Big Brother is watching it

Industrie 4.0 und das Internet of Things

Marcel Bagemihl, Simon Dörner, Steffen Krieg

Industrie 4.0 erzielt die ständige virtuelle Abbildung von Produktionsprozessen durch moderne Sensorik. Industriekameras stellen einen nahezu unverzichtbaren Bestandteil dieser sensorischen Überwachung der Prozesskette dar. Der Nachteil etablierter Kamerasysteme ist jedoch, dass sie meist mit erheblichen Kosten verbunden sind. Der folgende Artikel skizziert anhand eines praktisch umgesetzten Anwendungsfalls die Implementierung eines kostengünstigen Systems mit aktueller embedded Technologie für die eigene IoT-Landschaft.

Das Internet der Dinge (Internet of Things – IoT) und Industrie 4.0 sind aktuell in aller Munde. Doch was hat es mit diesen Schlagwörtern auf sich?

IoT beschreibt zumeist eine Ansammlung untereinander kommunizierender Devices, die Informationen zur Verfügung stellen. In diesem Zusammenhang spricht man auch gerne von sogenannten (mehr oder weniger) „smarten Devices“, je nachdem wie weit die gesammelten Daten vorverarbeitet oder gefiltert werden. Die so gewonnenen Informationen werden sowohl zentral als auch dezentral gesammelt und verarbeitet.

Die Industrie 4.0 beschreibt die 4. industrielle Revolution und die voranschreitende Automatisierung von Maschinen, die miteinander kommunizieren. Es geht darum, möglichst wenige Arbeitsschritte manuell auszuführen und den Gesamtprozess durch moderne Sensorik jederzeit überblicken zu können. Für diesen Zweck werden häufig hochspezialisierte, intelligente Kamerasysteme – sogenannte Industriekameras – eingesetzt. Diese Kameras dienen der Überwachung und Prüfung des Produktionsprozesses.

Griff in die Kiste

In der Überwachung ist ein wichtiger Anwendungsfall die Lokalisierung von Objekten, um diese automatisiert weiterverarbeiten zu können. Ein Beispiel für diese Verarbeitung ist der sogenannte „Griff in die Kiste“. Dabei werden die Positionen von bekannten Objekten in einem definierten Bereich ermittelt, damit beispielsweise ein Industrieroboter ein Objekt greifen kann. Hierzu werden visuelle Attribute wie Farbe, Größe oder geometrische Formen analysiert. Anhand dieser Informationen werden die Anzahl und Position der gesuchten Objekte ermittelt.

In der Industrie wird für die Erkennung von Objekten ein optimaler, einfarbiger Untergrund vorausgesetzt. Hierauf wurde in dieser Beispielanwendung bewusst verzichtet, um die Leistungsfähigkeit des Systems unter widrigen Bedingungen zu zeigen.

Das Beispielsystem besteht aus einem Raspberry Pi 2 Model B+, der – wortwörtlich – auf eine Kiste montiert wurde. Für die Aufnahme der Bilder wird im Inneren der Kiste ein Standardkameramodul angebracht und an den Pi angeschlossen. Um für konstante, kontrollierbare Belichtung zu sorgen, wird die Kiste gegen Streulicht von außen abgedunkelt und von innen mit einem RGB-LED-Band belichtet (s. Abb. 1). Um Reflexionen an metallischen Oberflächen zu verhindern, wird das Licht durch eine Folie diffus gestreut.

Das Programm, welches die Verarbeitung übernimmt, ist komplett mit Java realisiert.

Um zeitgleich realistische und widrige Bedingungen zu schaffen, sollen M8-Sechskantmuttern auf grau gemustertem Teppich (s. Abb. 2) erkannt werden.

Bildverarbeitung mit OpenCV

Für die Bildverarbeitung wird die aktuell meist-verbreitete Bibliothek OpenCV genutzt. Dank der Portierung von OpenCV in Java ist es möglich, den Funktionsumfang der ursprünglich für C/C++ entwickelten Bibliothek einzusetzen. OpenCV stellt Methoden für die meisten gängigen Filter und Algorithmen in der Bildverarbeitung bereit. Ein detailliertes Verständnis der Filter ist hierbei nicht zwingend erforderlich (jedoch oft sehr hilfreich).

