Echtzeitbildverarbeitung SoSe 2015
Prof. Dr.: Udo Frese

Annika Ofenloch - 2992807 - ofenloch@uni-bremen.de
Frank Ihle - 3010158 - fihle@uni-bremen.de
Simon Schirrmacher - 4000884 - simons@informatik.uni-bremen
Noshaba Cheema - ncheema@uni-bremen.de

Übungsblatt

Abgabe am: 05.06.15

Aufgabe 1: Ballspiele II

Listing 1: Computes the perspective projection (in pixel) of \c p

```
int CameraCalibration::project (double& x, double& y, const VVector& pInWorld) const
{
  // TODO: implement (1P)
  Transform worldInCamera;
  inverse(worldInCamera, cameraInWorld);
  VVector pInCamera = worldInCamera * pInWorld;

  // avoid division by zero
  if (pInCamera[2] == 0) return 0;

  x = centerX + alpha * pInCamera[0] / pInCamera[2];
  y = centerY + alpha * pInCamera[1] / pInCamera[2];

  // check if behind camera
  if (pInCamera[2] < 0) return 0;
  // check if inside image
  return isInside(x, y) + 1;
}
```

Listing 2: Project a sphere of \c radius at \c p into the image

```
int CameraCalibration::project (double& x, double& y, double& r, const VVector& pInWorld, double
    radius) const
{
  // TODO: implement (1P)
  Transform worldInCamera;
  inverse(worldInCamera, cameraInWorld);
  VVector pInCamera = worldInCamera * pInWorld;

  // avoid division by zero
  if (pInCamera[2] == 0) return 0;

  x = centerX + alpha * pInCamera[0] / pInCamera[2];
  y = centerY + alpha * pInCamera[1] / pInCamera[2];
  r = alpha * radius / pInCamera[2];

  // check if behind camera
  if (pInCamera[2] < 0) return 0;
  // check if inside image
  return isInside(x, y) + 1;
}
```

Listing 3: Compute the world position of an image sphere

```
bool CameraCalibration::generate (VVector& pInWorld, double x, double y, double r, double radius)
    const
{
  // TODO: implement (1P)
  if(r <= 0) return false;

  VVector pInCamera;
  pInCamera[3] = 1;
  pInCamera[2] = alpha * radius / r;
  pInCamera[1] = (y - centerY) * pInCamera[2] / alpha;
  pInCamera[0] = (x - centerX) * pInCamera[2] / alpha;

  pInWorld = cameraInWorld * pInCamera;

  // check if input of generate is equal to output of project
  double xImage, yImage, rImage;
  project(xImage, yImage, rImage, pInWorld, radius);
```

```
17   assert("project(generate(INPUT)) must be equal to INPUT!" &&
18     fabs(x - xImage) + fabs(y - yImage) + fabs(r - rImage) < 0.1);
19   return true;
20 }
```

Listing 4: The parameters used by this algorithm

```
1 ParticleFilter::Parameters::Parameters ()
2 // TODO: choose good parameter (1P)
3 :ballRadius(0.1), sigmaVelocity(0.5), sigmaImage(2), deltaT(1.0 / 25), waitUntilSecondObservation(3.0
    / 25), nrOfParticles(200)
4 {
5   g[0] = 0;
6   g[1] = 0;
7   g[2] = -9.81;
8   g[3] = 0;
9 }
```

Listing 5: Integrate an observation of the ball during initialization

```
1 void ParticleFilter::Particle::observeInit (double x, double y, double r)
2 {
3   // TODO: implement (2P)
4   double sigma = filter->param.sigmaImage;
5   VVector p;
6   // add noise
7   x += sigma * randomGaussian();
8   y += sigma * randomGaussian();
9   r += sigma * randomGaussian();
10  if (filter->camera.generate(p, x, y, r, filter->param.ballRadius)) {
11    if (state == POSITIONDEFINED) {
12      double dT = time - timeOfLastObservation;
13      // determine velocity after waitUntilSecondObservation has passed
14      if (dT > filter->param.waitUntilSecondObservation) {
15        // acceleration added to initial velocity for analytical approach
16        velocity = (p - position) / dT + filter->param.g * dT * .5;
17        position = p;
18        state = FULLDEFINED;
19        timeOfLastObservation = time;
20      }
21    }
22    else {
23      // determine initial position of particle
24      position = p;
25      state = POSITIONDEFINED;
26      timeOfLastObservation = time;
27    }
28  }
29  // for a better result, particles outside of field of view can be already winthdrawn in the
      initialization
30  else weight = 0;
31  if (position[2] < 0) weight = 0;
32 }
```

