Bildverarbeitung mit Webcams

von Reinhard Oldenburg

Eine *Webcam* ist eine Kamera, deren Bilder direkt in einen Computer, ein Computernetz oder auf eine Seite des *World Wide Web* übertragen werden können. Webcams werden häufig durch Firmen oder öffentliche Einrichtungen betrieben, die über eine Standleitung zum Internet verfügen. Aber auch Privatpersonen (z. B. mit DSL-Anschluss) stellen zunehmend Webcam-Aufnahmen bereit. Die Motive sind unterschiedlicher Natur: Von der reinen Hobbyausübung, über Exhibitionismus und Werbung bis hin zu finanziellen Interessen. Dargestellt werden alle möglichen Inhalte: Straßen und öffentliche Plätze (Bild 1), Büroarbeitsplätze, Baustellen, Privatwohnungen und anderes.

Die für den Betrieb notwendige Technik ist inzwischen preiswert und weltweit sehr verbreitet. Dies erlaubt früher in dieser Weise nicht mögliche Einblicke in das Privatleben von Menschen, die sich aus verschiedenen Gründen – oftmals rund um die Uhr – von einer Webcam überwachen lassen (Stichwörter sind: Exhibitionismus, Voyeurismus). Diese Beispiele sind Zeichen einer Gesellschaft, die immer mehr dazu neigt, selbst die persönlichsten und intimsten Angelegenheiten wie das eigene Leben und Sterben der Öffentlichkeit zugänglich zu machen. Ein Unterricht, der zu kritischem und mündigem Denken erziehen will, muss u. a. diese bedenklichen Tendenzen thematisieren. Auch die in diesem Beitrag gezeigten Möglichkeiten sollten mit den Schülerinnen und Schülern in diesem Sinne diskutiert werden.

Als erste Webcam gilt die *Trojan Room Coffee Pot Camera* (Stafford-Frazer, 2001), die von 1991 bis 2001 Bilder einer Kaffeemaschine aus dem Computerlabor der Universität Cambridge an die Mitarbeiter und (ab November 1993) ins Internet sendete (Bild 2, nächste Seite). Der Blick auf den Füllstand ersparte damit so manchem Mitarbeiter den vergeblichen Weg zu ihr. Nachdem die Kaffeemaschine im Frühjahr 2001 wegen eines Defekts den Dienst quittierte, wurde sie für 3350 Pfund (damals 10452,72 DM) von *Spiegel Online* ersteigert (Seidler/Büchner, 2001). Anschließend wurde sie von der Herstellerfirma *Krups* kostenlos repariert und kann seitdem wieder durch zwei Webcams bestaunt werden (Bild 3, nächste Seite).

http://www.vaticanstate.va/DE/Monumente/webcam/index?cam=webcam1&testo=Petersplatz%20und%20die%20Stadt%20Rom

Bild 1: Blick auf Petersplatz und Rom (am 01.01.2010 um 11:31 Uhr).

Didaktisches Potenzial des Themas

Die Bilder einer Webcam selbst zu verarbeiten, kann ein interessantes Thema für den Informatikunterricht sein. Gemäß einer bekannten Definition von Kompetenz (Weinert, 2001), gehören zur Kompetenz auch die motivationalen und volitionalen Voraussetzungen, und die sollten im Unterricht beachtet werden. Weinert fordert insbesondere als Bestandteil einer Kompetenz den verantwortungsvollen Umgang mit den erworbenen Fähigkeiten. Mit trägem, nutzlosem Wissen lässt sich kaum etwas Unmoralisches tun, mit den hier im Unterricht erworbenen Fähigkeiten dagegen sehr wohl, beispielsweise Live-Bilder aufnehmen und verfremden. Aber gerade weil Missbrauch möglich ist, können die Schülerinnen und Schüler auch einen verantwortungsvollen Umgang mit ihren Fähigkeiten beweisen.

