

Anwendungen mit dem Xbox 360 Kinect-Controller entwickeln

Bewegungsdrang

Mithilfe des Kinect-Controllers lassen sich Anwendungen mit natürlichen Benutzerschnittstellen realisieren. Für Entwickler ergeben sich damit neue Möglichkeiten. Ein Software Development Kit für das Gerät zeigt, in welche Richtung sie gehen.

Natürliche Benutzerschnittstellen, auch Natural User Interfaces (NUI) genannt, sind wegweisend für künftige Anwendungen. Dabei erfolgt die Interaktion zwischen Mensch und Maschine in einer Weise, die der Mensch als intuitiv empfindet, nämlich mit Gesten. Damit greift sie auf Fähigkeiten zurück, die der Nutzer im Laufe seines Lebens erworben hat. Die Ansätze sind vielfältig und reichen vom Einsatz von Touchscreens bis hin zu einer Gestensteuerung mit dem gesamten Körper.

Für die Gestensteuerung eignet sich der Kinect-Controller besonders gut. Eingesetzt wird er bei Microsofts eigener Spielkonsole Xbox 360. Was ihn so interessant macht, ist die technische Ausstattung im Verhältnis zum Anschaffungspreis.

Bisher war eine optische Bilderfassung, wie sie der Controller mitbringt, auf relativ teure Geräte beschränkt, die nicht für den Hausgebrauch gedacht sind. Mit dem Kinect-Controller aber ist ein günstiges Gerät mit interessanter Technik erhältlich.

Funktionsweise des Controllers

Der Controller verfügt im Wesentlichen über eine RGB-Kamera und zwei Infrarotsensoren. Erstere liefert einen üblichen Video-Stream, vergleichbar mit einer Webcam (**Abbildung 1**). Mit den anderen beiden Sensoren sendet und empfängt das Gerät ein Infrarotsignal und tastet damit die Umgebung ab. Der Controller misst dabei, wie lange das Signal unterwegs ist, bis es reflektiert wird. Aus diesen beiden Informationen, Bild und Entfernung, kann er die Entfernung von Objekten bestimmen und sie im Zusammenhang mit anderen Objekten erfassen.

Für eine optimale Erkennung sollten sich die Akteure in einem Abstand von 1,2 bis 3,5 Metern vor dem Kinect-Controller befinden. Die Frame-Rate liegt bei beiden optischen Sensoren bei 30 Frames pro Sekunde, das RGB-Bild enthält dabei mit einer VGA-Auflösung von 640 x 480 Bildpunkten etwas mehr als das Tiefenbild (320 x 240 Bildpunkte). Das Farbbild kann auch mit einer Auflösung von 1280 x 1024 abgerufen werden, jedoch stehen dann weitaus weniger Frames pro Sekunde zur Verfügung. Zur Audioaufnahme ist im Controller ein Mikrofon eingebaut, das auch der Spracherfassung dient.

Identifizierung von Objekten

Die räumliche Erfassung über Infrarotsignale macht es einfacher, Objekte zu identifizieren, während eine reine Farbbilderfassung sich lediglich auf das Erkennen von Mustern in den Bildern stützt.

Das zweite Verfahren ist relativ fehleranfällig, da Informationen über die dritte Dimension fehlen und die Dimensionen nur sehr schwer zu unterscheiden sind. Durch die räumliche Einordnung der Umgebung mit Infrarotsignalen lassen sich Hintergrundobjekte erkennen und ausschließen, was den Erkennungsprozess von Körpern und Bewegungen stark unterstützt.

Mithilfe der im Kinect-SDK integrierten Mustererkennungsverfahren lässt sich errechnen, an welchen Positionen und in welchem Zustand sich Körperteile des Nutzers befinden. Auf diesem Weg lassen sich auch Bewegungen erfassen.

Das Kinect-SDK

Derzeit befindet sich das Software Development Kit (SDK) für das Gerät noch im Betastadium und steht auf der offiziellen Website kostenlos zur Verfügung [1]. Für die Arbeit mit Audioeingaben sind auch noch das Microsoft Speech Platform SDK [2], die Laufzeitumgebung Speech Platform Server Runtime [3] und das Kinect for Windows Runtime Language Pack [4] nötig. Nach erfolgreicher Installation des SDK sollten im Gerätemanager folgende Einträge unter *Microsoft Kinect* aufgelistet sein:

- *Microsoft Kinect Audio Array Control*,
- *Microsoft Kinect Camera*,
- *Microsoft Kinect Device*.

