Projektdokumentation *Messgerät*

HAW Hamburg // DMI // Media Systems

Mobile Systeme // Sommersemester 2014

Betreuender Dozent: Prof. Dr. Andreas Plaß

‘EGOmeter’

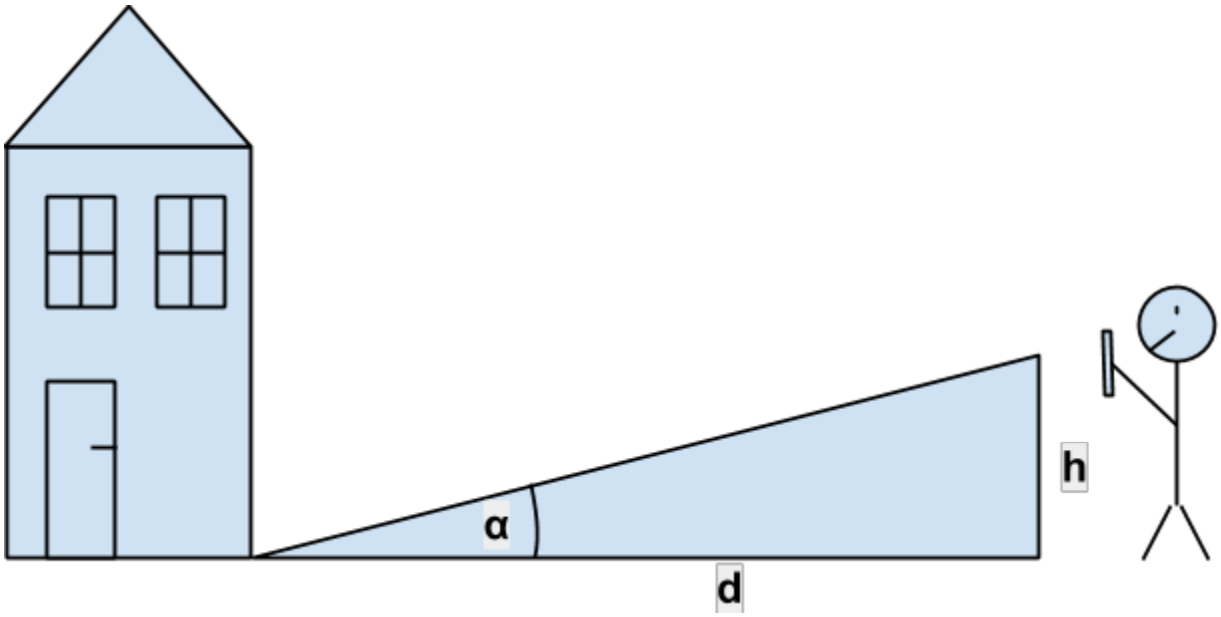
Timo Luttmann // Marko Vukadinovic

Inhaltsverzeichnis

Konzept

Idee

Nachdem wir uns in der ersten Gruppenübung im Kurs Mobile Systeme schon mit den Bewegungssensoren, also dem Gyroskop und dem Accelerometer beschäftigt hatten, war für uns relativ schnell klar, welche Sensoren wir für unsere App benutzen möchten. Zunächst wollten wir dann ein Spiel entwickeln, bei dem man eine Kugel durch ein Labyrinth bewegt und zur Steuerung die ausgelesenen Werte des Gyroskops verwendet. Diese Idee konnten wir jedoch nicht umsetzen, da wir laut Vorgabe ein App entwickeln sollten, “die eine physikalische Größe misst und das Ergebnis darstellt “ - was bei einem Spiel natürlich nicht der Fall ist.
Wir begannen also mit unseren Überlegungen von vorne, wollten aber weiterhin die Bewegungssensoren als Grundlage für die physikalische Größe nutzen. Es dauerte nicht lange, bis wir uns an den Geometrieunterricht in der Schule und die Winkelfunktionen Sinus, Kosinus und Tangens erinnerten. Wir überlegten eine Weile, was wir mit diesen Funktionen und den aus dem Gyroskop ausgelesenen Werten berechnen können, kamen aber auf kaum brauchbare Ergebnisse. Als Marko dann in einer alltäglichen Situation die Distanz zwischen zwei Gegenständen mit einem Zollstock (Meterstab) messen wollte, fiel ihm auf, dass man für diese Aufgabe ja auch eine App entwickeln könnte, die auf das Gyroskop und einen fixen Wert als Gegenkathete zugreift. Eine Skizze verdeutlicht diese Überlegung:



In der obigen Skizze ist *d* die Distanz zwischen User und einem beliebigen Objekt (Ankathete), *h* stellt die Größe des User da (Gegenkathete) und α ist der Winkel zwischen Hypotenuse und Gegenkathete. Es entsteht also ein rechtwinkliges Dreieck zwischen dem Smartphone, den Füßen des Users und dem Punkt, an dem das angepeilte Objekt den Boden berührt.