Bei der Aufnahme von Personen sind die Rechte am eigenen Bild sowie die Bestimmungen des Datenschut-

Quelle: LOG-IN-Archiv

Bild 2: Das historisch erste Webcam-Bild – eine Kaffeemaschine.

http://www.spiegel.de/static/popup/coffeecam/cam2.html

Bild 3: Die Kaffeemaschine heute (links).

zes zu beachten. Mit einer Webcam dürfen ohne Einwilligung der Betroffenen keine Bild- und Tonaufnahmen gemacht werden, auf denen Personen identifiziert werden können. Dies gilt auch für Merkmale, die eine Identifizierung ermöglichen. Webcams im öffentlichen Raum unterliegen diesen Bestimmungen genauso wie solche, die lediglich innerhalb von Unternehmen oder Institutionen betrieben werden (vgl. Müller, 2006).

Technisch geht es um den Umgang mit einem Datenstrom erheblicher Größe: Wollte man die nicht komprimierten Bilder einer Webcam fortlaufend speichern, hätte man nach etwa einer Minute einen heutzutage üblichen Arbeitsspeicher gefüllt. Schüler haben hier – was sonst recht selten ist – Überlegungen zum Haushalten mit Speicherplatz anzustellen. Zur Bildmanipulation kommen verschiedene Algorithmen zum Einsatz, und die Anwendung von ähnlichen Techniken in der professionellen Medienwelt weist dem Thema auch einen Platz im Inhaltsbereich *Informatik*, *Mensch und Gesellschaft* der Bildungsstandards (AKBSI, 2008) zu. Von den Prozessbereichen werden angesprochen: das Implementieren von Programmen, das Begründen der Verfahren und das Interpretieren der Ergebnisse, die oft auch unerwartet ausfallen. Sobald mehrere Bilder kombiniert werden, werden auch Modellierungen relevant, weil man sich entscheiden muss, ob man das Übereinander-Projizieren wie mit zwei Diaprojektoren modelliert, oder das Hintereinander-Halten von Dias.

Die folgenden Beispiele werden zeigen, dass es sehr einfach ist, eine Webcam in eigenen Programmen anzusprechen und ihre Daten zu verarbeiten. Damit ist aber die Frage, ob man das im Unterricht tun sollte, noch nicht beantwortet. Auch die obige Feststellung, dass das Thema sich an mehrere Prozess- und Inhaltskompetenzen der Bildungsstandards anknüpfen lässt, ist allein noch keine hinreichende Begründung. Die Beschäftigung mit Webcams hat aber in der Tat ein *didaktisches Potenzial*, das sich aus vielen Quellen speist:

- Webcams und auch ihre Geschwister als Überwachungskameras und Filmkameras sind allgegenwärtig. Eine Beschäftigung damit dient also der vertieften Erschließung der Lebenswelt.
- Das Thema *Echtzeitdatenverarbeitung* wird selten im Informatikunterricht angesprochen, ist aber in vielen Bereichen wesentlich.
- Effizienz und Speicherbedarf von Algorithmen sind bei den meisten im Informatikunterricht erstellten Programmen irrelevant. Beispielsweise werden selten Datenmengen sortiert, bei denen das Verfahren *Sortieren durch Austausch* (Bubblesort) untragbar lange Zeit benötigen würde. Im Kontext der Echtzeitverarbeitung werden aber Geschwindigkeit und Speicherbedarf wichtige Kriterien.
- Betreten von Neuland: Algorithmen der Bildverarbeitung sind auch im statischen Falle ein lohnendes Thema, aber viele Schülerinnen und Schüler kennen in Programmen wie *GIMP* oder *Photoshop* schon mehr Effekte, als man selbst programmieren kann. Der Unterricht leistet daher im Wesentlichen eine Rekonstruktion und ein Aufdecken der Grundlagen. Das ist auch lohnend, aber bei Webcams kann man wirklich Neuland betreten und Dinge machen, die noch in keinem „Programm von der Stange“ realisiert sind.
- Das Thema erfordert und stärkt die Kreativität.