Die hier dargestellte Lösung beginnt mit der Aufnahme eines Bildes. Diese wird mittels gängiger Filter und Methoden manipuliert, um die Voraussetzungen für eine erfolgreiche Objekterkennung zu schaffen. Als Ergebnis des Systems werden die Koordinaten der Objekte ausgegeben. Das Verfahren verwendet bewährte Methoden für die Bilderkennung und ist auch für andere Objekte anwendbar. Jedoch erfordern verschiedene Objekte Anpassungen der Logik und der zugehörigen Schwellwerte, um optimale Ergebnisse zu erzielen.

Die digitale Bearbeitung von Bildern erfolgt durch Operationen auf Frequenz- oder Bildebene. Bei Operationen in der Bildebene, wie sie hier angewandt werden, wird das Bild in eine Matrix von Pixeln überführt. Die einzelnen Pixel können mit konstanten Werten oder Filtermatrizen verändert werden. Diese Filtermatrizen können sowohl vordefinierte als auch berechnete Werte beinhalten. Beispielsweise kann jeder Pixel in einem Graustufenbild aufgehellt oder verdunkelt werden, indem ein konstanter Wert addiert oder subtrahiert wird. Durch den Einsatz geeigneter Filter gelingt es im Idealfall, störende Bildfehler zu eliminieren (oder zumindest zu schwächen) und Kanten hervorzuheben. Bildrauschen ist einer der häufigsten Bildfehler, welches durch viele kleine Pixelfehler entsteht.

Abb. 1: Versuchsaufbau

Abb. 2: Objekte vor Hintergrund

Durch die feinen Fasern und das farbliche Muster des Teppichs ist das Eingangsbild stark verrauscht. Dieses Rauschen wird hauptsächlich mit drei ungewichteten Mittelwertfiltern reduziert, welche aufsteigende Kernelgrößen verwenden. Ein Mittelwertfilter ersetzt einen Pixel durch den Mittelwert seiner umliegenden Bildpunkte (Nachbarschaft). Die Kernelgröße definiert die Menge der miteinbezogenen Nachbarschaft. Das Filter normalisiert extreme Farbwerte und somit den Bildkontrast („Weichzeichnen“).

Objekterkennung

Nachdem das Eingangsbild vom Bildrauschen befreit ist, sollen die Muttern mit ihren Positionen ermittelt werden. Eine direkte Auswertung ist jedoch nicht möglich, da die Merkmale des Untergrunds und der Muttern eine zu geringe Differenz aufweisen. Dadurch können die Objekte nicht anhand ihrer Farbe oder Form separiert werden.

Durch eine Extraktion der Objektkonturen kann dieses Problem umgangen werden. Mit dem Canny-Algorithmus [Canny] können diese Informationen vom Bild getrennt werden. Vor der Anwendung wird die Aufnahme zunächst in ein Graustufenbild umgewandelt. Dieses wird durch Canny in ein Binärbild gewandelt, in dem zusammenhängende Konturen herausgelöst werden. Konturen sind dabei weiß und Flächen schwarz dargestellt (s. Abb. 3).

Die identifizierten Konturen werden daraufhin mit der Hough-Transformation auf geometrische Merkmale analysiert. Das Verfahren dient der Erkennung von Geraden, Kreisen und anderen geometrischen Formen in einem Binärbild. Für die Kreiserkennung wird die spezialisierte Variante Hough Circles Transform [CHT] verwendet. Dieser wird zunächst ein minimaler und maximaler Kreisradius vorgegeben. Für jeden Pixel der gefundenen Kontur wird angenommen, dass dieser Teil eines Kreises ist. Um dies zu überprüfen, wird um jeden Pixel ein Hilfskreis mit dem Radius r gelegt, welcher dem festgelegten minimalen Kreisradius entspricht (s. Abb. 4). Die Radien aller Hilfskreise werden anschließend iterativ erhöht und deren Schnittpunkte überwacht. Schneiden sich alle Hilfskreise in einem Punkt, ist dieser der Mittelpunkt eines Kreises mit dem Radius r. Dieser Kreismittelpunkt steht für die Ortskoordinate (x, y) des erkannten Objekts.