Listing 6: Integrate an observation of the ball once the particle is fully initialized

```
1 void ParticleFilter::Particle::observeRegular (double x, double y, double r)
2 {
3   // TODO: implement (1P)
4   double dx, dy, dr, sigma = filter->param.sigmaImage;
5   // ignore particle if behind camera
6   if (!filter->camera.project(dx, dy, dr, position, filter->param.ballRadius))
7     weight = 0;
8   else {
9     // weight particle
10    dx = x - dx;
11    dy = y - dy;
12    dr = r - dr;
13    weight *= exp(-(dx*dx + dy*dy + dr*dr) / (2 * sigma * sigma));
14  }
15  timeOfLastObservation = time;
16 }
```

Listing 7: Time \c deltaT has passed. Proceed to the next image and update the particles accordingly.

```
1 void ParticleFilter::dynamic ()
2 {
3   // TODO: implement (included with Particle::dynamic)
4   for (int i = 0; i < particle.size(); ++i)
5     particle[i].dynamic(param.deltaT);
6 }
7
```

```
8  void ParticleFilter::Particle::dynamic (double deltaT)
9  {
10   // TODO: implement (2P)
11   if (state == FULLDEFINED) {
12     VVector n = { randomGaussian(), randomGaussian(), randomGaussian(), 0};
13     // analytical approach
14     position += deltaT * velocity + deltaT * deltaT * filter->param.g * .5;
15     velocity += deltaT * filter->param.g + filter->param.sigmaVelocity * sqrt(deltaT) * n;
16   }
17   time += deltaT;
18 }
```

Listing 8: A ball has been seen with image \c x y and radius \c r.

```
1  void ParticleFilter::observe (double x, double y, double r)
2  {
3    // TODO: implement (included with Particle::observe)
4    for (int i = 0; i < particle.size(); ++i)
5      particle[i].observe(x, y, r);
6  }
```

Listing 9: Initialize the particle filter with \c nrOfParticles particles

```
1  void ParticleFilter::createSamples (int nrOfParticles)
2  {
3    // TODO: implement (included with ParticleFilter::Parameters::Parameters)
4    double weight = 1.0 / nrOfParticles;
5    for (int i = 0; i < nrOfParticles; ++i)
6      particle.push_back(Particle(this, weight));
7  }
```

Listing 10: Draw particles from the particle set resetting their weight to 1.

```
1  void ParticleFilter::resample (int nrOfParticles)
2  {
3    // TODO: implement (1P)
4    // 'j' and 'weightUpToJ' just for solution in lecture
5    double totalWeight = 0, weightUpToJ = 0;
6    int j = -1;
7    vector<Particle> pNew;
8
9    // calculate total weight
10   for (int i = 0; i < particle.size(); ++i)
11     totalWeight += particle[i].weight;
12
13   double normWeight = totalWeight / nrOfParticles, unitWeight = 1.0 / nrOfParticles;
14   // initial "wheel of fortune" pointer
15   double weightChosen = randomUniform() * normWeight;
16
17   for (int i = 0; i < nrOfParticles; i++) {
18
19     // solution from lecture creates an infinite loop because the Gaussian distribution becomes 0 at
       some points
20     // where the particle is too far from the actual cirle (numerical underflow)
21
22     /*
23       while (weightChosen >= weightUpToJ) {
24         j++;
25         weightUpToJ += particle[i].weight;
26       }
27       pNew.push_back(particle[j]);
28       pNew.back().weight = unitWeight;
29       weightChosen += normWeight;
30     */
31
32     // current "wheel of fortune" pointer
33     weightChosen -= particle[i].weight;
34     // add normWeight to the current pointer until the wanted wight is archieved
35     while (weightChosen <= 0) {
36       pNew.push_back(particle[i]);
37       pNew.back().weight = unitWeight;
38       // next "wheel of fortune" pointer
39       weightChosen += normWeight;
40     }
41   }
42   // make sure sizes are equal
43   assert("Resampled particles must have the size of nrOfParticles" && pNew.size() == nrOfParticles);
44   particle = pNew;
45 }
```