Technische Möglichkeiten

Um Informatiksysteme mit Webcam-Nutzung zu gestalten, benötigt man wenig Funktionalität: Es muss eigentlich nur eine Methode oder Funktion bereitstehen, die das aktuelle Bild der Kamera (als Bildobjekt, zweidimensionale Reihung o.Ä.) liefert. Dazu gibt es mehrere Möglichkeiten. Für JAVA gibt es das Media-Framework (siehe Kasten „Webcam-Zugriff mit dem Media-Framework unter JAVA“, nächste Seite). Einfacher geht es unter PYTHON. Man installiert die

- ***PYTHON Imaging Library***
  (PIL; http://www.pythonware.com/products/pil/)
  und das Modul
- *VideoCapture*
  (http://videocapture.sourceforge.net/).

Die entscheidende Methode des Moduls ist getImage(), die das aktuelle Bild als PIL-Image liefert. Die PIL stellt viele Methoden bereit, um mit diesen Bildern zu arbeiten. Neben dem Zugriff auf einzelne Pixel gibt es auch eine Reihe von fortgeschrittenen Anweisungen zur Bildverarbeitung, etwa zur Schärfung. Um die Bilder darzustellen, empfiehlt sich eine weitere Bibliothek; wir verwenden hier

▷ *PyGame* (http://www.pygame.org/).

Unterrichtsbeispiele

Beispiel 1: Welche Farbe hat Blut?

Als Einstieg kann man die Farbkanäle ausmitteln lassen. Die Ausgabe ist dann eine schlichte Folge von drei Zahlen, nämlich die der mittleren Intensitäten. Damit lassen sich ein paar interessante Dinge demonstrieren: Zeigt man der Kamera etwa ein LOG-IN-Heft, geht der Rot-Anteil nach oben, hält man ihr ein Foto vom Himmel hin, steigt der Blauwert. Hält man die Kamera mit schwarzer Pappe zu, geht alles auf null. Mit dem Finger aber gehen nur Blau- und Grünkanal auf null, der Rotkanal nicht – woraus sich sofort die Frage ergibt, welche Farbe eigentlich Blut hat.

```
# Beispiel 1: Farbe ausmitteln

from VideoCapture import Device
import ImageDraw, sys,  time

(breite, hoehe) = (640, 480)
cam = Device()
cam.setResolution(breite, hoehe)

while 1:
  camshot = cam.getImage()
  (r, g, b)=(0, 0, 0)
  for x in range(0, breite):
    for y in range(0, hoehe):
      pixel = camshot.getpixel((x,y))
      r = r + pixel[0]
      g = g + pixel[1]
      b = b + pixel[2]
      r = r / (breite * hoehe)
      g = g / (breite * hoehe)
      b = b / (breite * hoehe)
      print "r = %3d, g = %3d, b = %3d" %(r, g, b)
```

Hier bieten sich einige physikalische Experimente an: Man beleuchtet die Kamera mit weißem (nicht zu hellem) Licht, dann bringt man absorbierende Substanzen in den Strahlengang.

Beispiel 2: Bildverarbeitung live

Auf die Pixel von PIL-Bildern kann man einzeln zugreifen – aber das ist recht langsam. Schneller gehen Manipulationen mit den eingebauten PIL-Methoden: Die Methode point wendet eine Funktion auf jedes Pixel an. Damit erhält man z. B. das eigene Bild als Negativ:

```
# Beispiel 2 : Live-Bild-Manipulation

from VideoCapture import Device
import ImageDraw, Image, sys, pygame, time
from pygame.locals import *

res = (breite, hoehe) = (640, 480)
pygame.init()
cam = Device()
cam.setResolution(breite,hoehe)
screen = pygame.display.set_mode(res)
pygame.display.set_caption('Webcam')

while 1:
  camshot = cam.getImage().convert("L")
      # Bild holen und in Graustufen konvertieren
  for event in pygame.event.get():
      # Ereignisverarbeitung
    if event.type == pygame.QUIT: sys.exit()
    keyinput = pygame.key.get_pressed()
    if keyinput[K_x]: break      # Beenden mit "x"
    neuBild = camshot.point(lambda p: 255-p)
      # Ins Negativ
    cs = pygame.image.frombuffer(
          neuBild.convert("RGB").tostring(),
             res, "RGB")
    screen.blit(cs, (0,0))
      # Bild in Grafikpuffer ...
    pygame.display.flip()  # ... und anzeigen
```