In der Kategorie *Audio-, Video- und Gamecontroller* sollte sich außerdem der Eintrag *Kinect USB Audio* befinden.

Derzeit bieten sich dem Entwickler mit dem SDK die folgenden Möglichkeiten: Zum einen liefert der Controller einen Farb-Stream, wie er von Webcams her bekannt ist; zum anderen erhält der Entwickler detaillierte Informationen, die die Entfernung der Umgebung vom Sensor darstellen. Zu den Stärken des SDK gehört, dass es automatisch ein Skelettmodell erzeugt, das einzelne Körperteile unterscheidet. Dabei liegen 20 Schlüsselpunkte des Körpers zugrunde [5]. Dieses Modell kann bis zu zwei Personen erfassen.

Auf einen Blick



Andy Stumpp ist Inhaber von BrilliantVision und befasst sich mit der Entwicklung von Software auf der Basis von .NET-Technologien. In seiner Freizeit beschäftigt er sich leidenschaftlich mit der Makrofotografie. Sie erreichen ihn über www.brilliantvision.de.

Inhalt

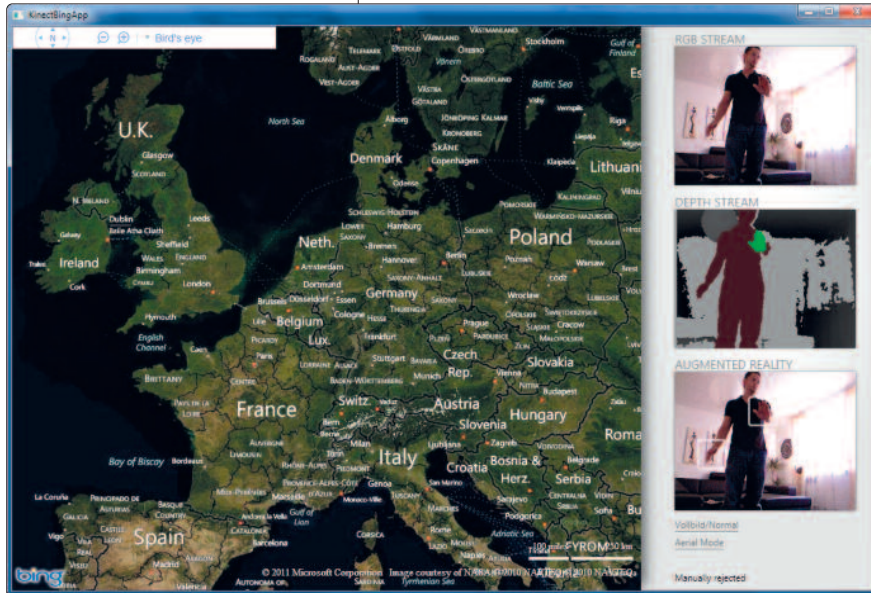
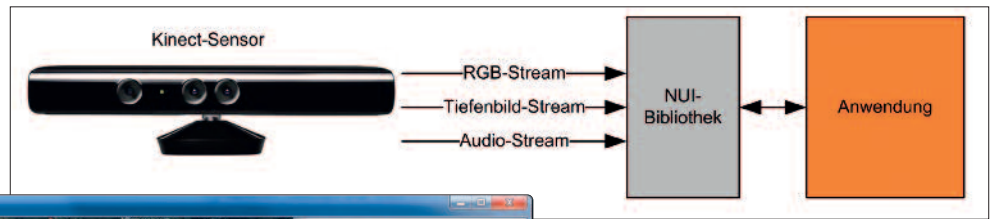
- ➔ Die Funktionsweise des Kinect-Controllers.
- ➔ Das Kinect-SDK zum Erkennen von Objekten nutzen.
- ➔ Augmented Reality auf einfache Weise.

dnpCode

A1202Kinect



[Abb. 1] Das Schema der Schnittstelle zum Kinect-Controller.



[Abb. 2] Die Beispielanwendung, die sich mit Gesten steuern lässt.

Neben den optischen Eingangsdaten lassen sich Audiodaten des integrierten Mikrofons aufzeichnen oder direkt über die vom SDK implementierte Spracherkennung nutzen. Die besondere Bauweise des Mikrofons ermöglicht es außerdem, den Standort der Quelle zu lokalisieren.