Auf einem Rechner mit einem Intel Pentium 1.4GHz Prozessor, 4GB RAM, zwei Kernen und keiner externen GPU benötigt das Programm im Durchschnitt pro Bild 200ms. Wenn man die Framerate auf 25 FPS schätzt, muss ein Bild in $\frac{1}{25}$s = 0.04 s = 40 ms abgehandelt werden, damit man auf 25 Bilder pro Sekunde kommt und das Programm in Echtzeit abläuft. Dies ist leider auf diesem Rechner nicht der Fall. Auch bei einer Schätzung von 24 FPS würde das Programm nicht in Echtzet ablaufen, da es hierfür ein Bild in 41.67 ms abhandeln müsste. Auf dem Rechner benötigt das Programm für 25 Frames ca. 5 Sekunden.

Aufgabe 2: Freistoß (4 Punkte)

Bei einer Fußball-TV-Übertragung soll für den Zuschauer die Zone in das Bild eingezeichnet werden, auf der sich kein Gegenspieler bei einem Freistoß befinden darf.

a Allgemein

Der Algorithmus orientiert sich an bekannten Objekten, deren Dimensionen ihm im physikalischen Raum bekannt sind. Die meisten Bemaßungen auf einem Fußballfeld sind festgeschrieben und falls es hiervon Abweichungen gibt, können diese vor Spielbeginn eingeholt und händisch eingetragen werden.

Das Fußballfeld ist rechteckig, aber je nach Blickwinkel der Kamera ergeben sich Bildverzerrungen (beispielsweise erscheint der Platz, wenn die Kamera flach, von der Seite sowie mittig auf das Spielfeld schaut, trapezförmig). Diese Verzerrung muss ebenso beim Kreis berücksichtigt werden. So wird aus dem Kreis eine Ellipse, deren kurze Diagonale umso größer wird, je näher die Ellipse an der Kamera ist. Diese Verzerrung kann durch die Kameragleichung ermittelt werden (siehe Gleichung 1)

Wenn nun bekannt ist, mit welchen Parametern die Ellipse eingezeichnet werden soll, gibt es hierfür zwei Möglichkeiten, diese um den Ball, als Mittelpunkt zu zeichnen. Zum einen kann statisch festlegen werden, dass das Kamerabild genau an einer Stelle stehen bleibt und erst beim Anspielen wieder geschwenkt werden darf, da sich ab hier der Kreis wieder auflösen soll - so muss die Zone nur einmal berechnet und eingetragen werden. Für die nachfolgenden Bilder kann diese dann einfach rein kopiert werden.

Die einzelnen Punkte des Kreises werden mithilfe der Kameragleichung eingezeichnet. Dafür müssen natürlich Parameter wie Verzerrung der Kamera bekannt sein. Für die Kameragleichung wird auch eine Transformationsmatrix benötigt. Näheres zur Berechnung siehe Abschnitt c. Als Skalenmaßstab kann eine Größe einer der Skalierobjekte genommen werden, da diese eine genormete Größe haben.

Die andere Möglichkeit besteht darin, den Freistoß-Kreis beim Zoom-Vorgang zu zeigen. Hierfür werden mindestens zwei Bilder benötigt, auf denen Objekte zu sehen sind, deren reale Größe bekannt sind (siehe b Skaliergrößen). Im Mittelpunkt der Bilder ist der Ball und es werden wie gewohnt die Ellipsen berechnet. Da sich aber beim Zoomen die Zone im Folgebild vergrößert, kann nun ein $\triangle\ r$ berechnet werden und für die nächsten Aufnahmen (vor allem für diese, wo kein Objekt mit bekannter Größe zu erkennen ist) der Radius und dadurch die Ellipse definiert werden. Da dies einer linearen Zoom-Funktion entspricht, funktioniert dieses Verfahren ausschließlich bei konstanter Vergrößerungsgeschwindigkeit. Wenn mehr als zwei Aufnahmen zur Berechnung von $\triangle\ r$ verwendet und deren Ergebnis gemittelt werden, lässt sich dieser Faktor genauer einstellen.