Durch Variation einer Zeile lassen sich andere Effekte erreichen:

```
neuBild = camshot.point(lambda p: p+50)
          # Bild heller
neuBild = camshot.point(lambda p: 50+p/2)
          # Kontrast geringer
```

Dank der Funktionalität von PIL (oder mit einer Schleife über alle Pixel) kann man sich auch live auf den Kopf stellen oder in vertikaler Richtung strecken.

Beispiel 3: Bildverarbeitung in Farbe

Im obigen Programm (Beispiel 2) wurden die Farbbilder der Kamera zwecks einfacherer Verarbeitung in Graustufen umgewandelt und für die Darstellung auf dem RGB-Bildschirm wieder zurückverwandelt. Das zeigt schon, dass sich Farbbilder ebenso verarbeiten lassen. Allerdings fehlt in PIL ein Analogon der *point*-Methode für Farbbilder, sodass man eine Schleife über alle Pixel braucht:

```
# Beispiel 3 (Fragment):
  Live-Bild-Manipulation in Farbe

while 1:
  camshot = cam.getImage()
  keyinput = pygame.key.get_pressed()
  if keyinput[K_x]: break
  for x in range(breite):
    for y in range(hoehe):
      p = camshot.getpixel((x,y))
      p = (p[2], p[1], p[0]) # Kanäle vertauschen
      camshot.putpixel((x,y), p)
      cs = pygame.image.
         frombuffer(camshot.tostring(), res, "RGB")
      # weiter wie in Beispiel 2
```

Beispiel 4: Verfolgung farbiger Objekte

Mit einer ähnlichen Schleife kann man beispielsweise rote Objekte finden. Wenn man annimmt, dass es ein interessantes rotes Objekt gibt (z.B. einen Ball, den man verfolgen will), dann sucht man nach dem Pixel, dessen Farbe am nächsten bei Rot (255, 0, 0) liegt. Dabei ist eine interessante Aufgabe zu bewältigen: „Was bedeutet es, nahe bei Rot zu sein?“ Man könnte den Euklidischen Abstand von (255, 0, 0) oder eine Regel wie im folgenden Programm nehmen. Die Folgen dieser Modellierungs-Entscheidung können sofort ausprobiert werden.

```
# Listing 4 (Fragment):
  Ein roter Ball wird verfolgt

speedup = 3     # Nur jedes dritte Pixel prüfen
opt = -1000     # Bisher gefundenes Optimum
pos = [0, 0]    # Position des bisher
                  "rotesten" Pixel
  for x in range(0, breite, speedup):
    for y in range(0, hoehe, speedup):
      p = camshot.getpixel((x,y))
      if p[0] - p[1]/2 - p[2]/2 > opt:
        opt = p[0] - p[1]/2 - p[2]/2
        pos = [x, y]
      draw = ImageDraw.Draw(camshot)
                # Zeichner für Bild
      draw.ellipse((pos[0]-10, pos[1]-10,
                    pos[0]+10, pos[1]+10),
                    fill = (255,0,0))
```

Diese Technik eröffnet die Möglichkeit verschiedener kleiner Projekte:

- *Messwerterfassung:* Wie fällt ein Ball (Zeiten messen), welche Flugbahn nimmt er?
- *Virtueller Stift:* Man speichert die Position des Balls in einer Liste und zeichnet so in jedem neuen Bild die Spur des Balls ein.
- *Virtueller Joystick:* Das Bild der Webcam wird gar nicht dargestellt, sondern es werden nur die Koordinaten des roten Punktes genommen, um eine Spielfigur zu steuern.