In der Realität besteht die benötigte Kreisfläche aus zu vielen Pixeln: Wird um jeden Kreispunkt ein Hilfskreis gezogen, führt das zu einem erheblichen Performanzverlust. Hier kommt der sogenannte Akkumulatorschwellwert zum Einsatz. Dieser definiert die Anzahl der benötigten Hilfskreise, um eine Kreisform zu bestätigen. Je geringer besagter Wert, desto mehr

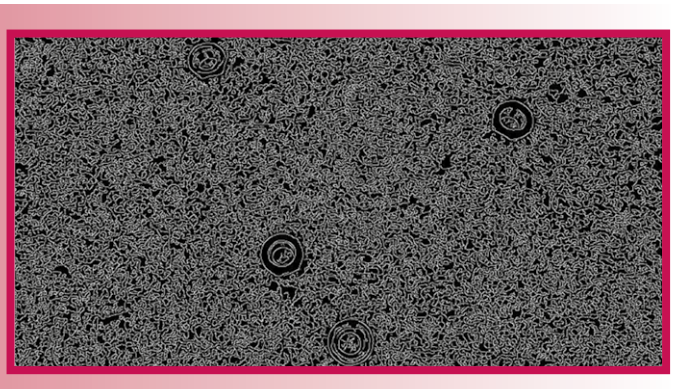

Abb. 3: Ergebnis des Canny-Algorithmus

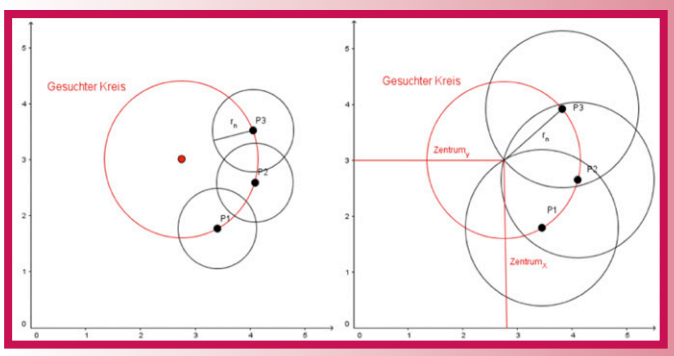

Abb. 4: Hough Circles Transform

Ellipsen und andere Geometrien werden fälschlicherweise als Kreise erkannt. Ist dieser Wert zu hoch gewählt, steigt die Verarbeitungsdauer des Systems. Da bei der Auslegung dieses Wertes kein Standardwert existiert, muss dieser unbedingt für jede Anwendung individuell angepasst werden.

Für den hier beschriebenen Anwendungsfall ergaben sich mit dem Akkumulatorschwellwert von 50 und dem minimalen Radius 15px sowie dem maximalen Radius 50px die besten Ergebnisse.

Da die Innengewinde der Muttern nahezu perfekte Kreise darstellen, kann der Akkumulatorschwellwert zur Detektion theoretisch sehr hoch gewählt werden, um möglichst wenig falsche Ergebnisse zu liefern. Dies ist jedoch mit Blick auf die Performanz nicht notwendig, da im Teppichmuster bereits ab einem Wert von 50 keine falschen Kreise mehr erkannt werden. Der Maximalradius muss größer gewählt werden als der Außenradius der Muttern. Der Minimalradius sollte möglichst nah am Radius des Innengewindes liegen.

Der Programmieraufwand für diesen Algorithmus hält sich in Grenzen. Das Foto wird über ein shell-Kommando aufgenommen, das aus Java aufgerufen wird. Raspistill ist die Software, die Fotos auf dem Pi erstellt:

```
Process p = Runtime.getRuntime().exec(
  "/opt/vc/bin/raspistill -n -bm -t 1 -w 1024 -h 768 " +
  "-q 100 -e jpg -o image.jpg");
p.waitFor();
```

Vor dem Aufruf der Methoden von OpenCV muss die Bibliothek geladen werden:

```
System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
```

Das erstellte Foto wird in zwei verschiedenen Farbcodierungen in **`BufferedImage`**s geladen: ein buntes Bild sowie eines in Graustufen:

```
File input = new File(fileName);
BufferedImage image_color, image_gray;

image_color = ImageIO.read(input);
image_gray = new BufferedImage(image_color.getWidth(),
  image_color.getHeight(), BufferedImage.TYPE_BYTE_GRAY);

image_gray.getGraphics().drawImage(image_color,0,0,null);
```