Optional kann der Kreis farblich markiert werden, wenn ein Gegenspieler sich innerhalb des Kreises befindet. Hierfür tippt der Kameramann die Personen an, die sich innerhalb der Zone befinden dürfen. Anschließend wird deren Positionen immer mit dem vorhergehenden Bild verglichen, um eine Falschmeldung auszuschließen, wenn die Signalfarbe gesetzt werden soll. Gegenspieler im Kreis werden über deren Position und farbliche Unterschiede zum jeweiligen Boden erkannt.

Um Verwechslungen und Fehlausstrahlung zu vermeiden, soll der Kameramann zum einen, einen Bereich um den Kreis angeben, mit welcher der Algorithmus über Hough zur Ballposition kommt. Zum

anderen soll der auszustrahlende Kreis nochmal von einer Person bestätigt werden, ob dieser ungefähr in Position und Größe richtig ermittelt wurde.

b Skalierobjekte

Nachfolgend, Beispiele für Objekte mit bekannten Ausmaßen, mit deren Hilfe letztlich die Freistoßzone definiert werden kann: Torlatte und -pfosten, Rasenmuster sowie die weiße Linien (darunter: Seiten- und Mittellinien, Strafraum und Anstoßkreis).

Damit der Algorithmus weiß, um welches Skalierobjekt es sich gerade handelt soll entweder der Kameramann angeben, dass im Blickpunkt sich z.B.: das Tor befindet. Oder es muss noch ein automatischer Klassifizierungsalgorithmus hinzugefügt werden, welcher z.B.: den Mittelkreis über dessen Form, oder den Strafraum über seine rechteckige Form vor dem Tor erkennt.

c Kameraeinstellung

Ein Bildpunkt im Weltkoordinatensystem kann auf das Kamerasystem mit nachfolgender Gleichung umgerechnet werden:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \alpha \cdot d_{\kappa 1,\kappa 2}(p(T^{-1}_{W \leftarrow C} \cdot p^W)) \tag{1}$$

Die Transformationsmatrix beinhaltet nur die Verschiebung des Balls zum Ursprung und die Rotation um die Z- und X- Achse, die die Perspektive, Schwenken und das Nicken ausgleichen können. Da die Kamera aber nicht gieren kann, wäre eine Rotation um die Y-Achse nur nötig, um eine Kamerafehleinstellung auszugleichen.

Hier soll nun α_r die Verdrehung um die Z-Achse veranschaulichen. α_r ist der Winkel zwischen dem ursprünglichen und dem, um eine Achse verdrehtes Koordinatenkreuz. Dieser ist variabel und nähert sich immer mehr $\frac{\pi}{2}$, je mehr er sich der Kamera bzw. der blauen Linie nähert. Repräsentativ für den anderen Winkel soll die Winkelberechnung anhand der Beziehung für α_r erläutert werden, die in Abb. 1 veranschaulicht ist.

Die blaue Strecke teilt das Feld in zwei Bereiche auf, sie ist gleichzeitig die Spiegelachse für α_r und orientiert sich an der Kameraposition (hier ist $x = 0$). So haben Strecken, die parallel zur Torlinie verlaufen, aus dieser Perspektive in der linken Hälfte eine positive Steigung und in der rechten eine negative Steigung. Aus diesem Grund muss hier auch eine Fallunterscheidung gemacht werden, sodass α_r immer über das Winkel/Positionsverhältnis berechnet wird, wie in Gleichung 2 beschrieben. Eine Ähnliche Berechnung muss für den Verkippungswinkel (um X) des Koordinatenkreuzes durchgeführt werden. β wird dann ähnlich wie α über das Verhältnis von min. und max. Ballposition in Z Richtung ermittelt