Eine Beispielanwendung soll die Nutzung des SDK demonstrieren. Dabei handelt es sich um eine Anwendung, deren Oberfläche mithilfe der Windows Presentation Foundation (WPF) aufgebaut wird. Sie enthält ein Bing-Kartensteuerelement und lässt sich über natürliche Bewegungen steuern (Abbildung 2). Der Nutzer soll die Karte sowohl bewegen als auch hinein- oder hinauszoomen können, und die Zoom-Funktion soll ebenfalls über Sprachkommandos auszuführen sein.

Erkennung von Gesten

Winkt der Nutzer mit einer Hand oder bringt er beide zusammen, kann dies als Geste gewertet werden, auf die die Anwendung reagieren soll. Derzeit erkennt das Kinect-SDK solche Gesten noch nicht als solche. Es stellt sich daher die Frage, wie sich die räumlichen Sensoren nutzen lassen, um Gesten zu identifizieren. Grundsätzlich lassen sich zwei Ansätze unterscheiden:

- die algorithmische Suche und
- die vorlagenbasierte Suche.

Bei der algorithmischen Suche wird mithilfe eines Algorithmus der zeitliche Ablauf der Gestik untersucht. Die Anwendung kann etwa prüfen, ob sich die Position der Hand zum Zeitpunkt t um x Längeneinheiten im Vergleich zu $t - 1$ geändert hat.

Die vorlagenbasierte Suche greift auf ein Muster zurück, das zuvor bereits aufgezeichnet wurde. Dabei handelt es sich um Werte, die beim Bewegen der Hand in einer Kalibrierungssitzung gespeichert werden. Das hier erläuterte Beispiel soll sich auf ein einfaches Erkennen von Gesten auf Basis von Algorithmen beschränken.

Eine weitere Herausforderung ist das Ermitteln der steuernden Gesten. Dabei ist darauf zu achten, dass die Zuordnung von Gesten zu Befehlen eindeutig ist und diese Gesten in der Realität hinreichend unterscheidbar sind. Das Bewegen der Karte mit der rechten Hand in x -Richtung und die gleichzeitige Verwendung derselben Hand zum Zoomen würde sich unweigerlich zu einer komplexen Anwendungsnavigation entwickeln, sowohl aus der Sicht des Nutzers als auch aus der des Entwicklers. Schließlich müssen auf Entwicklerseite bei der Verwendung von NUIs immer Situationen berücksichtigt werden, in denen der Benutzer ungewollt durch eine Körperbewegung eine Programmaktion auslöst. Dadurch würde eine Geste erkannt, obwohl

dies vom Nutzer nicht beabsichtigt war. Abhilfe kann eine Differenzierung von Gesten im Hinblick auf Randkriterien schaffen, wie zum Beispiel folgende:

- Entfernung des Nutzers vom Sensor,
- Position/Winkel des Nutzers zum Sensor,
- Haltung der anderen Hand.

Initialisierung des SDK

Eine Anwendung, die das SDK nutzen soll, muss die *Microsoft.Research.Kinect*-Assembly referenzieren; außerdem sollte sie den dazugehörigen Namensraum importieren, da dort alle zur optischen Erkennung notwendigen Elemente untergebracht sind:

```
using Microsoft.Research.Kinect.Nui;
```

Anschließend kann ein *Runtime*-Objekt erzeugt werden, das es ermöglicht, den Kinect-Controller in Betrieb zu nehmen:

```
runtime = new Runtime();
```

Sind mehrere Controller angeschlossen, kann der Konstruktor unter Angabe des Controller-Index aufgerufen werden. Außerdem ermöglicht es das Ereignis *Runtime.Kinects.StatusChanged*, einen neu angeschlossenen Controller zu erkennen und zu übergeben. Initialisiert werden muss das *Runtime*-Objekt mit Angaben dazu, welche Daten-Streams entgegengenommen werden sollen. Dabei lassen sich die Argumente mit dem bitweisen ODER-Operator verknüpfen, hier die für das Tiefenbild inklusive Spielererkennung, Skelettmodell und RGB-Bild:

```
runtime.Initialize(
    RuntimeOptions.UseDepthAndPlayerIndex
    | RuntimeOptions.UseSkeletalTracking
    | RuntimeOptions.UseColor);
```