Beispiel 5: Erkennen von Bildänderungen

Bisher wurde immer das aktuelle Bild bearbeitet. Neue interessante Effekte ergeben sich, wenn man sich einige alte Bilder merkt. Die Differenz des aktuellen und eines früheren Bildes zeigt an, wo es Änderungen gab. Mit einer Schleife über alle Pixel eines Graustufenbildes könnte man diese Differenz direkt berechnen und zuweisen, z.B. mit

```
camshot.putpixel((x,y), 100 + alt.getpixel((x,y))
- neu.getpixel((x,y)).
```

Zwecks höherer Geschwindigkeit ist es aber wieder ratsam, die Methoden von PIL zu verwenden. Man konvertiert das alte Bild ins Negativ (aus alt wird –alt), dann addiert man neu und –alt mit der Methode blend zur Überlagerung zweier Bilder.

Für diese Anwendung reicht es, immer nur das letzte Bild zu speichern und mit dem neuen zu verrechnen. Es liegen aber auch Varianten nahe, bei denen man auf andere alte Bilder zugreifen will. Das Programm zu Beispiel 5 speichert deshalb gleich die letzten 10 Bil-

Webcam-Zugriff mit dem Media-Framework unter JAVA

Für JAVA steht das Media-Framework zur Verfügung. Im Gegensatz zu PYTHON empfiehlt sich seine Verwendung, wenn nicht schnelle Erfolgserlebnisse und eigenes Handeln der Schülerinnen und Schüler im Vordergrund stehen, sondern die systematische Beschäftigung mit einer großen, gut strukturierten Bibliothek.

Um ein Bild aufzunehmen, sind die entscheidenden Schritte die Erzeugung eines *FrameGrabbingControls*, der ein neues Bild einfangen kann. Dessen Daten verwandelt man in einen *Buffer*, der mittels einer weiteren Klasse in ein *Image* verwandelt wird, mit dem JAVA umgehen kann:

```
FrameGrabbingControl fgc =
  (FrameGrabbingControl)
  player.getControl("javax.media.
            control.FrameGrabbingControl");
Buffer buf = fgc.grabFrame();
BufferToImage btoi = new BufferToImage
            ((VideoFormat)buf.getFormat());
Image img = btoi.createImage(buf);
```

Da die Image-Klasse in JAVA von der Pixelstruktur abstrahiert, kann man keine Pixel auslesen oder verändern. Die Klasse PixelGrabber kann aber ein Image in eine (eindimensionale) Reihung verwandeln – und umgekehrt ist die Klasse MemoryImageSource hilfreich. Vom Image zur Reihung:

```
// img wie oben
int breite = img.getWidth(null);
int höhe = img.getHeight(null);
int[] pixelarray = new int[breite * höhe];
PixelGrabber grabber =
  new PixelGrabber(img, 0, 0, breite, hoehe,
                   pixelarray, 0, breite);
  try {grabber.grabPixels();}
  catch(InterruptedException e) {pixelarray =
                                  null;}

Danach kann man auf ein einzelnes Pixel
zugreifen und darin z. B. den Grün-Anteil
ermitteln.

int grün = (pixelarray[y * breite + x]
            & 0x0000ff00)/256;
```

Das Setzen der Pixel geht analog dazu bit-basiert. Die Rückverwandlung in ein Image, das man in AWT oder *Swing* in ein Fenster zeichnen kann, geht so:

```
MemoryImageSource ms =
  new MemoryImageSource(breite, höhe,
                        pixelarray, 0, breite);
Image im = createImage(ms);
  // Methode von Canvas
```

In der didaktischen Diskussion wurde oft darauf hingewiesen, dass die Wahl der Programmiersprache nebensächlich sei. Das vorliegende Beispiel zeigt allerdings, dass beim gleichen Thema die verwendete Sprache doch einen Einfluss auf die Unterrichtsinhalte hat: Durch JAVA rücken die in der Bibliothek vorgenommenen Modellierungen der verschiedenen Arten von Bildern deutlich in den Vordergrund.

der. Eine Begrenzung ist aber nötig, weil sonst sehr schnell der Speicher überläuft – durchaus eine interessante Erkenntnis, die man Schülerinnen und Schüler erst einmal machen lassen kann.

```
# Beispiel 5 (Fragment): Detektion von Änderungen

history = []
while 1:
  camshot = cam.getImage().convert("L")
  # Ereignisverarbeitung wie oben
  history.append(camshot)
  if len(history) > 10: del history[0]
  if len(history) > 1:
    alt = history[-2].point(lambda i: 255-i)
    neu = history[-1]
    camshot=Image.blend(alt, neu, 0.5)
  # Bild darstellen wie oben
```

Auch hier ergeben sich viele Anwendungen, beispielsweise Diebstahlssicherung oder Geschwindigkeitsmessung.