$$\alpha_r = \begin{cases} [\frac{\pi}{2} - (\frac{\pi}{2} - \alpha_{1,min})\frac{b'}{x}] & wenn\ x \geq 0 \\ [\frac{\pi}{2} - (\frac{\pi}{2} - \alpha_{2,min})\frac{b'-b}{x}] & wenn\ x \leq 0 \end{cases} \tag{2}$$

d Beurteilung

Da beim Fußball schon wenige cm entscheidend sind, sollte der einzusetzende Algorithmus genaue Ergebnisse liefern, da „Fehlentscheidungen“ ansonsten den Zuschauer verärgern können. Die unter diesen Bedingungen, gezeigten Methoden haben vermutlich nur eine begrenzte Genauigkeit, je nachdem wie exakt ein Objekt mit bekannter Größe erfasst wird (Negativbeispiel: Auflösung des Fußballs bei Aufnahme aus der Entfernung).

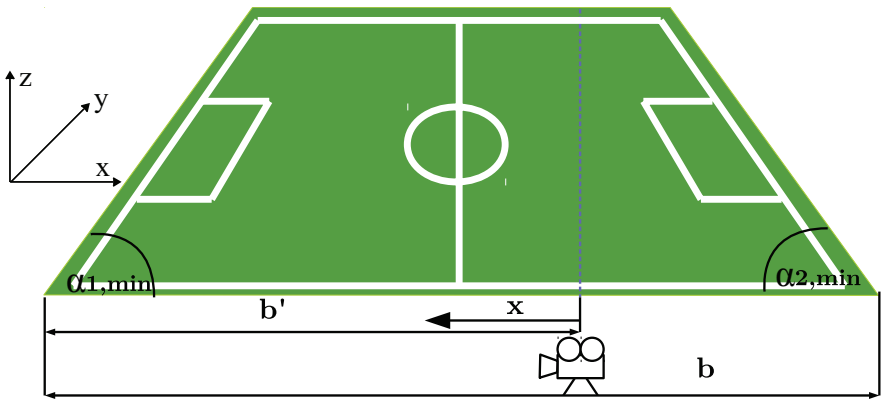

Abbildung 1: Veranschaulichung für die α_r-Berechnung

Aufgabe 3: Der einäugige Ballfangkönig (1 Punkt)

Um den Skalenmaßstab aus den Messungen zu bestimmen, soll eine Analyse der Situation durchgeführt werden. Zum einen lässt sich der Skalenmaßstab über die Tiefenschärfe und zum anderen über die Flugbahn des Balls ermitteln. Bei der Flugbahn werden zwei Ansätze betrachtet. Im Folgenden werden die drei Verfahren kurz erläutert.

a Verfahren mit Hilfe der Tiefenschärfe

Die Brennweite bestimmt den Abstand Linsenhauptebene und ihrem Brennpunkt. Ist nun also diese Brennweite (Abstand in z.B.: Metern) bei einer Kamera gegeben, so kennt man den Abstand eines Objekts zur Linse, wenn dieses den höchsten Schärfegrad erreicht hat. Dies kann beispielhaft bei unserem Ballwurfszenario implementiert werden. Hier wird einfach für jedes Bild die Kantenbreite über Sobel berechnet und das Bild, mit der der Ball die größten Sobellänge erreicht, ist dann dem Fokus der Kameralinse am nächsten. Die Genauigkeit der Methode hängt zum einen von der Genauigkeit der Brennweitenangabe und zum anderen von der Möglichkeit, den Ball genau im Brennpunkt erfassen zu können, ab.