Bei der Initialisierung der Matrizen ist es sinnvoll, für Ein- und Ausgabe getrennte Matrizen zu verwenden, da die Eingabematrix teilweise durch die Operationen verändert wird. Außerdem muss zu diesem Zeitpunkt bereits bekannt sein, welche Auflösung und Größe das Bild hat. Das führt zu folgendem Vorgehen bei der Objekterkennung:

```
byte[] data_in = ((DataBufferByte) image_color.getRaster().
  getDataBuffer()).getData()

Mat mat_out, mat_gray, mat_circles, mat_bin;

mat_out = new Mat(image_color.getHeight(),
  image_color.getWidth(), CvType.CV_8UC1);
mat_gray = new Mat(image_color.getHeight(),
  image_color.getWidth(), CvType.CV_8UC1);
mat_circles = new Mat();
mat_bin = new Mat();
```

Hat die Matrix dieselbe Farbcodierung wie das Daten-Array, das gerade aus dem Bild extrahiert wurde, kann sie mit den Daten des Arrays befüllt werden. Anschließend wird das Bild durch Canny binarisiert und das Mittelwertfilter wird, wie zuvor beschrieben, dreifach angewendet. Nähere Informationen zu den Codierungen der Matrizen (`CvType`) können unter [Ninhang] nachgelesen werden:

```
mat_gray.put(0, 0, data_in);

Imgproc.Canny(mat_gray, mat_bin, 10,50, kernel_size, true);

Imgproc.blur(mat_bin, mat_out, new Size(3,3));
Imgproc.blur(mat_out, mat_out, new Size(4,4));
Imgproc.blur(mat_out, mat_out, new Size(5,5));
```

Auf der nun vorliegenden Matrix wird die Objekterkennung mit der Hough-Kreis-Detektion durchgeführt. Diese Methode liefert als Rückgabewert eine eindimensionale Matrix, die in jeder Spalte ein Double-Array enthält. Diese Arrays enthalten die X- und Y-Koordinaten der Mittelpunkte sowie den Radius der Kreise um diese Punkte. Anhand dieser Liste werden alle gefundenen Kreise in der Ausgabe-Matrix markiert.

Zuletzt wird aus der Matrix wieder ein Daten-Array extrahiert, aus welchem das Ausgabebild (s. Abb. 5) gefüllt wird:

```
Imgproc.HoughCircles(mat_out, mat_circles,
  Imgproc.CV_HOUGH_GRADIENT, 1, 20, 10, 50, 15, 50);
Point pt;
int radius;
for (int x = 0; x < mat_circles.cols(); x++) {
  double vCircle[] = mat_circles.get(0,x);
  if (vCircle == null)
    break;
  pt = new Point(Math.round(vCircle[0]), Math.round(vCircle[1]));
  radius = (int)Math.round(vCircle[2]);
  Core.circle(mat_out, pt, radius, new Scalar(0,0,255), 3);
    //circle outline
  Core.circle(mat_out, pt, 3, new Scalar(0,0,255), 3);
    //circle center
}
mat_out.get(0, 0, data_out);
image_color.getRaster().setDataElements(0, 0, mat_out.cols(),
  mat_out.rows(), data_out);
```

Abhängig vom Anwendungsfall werden nun die Koordinaten der Objekte an andere Microservices übergeben oder das Bild mit den Markierungen von Anwendern (oder Testern) betrachtet.

Multitasking statt Multithreading

Auf einem einzigen Kern (Raspberry Pi: 1 GHz) ausgeführt, benötigt das System etwa 2,2 Sekunden für die Aufnahme, Verarbeitung und Detektion. Multithreading bietet keine Steigerung der Performanz, da der Verwaltungsaufwand - gerade die Bildsegmentierung zur Verteilung auf die Threads - zu aufwendig wäre.

Eine weitere Skalierungsmethode bietet die Steigerung des Durchsatzes. Statt eine Instanz auf vier Kerne aufzuteilen, wird jedem Kern eine eigene Instanz zugewiesen. Dabei bleibt die

Abb. 5: Detektierte Objekte (Anmerkung: Rotstich im Zentrum ist ein bekanntes Phänomen bei günstigen Kameras)

Ausführungszeit jeder Instanz bei 2,2 Sekunden, allerdings werden so wesentlich mehr Ergebnisse pro Minute generiert. Hierfür ist es notwendig, die Ausführungen der einzelnen Instanzen zeitlich zu versetzen. Aus diesem Grund wird alle 0,55 Sekunden eine neue Instanz gestartet. Mit diesem Verfahren liefert das System im Dauerbetrieb zwei Bilder pro Sekunde.