Astronomen suchen auf diese (oder ähnliche) Weise nach Supernova-Explosionen: Die erkennt man u.a. daran, dass sich relativ schnell etwas an der Helligkeit von Sternen ändert (siehe Bild 4). Interessant ist auch, statt dem aktuellen Bild, das zwanzigst-letzte zu zeigen: Dann sieht man seiner jüngsten Vergangenheit ins Auge; die Modifikationen dazu sind ganz leicht. Ebenso spaßig: Wenn man gerade das n-te Bild hereinbekommt, zeigt man das n/2-te Bild an.

Beispiel 6: Blue-Box

Dass in Film und Fernsehen Bilder montiert werden und z.B. Reporter vor einem weißen Haus zu stehen scheinen, die in Wirklichkeit im warmen Studio sind, ist gang und gäbe. In der Regel wird der Reporter dann vor einer blauen Wand aufgenommen und alles Blaue durch den geplanten Hintergrund ersetzt. In Ermangelung einer blauen Wand kann man auch mit Weiß arbeiten. Allerdings ist der Test, wann ein Pixel in diesem Sinne als weiß gelten sollte, nicht so einfach und muss den Beleuchtungsverhältnissen angepasst werden. Das Programm zu Beispiel 6 zeigt einen Vorschlag, den man – im Sinne von Modellbildung und Modellkritik – bewerten und noch verbessern kann.

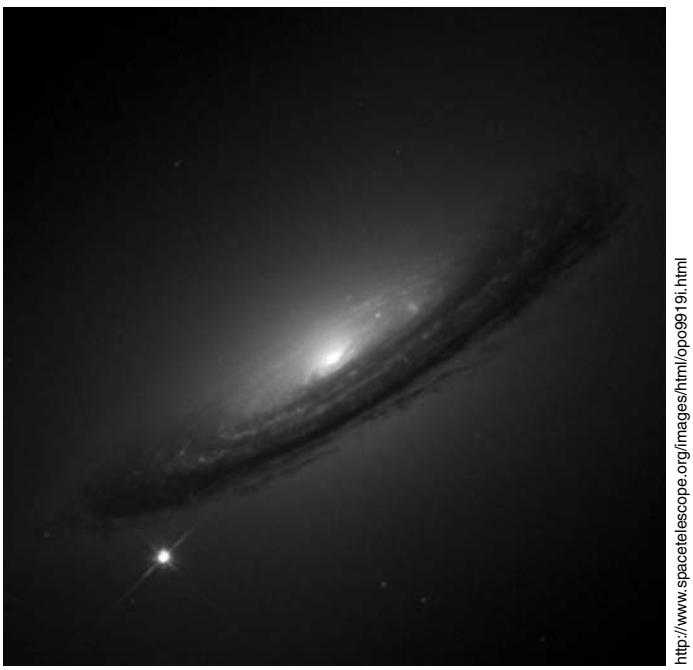
```
# Beispiel 6 (Fragment): White-Box statt Blue-Box

hintergrund = Image.open("wald.jpg")
while 1:
  camshot = cam.getImage()
  # Ereignisverarbeitung wie oben
  for x in range(breite):
    for y in range(hoehe):
      p = camshot.getpixel((x,y))
      s = 110 # Schwelle für weiß
      if p[0] > s and p[1] > s and p[2] > s and
         max(abs(p[0]-p[1]), abs(p[2]-p[1])) < 50:
        camshot.putpixel((x,y),
                hintergrund.getpixel((x,y)))
  # Darstellen wie oben
```