Anschließend erwarten Tiefenbild- wie Video-Stream weitere Angaben zum Format, bevor sie Daten aufnehmen können. Der RGB-Stream kann mit der *Open()*-Methode wie folgt konfiguriert werden:

```
runtime.VideoStream.Open(
    ImageStreamType.Video,
    2,
    ImageResolution.Resolution640x480,
    ImageType.Color);
```

Dabei wird mit *ImageStreamType.Video* der Typ des Datenstroms explizit angegeben und mit *ImageResolution.Resolution-640x480* die Auflösung; die Zahl 2 im zweiten Parameter gibt an, wie viele Frames die Runtime im Buffer halten soll, und ist eine Empfehlung der Dokumentation. Ähnlich wie hier funktioniert auch die Initialisierung des Tiefenbild-Streams.

Zur Beschaffung eines fertig erstellten Bildes aus dem Stream bietet das SDK zwei Möglichkeiten:

- das Polling-Modell und
- das Event-Modell.

Das erste Modell erlaubt es dem Entwickler, aktuelle Werte mittels zyklischer Abfragen zu ermitteln. Das Event-Modell hingegen bietet, wie der Name bereits vermuten lässt, Ereignisse an, die ausgelöst werden, sobald aktuelle Daten zur Verfügung stehen. Die Beispielanwendung arbeitet mit diesem zweiten Modell. Sie abonniert das *VideoFrameReady*-Ereignis und gibt dabei eine Methode zum Empfang der Daten an, welche die Daten übernimmt und weiterverarbeitet, hier ist es die Methode *nui_VideoFrameReady()*:

```
runtime.VideoFrameReady +=
    new EventHandler
    <ImageFrameReadyEventArgs>
    (nui_VideoFrameReady);
```

Analog geschieht dies beim Skelettmodell wie auch beim Tiefenbild-Stream.

Datenverarbeitung

Zuerst soll der RGB-Datenstrom eingefangen und in einer *Image*-Komponente angezeigt werden, ähnlich wie bei einer Webcam; der agierende Nutzer wird lediglich in der Anwendung abgebildet.

Listing 1

Die Markierung des Spielers in Grün.

```
if (intensity > 180)
{
    _depthFrame32[i32 + RED_IDX] = 0;
    _depthFrame32[i32 + GREEN_IDX] = 200;
    _depthFrame32[i32 + BLUE_IDX] = 0;
}
else
{
    _depthFrame32[i32 + RED_IDX] = 50;
    _depthFrame32[i32 + GREEN_IDX] = 0;
    _depthFrame32[i32 + BLUE_IDX] = 0;
}
```

Durch die Registrierung beim Eventhandler wird bei jedem fertiggestellten RGB-Bild die *nui_VideoFrameReady()*-Methode aufgerufen und es werden Bilddaten vom Typ *PlanarImage* übergeben. Diese Daten müssen für die Anzeige in ein gültiges Format umgewandelt werden, was folgendermaßen geschieht:

```
PlanarImage planarImage =
    e.ImageFrame.Image;
_lastVideoFrame = BitmapSource.Create(
    planarImage.Width,
    planarImage.Height,
    96,
    96,
    PixelFormats.Bgr32,
    null,
    planarImage.Bits,
    planarImage.Width*planarImage.
    BytesPerPixel);
```

Die Variable *_lastVideoFrame* ist vom Typ *BitmapSource* und kann der *Image*-Komponente zugewiesen werden, die sich auf der Programmoberfläche befindet. Wer dazu noch das Kinect-Toolkit von CodePlex [6] nutzt, kann die diversen Typumwandlungen, die bei der Arbeit mit dem Kinect-SDK nötig sind, vereinfachen.

Bei der Verarbeitung der Tiefenbildinformationen ist etwas mehr Arbeit nötig. Der Spieler selbst soll in einer bestimmten Farbe dargestellt werden, während die Umgebung des Spielers je nach Entfernung als Graustufenbild angezeigt werden soll. Dazu ist es notwendig, das eingehende Bildformat zu verstehen. Die übergebenen Daten bei einem fertigen Tiefenbild bestehen aus einem *Byte*-Array, das 16-Bit-Frames enthält.

Die ersten drei Bit eines Rahmens bestehen aus der Player-ID, die den Spieler kennzeichnet. Die folgenden 13 Bit geben Informationen über die Entfernung des Objekts an dieser Position preis.