Sollte die Performanz dennoch nicht ausreichen, kann das System noch skaliert werden. Werden beispielsweise vier Raspberry Pi mit je einer Kamera installiert, die wiederum vier Instanzen der Software in oben beschriebener Art betreiben, vervierfacht sich der Durchsatz nochmals. Theoretisch kann das Gesamtsystem dadurch etwa acht Bilder pro Sekunde auswerten. Praktisch sollte jedoch bedacht werden, dass die Berechnungsdauer schwanken kann und für die Synchronisation der Ergebnisse ebenfalls ein kleines Zeitfenster einberechnet werden muss. Tests haben ergeben, dass die maximale Dauer – je nach Anzahl der Objekte im Bild – zwischen 2,6 und 2,7 Sekunden beträgt.

Keine hundertprozentige Genauigkeit

Das Kameramodul des Pi ist dafür ausgelegt, unter allen Bedingungen passable Bilder zu erstellen. Es korrigiert automatisch die Lichtverhältnisse, was für eine normale Nutzung von Vorteil ist. Die Verarbeitung wird dadurch jedoch erheblich erschwert, da unter denselben Belichtungsverhältnissen unterschiedliche Aufnahmen generiert werden. Deshalb muss der Schwellwert abhängig vom Anwendungsfall so gewählt werden, dass entweder einige wenige Objekte nicht erkannt werden oder einige (nicht existierende) „Objekte“ fälschlicherweise erkannt werden.

Die Ergebnis-Genauigkeit kann erhöht werden, indem mehrere Aufnahmen derselben Objekte erstellt und miteinander verglichen werden. Das geschieht allerdings auf Kosten der Performanz. Wird eine bessere Qualität der Erkennung bei konstant bleibendem Durchsatz benötigt, muss das Kameramodul durch eine geeignetere Kamera ersetzt werden, wodurch die Kosten des Gesamtsystems gegebenenfalls stark ansteigen.

Aus Industriekamera wird Industrie 4.0-Kamera

In diesem Zustand hat die Kamera noch nichts mit Industrie 4.0 oder IoT zu tun. Erst die Kommunikation mit anderen Devices (s. Abb. 6) ergibt einen entscheidenden Mehrwert für das System. Diese Kommunikation erfolgt mittels MQTT [MQTT],

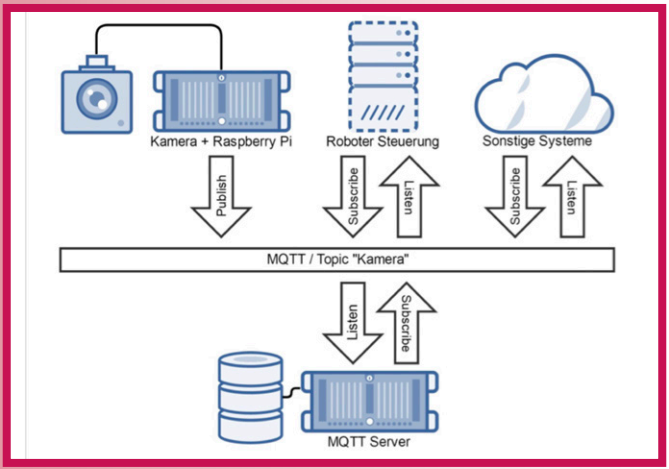

Abb. 6: Architekturbild

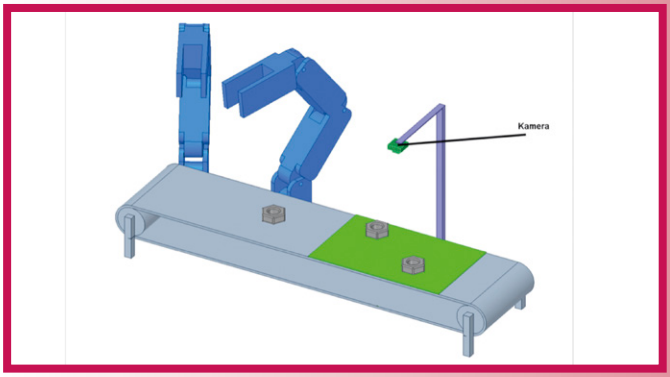

Abb. 7: Industrieller Showcase

dem aktuell wohl meistgenutzten Protokoll zur Kommunikation zwischen IoT-Devices. Es setzt auf TCP/IP auf und arbeitet nach dem Publish/Subscribe-Prinzip. Publisher und Subscriber einigen sich auf ein Topic, anhand dessen die Nachrichten zugeordnet werden können.

