation aufgeführt. In Zukunft wird sich die Software erweitern, erst nach ausführlichen Tests werden neue Verfahren veröffentlicht und dokumentiert.