Mathematische Grundlage

Als mathematische Grundlage für die Berechnung der Entfernung zwischen dem User und einem beliebigen Objekt dient uns also folgende Gleichung:

$$\tan\alpha = \frac{Gegenkathete}{Ankathete} = \frac{a}{b} = \frac{1}{\cot\alpha}$$

Wir formen diese Gleichung nun wie folgt um:

tan (α) * Gegenkathete = Ankathete

Mit anderen Worten: Wenn der User durch die Kamera seines Smartphones einen Gegenstand anpeilt (bzw. den Punkt, an dem das Objekt den Boden berührt), muss er das Smartphone in einem bestimmten Winkel halten. Das Produkt aus dem Tangens dieses Winkels und der Distanz zwischen Smartphone und Boden (bzw. Füße des Users, Gegenkathete) ist die Distanz zwischen dem Punkt auf dem Boden über dem sich das Smartphone befindet und dem angepeilten Objekt (Ankathete).

Dabei kann die Höhe des Smartphones nicht von den im iPhone verbauten Sensoren ermittelt werden. Das verbaute Altimeter kann lediglich die Höhe über Normalnull berechnen, was in den meisten Fällen jedoch nicht der Höhe über dem Boden, auf dem der User steht, entspricht. Somit muss die Höhe des Smartphones vom User eingegeben werden. Der Winkel α hingegen wird für die Berechnung aus dem Gyroskop ausgelesen.

Einschränkungen

Man überlegt sich leicht, dass die angewandten mathematischen Grundlagen nur bei einem rechtwinkligen Dreieck zu einem brauchbaren Ergebnis führen. Sollte der User also auf einem Hügel stehen und ein Objekt unterhalb dieses Hügels anpeilen, wäre das Ergebnis unbrauchbar. Gleiches gilt für den Fall, in dem sich User und angepeiltes Objekt auf einer Schräge befinden.

Darüber hinaus muss der User die Höhe des Smartphones wissen, welche in den meisten Fällen kaum bekannt sein dürfte. Wenn er die eigene Körpergröße als Gegenkathete eingibt, ist das Ergebnis unbrauchbar, solange er das Smartphone nicht auf dieser Höhe hält. Auf diese Problematik gehen wir im Abschnitt ‘Aufgetretene Probleme’ genauer ein.

Eine weitere Einschränkung ergibt sich aus der Tatsache, dass die im iPhone verbauten Sensoren keinen wissenschaftlichen Standards entsprechen. Schon in der ersten Gruppenübung stellten wir fest, dass die aus dem Gyroskop ausgelesenen Werte selbst in Ruhelage erheblich schwanken. Wenn man nun bedenkt, dass der User vielleicht auch nicht das ruhigste Händchen hat, wird einem schnell klar, dass das Ergebnis nur ein Näherungswert ist. Ab einer gewissen Entfernung kann das Ergebnis sogar als unbrauchbar betrachtet werden.

Umsetzung

Verwendete Ressourcen und deren Implementierung

CoreMotion Framework

Zu den wichtigsten Frameworks unserer App zählt zweifelsohne das CoreMotion Framework. Es enthält Klassen wie CMAccelerometerData, CMGyroData, CMMagnetometer und noch einige mehr und bildet damit die Schnittstelle zu den Bewegungssensoren. Darüber hinaus beinhaltet es den den CMMotionManager, der den Zugang zu den eben erwähnten Klassen ermöglicht.

```
CMMotionManager *motionManager;
```

Um den Winkel des iPhones im Verhältnis zur Schwerkraft zu ermitteln, könnte man nun die Werte des Gyroskops (Winkel) mit den Werten des Accelerometers (Schwerkraft) abgleichen und die Haltung ermitteln. Das CoreMotion Framework bietet Programmierern hierfür jedoch eine Klasse, die dies bereits übernimmt - CMDeviceMotion:

```
CMDeviceMotion *motion = motionManager.deviceMotion;
```

CMDeviceMotion enthält als Properties unter anderem *attitude*, was einer Instanz der Klasse CMAttitude entspricht.

```
CMAttitude *attitude;
attitude = motion.attitude;
```

CMAttitude wiederum stellt uns als Property *pitch* zur Verfügung was dem Haltungs-Winkel in Radian entspricht und vom Datentyp Double ist. Da wir als Ausgabe und auch für weitere Berechnungen jedoch den Winkel in Grad verwenden möchten, muss dieser noch umgerechnet werden. Diese Berechnung machen wir in einer eigens angelegten Funktion:

```
#define angle(x) (180 * x / M_PI)
double alpha = angle(attitude.pitch);
```

Schlussendlich muss dann nur noch der Tangens des Winkels berechnet und mit der Höhe multipliziert werden

```
angle = tan(alpha * M_PI/180);
result = newHeight*angle;
```

AVFoundation Framework

Damit der User überhaupt einen Gegenstand anpeilen kann, müssen die Werte der rückseitigen Kamera eingelesen und in Echtzeit auf dem Display ausgegeben werden. Um dies zu realisieren, haben wir uns für das AVFoundation Framework entschieden.