Der Balldurchmesser kann demnach mit Hilfe der internen Kamerabemaßungen und dem Strahlensatz ermitteln werden. Wenn $\overline{AB}$ die Breite des Fotochips, $\overline{A'B'}$ die Breite des sich auf der Linse befindenden Bilds, $\overline{A''B''}$ der Balldurchmesser, d_1 der Abstand vom Fotochip zur Linsenhauptebene und d_2 die Brennweite ist, kann $\overline{A''B''}$ wie folgt berechnet werden (siehe. Abb 2):

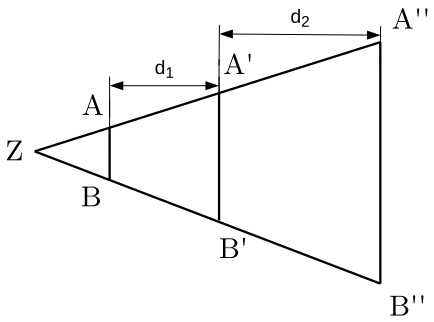

Abbildung 2: Veranschaulichung für die Balldurchmesserberechnung mit Hilfe der Tiefenschärfe.

$$\frac{\overline{AB}}{\overline{A'B'}} = \frac{\overline{ZA}}{\overline{ZA'}} = \frac{\overline{ZA}}{\overline{ZA}+\overline{AA'}}$$

$$(\overline{ZA} + \overline{AA'})\frac{\overline{AB}}{\overline{A'B'}} - \overline{ZA} = 0$$

$$\overline{ZA} = \frac{\overline{AA'}\frac{AB}{A'B'}}{1-\frac{AB}{A'B'}}$$

mit

$$\overline{A'A} = \sqrt{d_1^2 + (\frac{\overline{A'B'}}{2} - \frac{\overline{AB}}{2})^2}$$

dann erhält man über

$$\frac{\overline{AB}}{\overline{A''B''}} = \frac{\overline{ZA}}{\overline{ZA''}} = \frac{\overline{ZA}}{\overline{ZA}+\overline{AA'}+\overline{A'A''}}$$

mit

$$\overline{A'A''} = \sqrt{d_2^2 + (\frac{\overline{A''B''}}{2} - \frac{\overline{A'B'}}{2})^2}$$

das gesuchte $\overline{A''B''}$ mit folgender Berechnung:

$$\overline{A''B''} = \frac{\overline{ZA}+\overline{AA'}+\overline{A'A''}}{\overline{ZA}}\overline{AB}$$

b Verfahren mit Hilfe der Ballflugbahn

Erster Ansatz

Des Weiteren lässt sich der Skalenmaßstab über die Flugbahn des Balls berechnen, welche aus den vorhandenen Messungen hervorgeht. Es handelt sich hierbei um einen schrägen Wurf. Die Aufnahmezeit der einzelnen Frames ist bekannt, sodass über den Wurfwinkel α die Anfangsgeschwindigkeit v_0 ermittelt werden kann, wenn angenommen wird, dass die Ballposition auf dem ersten Frame den Start repräsentiert. v_0 setzt sich aus den Komponenten v_x und v_y zusammen und ist Abhängigkeit vom Abwurfwinkel α (siehe Abbildung 3). v_y stellt hierbei die Geschwindigkeit in y-Richtung und v_x die Geschwindigkeit in x-Richtung dar und lassen sich über die folgenden Gleichungen ausdrücken:

$$v_x = v_0 * cos(\alpha)$$
$$v_y = v_0 * sin(\alpha)$$

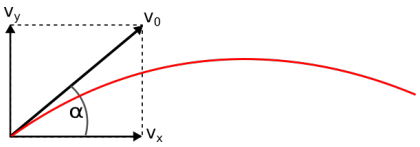

Abbildung 3: Geschwindigkeitsvektoren v_0, v_x sowie v_y

Mittels des Geschwindigkeit-Zeit-Gesetzes, wo ebenso die Erdbeschleunigung ($g \approx 9.81 m/s^2$) und Steigzeit t_H einbezogen wird, lässt sich die Geschwindigkeit v_y bei einem schrägen Wurf wie folgt berechnen:

$$v_y = v_0 * sin(\alpha) - g * t_H$$

Da v_y am höchsten Punkt der Bahnkurve gleich 0 ist, lässt sich die Gleichung nach v_0 umstellen. Bei t_H handelt es sich um die vergangene Zeit vom ersten Bild (Start bei t_1) bis zum Bild mit dem höchsten Punkt der Bahnkurve (Maximum bei t_2), wie in Abbildung 4 zu sehen ist. Da alle Parameter bekannt sind, kann die Anfangsgeschwindigkeit v_0 mit der Gleichung