Die Spieler-ID lässt sich mit einer Binäroperation auf den Frame ermitteln:

```
int player = depthFrame16[i16] & 0x07;
```

0x07 stellt die Binärzahl 111 dar und bewirkt als UND-Verknüpfung, dass nur die ersten drei Bit aus der Operation übernommen werden. Analog werden dem Frame die übrigen 13 Bit entnommen. Da handelsübliche Monitore nur 8-Bit-Grauwerte ($2^8 = 256$) darstellen können, muss der 13-Bit-Wert auf 8 Bit reduziert werden:

```
byte intensity = (byte) (255 -
    (255 * realDepth / 0x0fff));
```

Der Grauwert kann anschließend einem in WPF darstellbaren 32-Bit-Frame zuge-

ordnet werden. Das Verfahren dabei ist für jede der Farbkomponenten gleich:

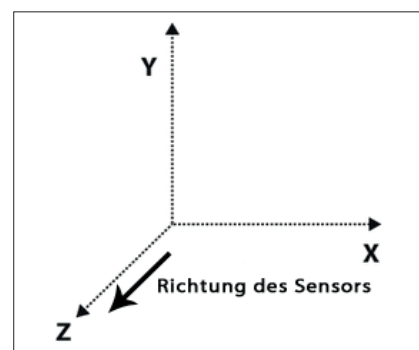
```
_depthFrame32[i32 + RED_IDX] =
    (byte) (intensity/2);
_depthFrame32[i32 + GREEN_IDX] =
    (byte) (intensity/2);
_depthFrame32[i32 + BLUE_IDX] =
    (byte) (intensity/2);
```

Für die Tiefeninformationen eines Spielers wird differenziert vorgegangen. Eine Grün-Markierung soll kenntlich machen, dass sich der Spieler in der vorgeschriebenen Nähe zum Sensor befindet (**Listing 1**).

Bewegungserkennung

Das Erkennen der Gesten soll bei der Übernahme des Skelettmodells geschehen, da dort detaillierte Informationen über Körperteile zur Verfügung stehen. Die Körperteile werden im SDK als sogenannte Joints repräsentiert und anhand eindeutiger IDs identifiziert. Zur algorithmischen Erkennung von Gesten muss eine Übereinstimmung des zeitlichen Ablaufs der Bewegung mit einem solchen Joint gesucht werden. Dies soll im Programmbeispiel geschehen, indem die Position der rechten Hand zwischengespeichert wird. Zum Zeitpunkt *t* prüft der Algorithmus, ob sich die Bewegung im Vergleich zum Zeitpunkt *t - 1* geändert hat. Das Vorzeichen der Differenz bestimmt zudem die Richtung der Bewegung in einer Dimension. Befindet sich nur die rechte Hand im gültigen Bereich, soll ihre Bewegung die Karten steuern (**Listing 2**).

Befindet sich hingegen nur die linke Hand im gültigen Bereich, soll der Zoom-Modus aktiviert werden. Der Benutzer kann mit der Bewegung der rechten Hand in z-Richtung die Karte vergrößern oder verkleinern. Zu beachten ist dabei, dass ein rechtshändiges Koordinatensystem zugrunde liegt, bei dem sich der Sensor im Ursprung befindet (**Abbildung 3**).



[Abb. 3] Das rechtshändige Koordinatensystem der Kinect-Steuerung.

Listing 2

Die Navigation.

```
if (rightHandVector.Z < 1.2 && leftHandVector.Z > 1.2)
{
    double rightHorizontalDiff = _oldRightHandVector.Value.X - rightHandVector.X;
    double rightVerticalDiff = _oldRightHandVector.Value.Y - rightHandVector.Y;

    double rightAbsoluteHorizontalDiff = Math.Abs(rightHorizontalDiff);
    double rightAbsoluteVerticalDiff = Math.Abs(rightVerticalDiff);

    // Horizontale Navigation
    if (rightAbsoluteHorizontalDiff > 0.005)
    {
        if (rightHorizontalDiff < 0)
        {
            _map.NavigateLeft();
        }
        else
        {
            _map.NavigateRight();
        }
    }
    else
    {
        // Vertikale Navigation
        if (rightAbsoluteVerticalDiff > 0.005)
        {
            if (rightVerticalDiff < 0)
                _map.NavigateDown();
            else
                _map.NavigateUp();
        }
    }
}
```