Am Subscriber dieser Nachrichten ist aktuell eine MongoDB aufgesetzt, welche die Ergebnisse der Kamera für spätere Auswertungen speichert. Das Kamerasystem publiziert nach der Detektion die Koordinaten der Objekte unter dem definierten Topic.

Folgender Anwendungsfall (s. Abb. 7) soll die Fähigkeiten des Systems verdeutlichen: Eine Zuführungsanlage, bestehend aus einem Förderband, führt einer Industrieanlage Werkstücke zu, die von mehreren Roboterarmen gegriffen werden sollen. Bei den Werkstücken handelt es sich um Schraubenmuttern. Die Position dieser Muttern kann auf dem Förderband beliebig sein und muss von der Objekterkennung detektiert werden, um deren Position an die Roboter zu übermitteln, damit diese die Muttern greifen können. Der zeitliche Versatz zwischen der Auswertung der Bilder und der tatsächlichen Position der Objekte kann dabei anhand der Geschwindigkeit des Förderbandes und dem Zeitpunkt der Aufnahme des Bildes berechnet werden. Diese Information über die Position kann nun den Robotern übermittelt werden, welche die Muttern greifen.

Der Kostenfaktor spielt in diesem Projekt eine wichtige Rolle. In Tabelle 1 werden die notwendigen Kosten für das System aufgelistet. Die Liste beinhaltet keine optionalen Kosten, wie die Beleuchtung.

Artikel	Kosten
Raspberry Pi 2 Model B+	42,99 €
Kameramodul	28,90 €
32 GB Micro-SDHC-Karte	10 €

Tabelle 1: Kostenaufstellung

Fazit

Das System, bestehend aus einem Einplatinencomputer samt preisgünstigem Kameramodul und einfachster Beleuchtung, erzielt für die Erkennung von Objekten mit kreisrunden Konturen sehr gute Resultate. Sollen andere Konturen erkannt werden, sind Anpassungen notwendig, wodurch Performanz sowie Genauigkeit unter Umständen stark leiden. Je nachdem, welches Attribut wichtiger ist, müssen diese zwei Werte ausbalanciert werden. Die Defizite in der Leistung holt das System durch den Preis und die damit verbundene Skalierbarkeit wieder auf.

Verbindet man fünf solcher Systeme zu einem Cluster und regelt die Ausführungszeiten, wird die Leistung deutlich erhöht, ohne den Preis in den vierstelligen Bereich zu treiben. Ein weiteres Ausbauszenario ist der redundante Betrieb von mehreren Systemen, die sich gegenseitig kontrollieren und ihre Ergebnisse abgleichen. Da die Software jederzeit angepasst werden kann, sind vielfältige Ausbaustufen denkbar. Für die beschriebenen Anwendungsfälle liefert das System mit geringem finanziellem Aufwand sehr gute Ergebnisse.

Literatur und Links

[Canny] Canny Edge Detector, OpenCV 2.4.13.0 Documentation, `http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html`

[CHT] `https://en.wikipedia.org/wiki/Circle_Hough_Transform`

[MQTT] Message Queue Telemetry Transport, Version 3.1.1, `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`

[Ninhang] List of Mat Type in OpenCV, `http://ninghang.blogspot.de/2012/11/list-of-mat-type-in-opencv.html`

[OpenCV] OpenCV-Dokumentation, `http://docs.opencv.org/2.4/`

Marcel Bagemihl ist als Consultant bei der NovaTec Consulting GmbH tätig. Er ist für die Entwicklung von IoT- und Industrie 4.0-Anwendungen zuständig. Besonders vertraut ist er mit der industriellen Objekterkennung.
E-Mail: marcel.bagemihl@novatec-gmbh.de

Simon Dörner ist als Consultant bei der NovaTec Consulting GmbH im Umfeld von Industrie 4.0-Anwendungen tätig. Sein Schwerpunkt liegt auf der dreidimensionalen Visualisierung von Produktionsprozessen.
E-Mail: simon.doerner@novatec-gmbh.de

Steffen Krieg verantwortet den Themenbereich IoT und Industrie 4.0 bei der NovaTec Consulting GmbH. Sein Fokus liegt auf der Implementierung und Beratung im Gebiet cyber-physikalische Systeme.
E-Mail: steffen.krieg@novatec-gmbh.de