Und ab ins Netz

Bisher wurden die Bilder lokal erzeugt und verändert. Dem Namen Webcam wird man damit nicht ganz gerecht. Die einfachste Art, die Bilder ins Netz zu bringen besteht darin, sie als JPEG-Datei zu speichern und per FTP auf einen Webserver hochzuladen. In der Distribution von VideoCapture findet sich die Datei webcam-uploader.py, die eben das macht. In ihr wird ein Bild gelesen, gespeichert und automatisch hochgeladen. Dazwischen kann man sehr gut die in diesem Artikel beschriebenen Effekte programmieren. Flüssiger wird der Bildstrom, wenn wie in der Datei webcam-server.py aus VideoCapture ein eigener Server programmiert wird, sodass man ohne den Umweg über Dateien auskommt. Aber obwohl dieses Programm weniger als 100 Zeilen umfasst, lässt es sich im Unterricht nur gut behandeln, wenn man tief in HTTP einsteigen möchte.

Fazit

Die Bearbeitung beweglicher Bilder ist ein Thema, das keine Motivationsprobleme kennt. Die nötige Programmiererfahrung beschränkt sich auf bescheidene Algorithmik. Fast könnte man versucht sein zu sagen, die Dinge seien für die Schülerinnen und Schüler sogar zu leicht. Aber dies trifft nicht zu. Ähnlich wie bei der Programmierung von Robotern liegt Wesentliches in den Details der Planung der Abläufe von Auswertung, Änderung und Darstellung. Nicht zuletzt schätze ich an diesem Thema, dass es Möglichkeiten für die eigenständige Beschäftigung mit dem Thema bietet; denkbar



http://www.spacetelescope.org/images/html/opo9919i.html

Bild 4: Hubble-Space-Telescope-Bild der Supernova 1994D (SN 1994D) am Rand der Galaxie NGC 4526 (SN 1994D ist der helle Fleck unten links). Die Entfernung von der Erde beträgt rund $55 \cdot 10^6$ Lichtjahre.

sind etwa *Jugend-forscht*-Projekte zur Untersuchung von Bewegungsvorgängen, zum Wachstum von Pilzen, zum eigen Schlafverhalten (z. B. jede Minute ein Bild) usw. Setzt man die Webcam auf einen Roboter, ergeben sich weitere Möglichkeiten.

Prof. Dr. Reinhard Oldenburg
Institut für Didaktik der Mathematik und Informatik
Senckenberganlage 9
60325 Frankfurt

E-Mail: oldenbur@math.uni-frankfurt.de

Literatur und Internetquellen

AKBSI – Arbeitskreis „Bildungsstandards“ der Gesellschaft für Informatik (Hrsg.): Grundsätze und Standards für die Informatik in der Schule – Bildungsstandards Informatik für die Sekundarstufe I. Empfehlungen der Gesellschaft für Informatik e. V. vom 24. Januar 2008. In: LOG IN, 28. Jg. (2008), Heft 150/151, Beilage.

Müller, J.: Legales Hacking. In: LOG IN, 26. Jg. (2006), Heft 140, S. 60–68.

Seidler, Chr.; Büchner, W.: SPIEGEL ONLINE rettet die Trojan-Room-Kaffeemaschine (11.08.2001).
http://www.spiegel.de/netzwelt/web/0,1518,148112,00.html
[zuletzt geprüft am 4. Januar 2010]

Stafford-Fraser, Qu.: The Life and Times of the First Web Cam. In: Communications of the ACM, 44. Jg. (2001), Nr. 7, S. 25–26.
http://www.cl.cam.ac.uk/coffee/qsf/cacm200107.html
[zuletzt geprüft am 4. Januar 2010]

Weinert, F. E.: Vergleichende Leistungsmessung in Schulen – eine umstrittene Selbstverständlichkeit. In: Weinert, F. E. (Hrsg.): Leistungsmessungen in Schulen. Weinheim: Beltz, 2001, S. 17–31.