Listing 3

Die Hand-Position in Canvas-Koordinaten umrechnen.

```
float depthX, depthY;
_runtime.SkeletonEngine.SkeletonToDepthImage(joint.Position, out depthX,
    out depthY);
depthX = Math.Max(0, Math.Min(depthX * 320, 320));
depthY = Math.Max(0, Math.Min(depthY * 240, 240));
int colorX, colorY;
ImageViewArea iv = new ImageViewArea();
_runtime.NuiCamera.GetColorPixelCoordinatesFromDepthPixel(
    ImageResolution.Resolution640x480, iv, (int) depthX, (int) depthY, (short) 0,
    out colorX, out colorY);

return new Point((int) (skeletonCanvas.Width * colorX/640.0),
    (int) (skeletonCanvas.Height * colorY/480));
```

Das Verfahren, das hier zur Anwendung kommt, ist zur Anschauung kurz und simpel gehalten. Es berücksichtigt nicht die Zeit, die seit dem Speichern der letzten Position vergangen ist, sondern verlässt sich auf eine gleichmäßige Lieferung von Skelettrahmen aus dem SDK. Das Verfahren lässt sich natürlich optimieren.

Augmented Reality

Ein weiterer, sehr spannender Anwendungsbereich, der mit dem Kinect-Controller erschlossen werden kann, ist Augmented Reality. Dabei handelt es sich um aufgenommene Bilder der Umwelt, die bei der Darstellung in Echtzeit um Informationen und Markierungen im Bild ergänzt werden.



[Abb. 4] Augmented Reality mit dem Kinect-SDK in der Beispielanwendung.

Bei der Demoanwendung sollen die RGB-Bilddaten um die Kennzeichnung der Hände erweitert werden. WPF erlaubt es, auf einfache Weise ansprechende Anwendungen mit Augmented-Reality-Unterstützung zu erstellen. Das Zeichnen der Handmarkierungen kann wieder beim Entgegennehmen der Skelettinformationen erfolgen, da dort detaillierte Informationen über Körperteile zur Verfügung stehen. Dabei müssen lediglich die Koordinaten der Hände auf die Größe der Ausgabe Komponente umgerechnet werden, was mithilfe der `getDisplayPosition()`-Methode möglich ist (Listing 3).

Als Komponente zur Ausgabe dient ein *Canvas*-Element, das umfangreiche Zeichenmethoden zur Verfügung stellt. Für die Handmarkierung wird ein abgerundetes Rechteck verwendet, das durch die Klasse *HandMarker* implementiert wird. Nach der Zuweisung der aktuellen Koordinaten ist auch schon eine kleine Augmented-Reality-Anwendung fertig (Abbildung 4).

Spracherkennung

Neben den optischen Sensoren bietet der Kinect-Controller in Verbindung mit dem SDK die Möglichkeit, Spracheingaben in die eigene Applikation einzubauen. Die Beispielanwendung soll einfache Kommandos wie „Kinect zoom in/out“ beherrschen, um das Hinein- und Hinaus-Zoomen aus der Karte per Spracheingabe zu steuern.

Wer bereits mit dem Microsoft Speech SDK vertraut ist, wird sich im Hinblick auf die Spracherkennung des Kinect-Controllers schnell zurechtfinden. Die Klassen *Choices*, *Grammar* und *GrammarBuilder* bilden das Grundgerüst für die Spracherkennung in .NET. Mögliche Spracheingabekommandos werden einer Instanz der

Listing 4

Die Spracheingabe initialisieren.

```

_kinectSource = new KinectAudioSource();

_kinectSource.FeatureMode = true;
_kinectSource.AutomaticGainControl = false;
_kinectSource.SystemMode = SystemMode.OptibeamArrayOnly;

var rec = (from r in SpeechRecognitionEngine.InstalledRecognizers() where r.Id ==
    RecognizerId select r).FirstOrDefault();

_speechEngine = new SpeechRecognitionEngine(rec.Id);

var choices = new Choices();
choices.Add(KINECT_ZOOM_IN_SPEECH);
choices.Add(KINECT_ZOOM_OUT_SPEECH);

GrammarBuilder gb = new GrammarBuilder();
gb.Culture = rec.Culture;
gb.Append(choices);

var g = new Grammar(gb);

_speechEngine.LoadGrammar(g);
_speechEngine.SpeechHypothesized += new EventHandler
    <SpeechHypothesizedEventArgs>(speechHypothesized);
_speechEngine.SpeechRecognized += new EventHandler
    <SpeechRecognizedEventArgs>(speechRecognized);
_speechEngine.SpeechRecognitionRejected += new EventHandler
    <SpeechRecognitionRejectedEventArgs>(speechRecognitionRejected);

_audioStream = _kinectSource.Start();

_speechEngine.SetInputToAudioStream(_audioStream, new SpeechAudioFormatInfo(
    EncodingFormat.Pcm, 16000, 16, 1, 32000, 2, null));

_speechEngine.RecognizeAsync(RecognizeMode.Multiple);