$$v_0 = \frac{g * t_H}{sin(\alpha)}$$

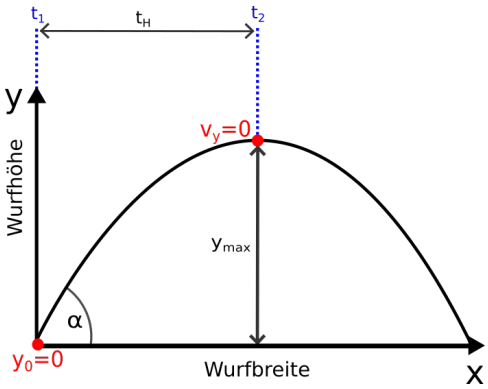

Abbildung 4: Ballflugbahn

berechnet werden. Anschließend kann die maximale Wurfhöhe y_{max} unter Verwendung des Ort-Zeit-Gesetz über die nachfolgende Gleichung ermittelt werden, wobei y_0 gleich 0 ist:

$$y_{max} = y_0 + v_0 * sin(\alpha) * t_H - g * (t_H)^2$$

Wenn nun die Länge der maximalen Wurfhöhe, wie in Abbildung 4 zu sehen ist, bekannt ist, verfügt man über einen Skalenmaßstab mit dem zum Beispiel der Balldurchmesser bestimmt werden kann. Es handelt sich hierbei jedoch nur um eine Approximation, da die z-Richtung des Wurfes nicht mit einbezogen wurde.

Für ein genaueres Ergebnis könnte die Tiefeninformation über den zuvor beschriebenen Tiefenschärfe-Ansatz einbezogen werden, sodass beide Verfahren kombiniert werden.

Zweiter Ansatz

Das zweite Verfahren zur Bestimmung des Skalenmaßstabs über die Ballflugbahn macht sich das Unabhängigkeitsgesetz zu nutze. Der Ball bewegt sich von der maximalen Wurfhöhe zum Zeitpunkt t_2 aus kurvenförmig zum Boden hin. Wenn ein weiterer Ball von der maximalen Höhe zum Zeitpunkt t_2 im freien Fall nach unten fallen würde, würden beide Bälle den Boden zum selben Zeitpunkt erreichen. Dieses Phänomen machen wir uns zu nutze, indem wir die Strecke h für den freien Fall ermitteln, also die Entfernung von der maximalen Wurfhöhe bis zum Aufprall (siehe Abbildung 5). Diese Strecke h wird mittels folgender Gleichung berechnet:

$$h = \frac{g * (t_H)^2}{2}$$

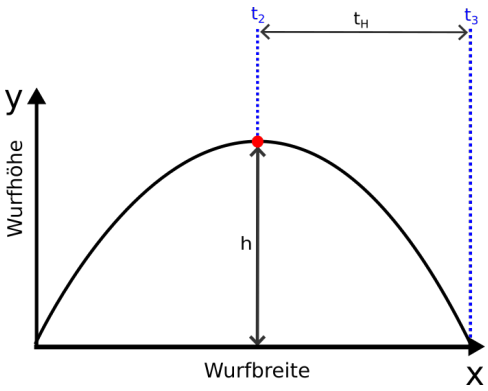

Abbildung 5: Ballflugbahn

t_H ist in diesem Fall die Fallzeit von t_2 bis zum Aufprall auf dem Boden. Aufgrund des Unabhängigkeitsgesetzes ist die Fallzeit über die Bildaufnahmen bekannt. Diese Strecke h stellt ebenso einen Skalenmaßstab

im Bild dar, wodurch unter anderem der Balldurchmesser ermittelt werden kann. Auch bei diesem Verfahren handelt es sich nur um eine Approximation, da die z-Richtung des Wurfes nicht mit einbezogen wird.

Hinzu kommt, dass bei den Berechnungen angenommen wurde, dass kein Luftwiderstand vorhanden ist (Annahme erfolgte ebenso bei der Programmieraufgabe).