```

Klasse *Choices* übergeben, die wiederum einem Objekt der Klasse *GrammarBuilder* hinzugefügt wird (Listing 4). Die Klasse *SpeechEngine* nimmt die entstandene Grammatik entgegen und stellt Ereignisse zur Verfügung, um über den Erkennungsprozess informiert zu werden. Das Ereignis *SpeechHypothesized* wird ausgelöst, sobald potenzielle Spracheingaben das Mikrofon erreichen. Die erfolgreiche Erkennung eines in einer Grammatik gelisteten Wortes wird durch das Auslösen des *SpeechRecognized*-Ereignisses angezeigt (Listing 5).

Beim Entgegennehmen des erkannten Wortes aus der Grammatik lässt sich ebenfalls prüfen, mit welcher Wahrscheinlichkeit es sich tatsächlich um dieses Wort handelt. Dabei werden in der Beispielanwendung nur Spracheingaben von mindestens 95-prozentiger Wahrscheinlichkeit akzeptiert. Ein manuelles Ausloten in Abhängigkeit vom Anwendungsfall und den zu er-

kennenden Wörtern scheint auf jeden Fall sinnvoll. Bei einem zu niedrigen Schwellenwert besteht die Gefahr, dass das SDK bei nahezu jedem Umgebungsgeschrei eines der Wörter aus der Grammatik erkennt.

Fazit

Das Kinect-SDK befindet sich noch in der Beta-Phase und bietet keine eigene Erkennung von Bewegungen. Das heißt, dass sich der Entwickler selbst um diese Belange kümmern muss. Nichtsdestotrotz zeigt das SDK bereits, welche Möglichkeiten sich mit der Bewegungssteuerung auf tun. Der Kreativität und spannenden Anwendungen mit einer neuartigen Steuerung scheinen dabei kaum Grenzen gesetzt. Bereits veröffentlichte Projekte reichen von einer Blindenführung durch den Controller bis hin zur Körpersteuerung von realen Flugobjekten. Eine Reihe von Anwendungen ist bei Coding4Fun zu finden [7].

Listing 5

Das Erkennen von Wörtern.

```

private void speechRecognized(object sender,
    SpeechRecognizedEventArgs e)
{
    string result = e.Result.Text;

    if (e.Result.Confidence < 0.95)
    {
        ShowSpeechState("Manually rejected");
        return;
    }

    ShowSpeechState(result);

    if (e.Result.Text ==
        KINECT_ZOOM_IN_SPEECH)
    {
        _map.ZoomIn();
    }
    else if (e.Result.Text ==
        KINECT_ZOOM_OUT_SPEECH)
    {
        _map.ZoomOut();
    }
}

private void ShowSpeechState(string msg)
{
    speechStatus.Text = msg;
}

```

Das Programmiermodell ist einfach gehalten und sollte nach kurzer Einarbeitung gut zu handhaben sein. Eine der Herausforderungen wird darin bestehen, die Interaktion mit dem Benutzer so zu gestalten, dass sie sowohl nachvollziehbar als auch eindeutig ist.

Spannende Anwendungen stehen also vor der Tür. Wer nicht warten will, programmiert am besten die eigene Kinect-Anwendung. In diesem Sinne viel Spaß bei der Entwicklung. [jp]

- [1] Kinect for Windows SDK, www.dotnetpro.de/SL1201Kinect1
- [2] Microsoft Speech Platform SDK, www.dotnetpro.de/SL1201Kinect2
- [3] Microsoft Speech Platform Server Runtime, www.dotnetpro.de/SL1201Kinect3
- [4] Kinect for Windows Language Pack, www.dotnetpro.de/SL1201Kinect4
- [5] Kinect Programming Guide, www.dotnetpro.de/SL1201Kinect5
- [6] Coding4Fun Kinect Toolkit, www.dotnetpro.de/SL1201Kinect6
- [7] Coding4Fun Kinect Projects, www.dotnetpro.de/SL1201Kinect7