Wir beginnen mit der Instanziierung eines Objekts von AVCaptureSession.

```
AVCaptureSession *session;
session = [[AVCaptureSession alloc] init];
```

Im nächsten Schritt erstellen wir Objekte von AVCaptureDevice und AVDeviceInput. AVCaptureDevice repräsentiert dabei das “Gerät”, also die Hardware (hier von Typ AVMediaTypeVideo, also Video). AVCaptureDeviceInput hingegen (als Sub-Klasse von AVCaptureDevice) ist der Stream und wird verwendet um die eingelesenen Daten “festzuhalten”.

```
AVCaptureDevice *inputDevice = [AVCaptureDevice
defaultDeviceWithMediaType:AVMediaTypeVideo];
AVCaptureDeviceInput *deviceInput = [AVCaptureDeviceInput
deviceInputWithDevice:inputDevice error:&error];
```

Das Objekt von Typ AVDeviceInput wird nur der AVCaptureSession als Argument von *addInput* übergeben. Vorher wird überprüft, ob eine Übergabe möglich ist:

```
if ([session canAddInput:deviceInput]){
        [session addInput:deviceInput];
}
```

Die eingelesenen Daten der Kamera werden nun also an unsere Session übergeben. Damit sie auch wieder ausgegeben werden, instanziieren wir nun ein Objekt der Klasse AVCaptureVideoPreviewLayer und übergeben das vorher erzeugten Objekt von AVCaptureSession:

```
AVCaptureVideoPreviewLayer *previewLayer
=[[AVCaptureVideoPreviewLayer alloc] initWithSession:session];
```

Im nächsten Schritt geht es darum eine Fläche zu erzeugen, auf dem die Daten ausgegeben werden. Hierzu erzeugen wir eine Instanz von CALayer und CGRect (*frameForCapture* ist Property von Typ UIView):

```
CALayer *rootLayer = [[self view] layer];
CGRect frame = self.frameForCapture.frame;
```

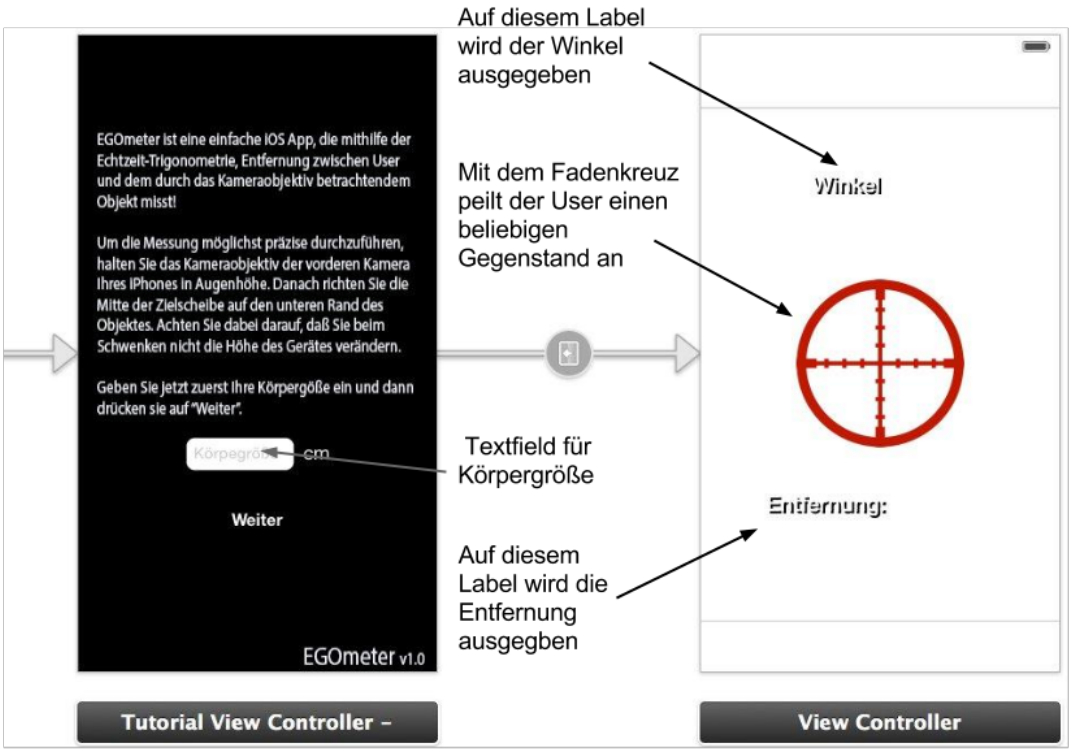
Dem vorher erzeugen Objekt von AVCaptureVideoPreviewLayer (*previewLayer*) weisen wir nun das eben erzeugte Objekt von CGRect (*frame*) zu, welches wir dann wiederum dem Objekt von CALayer (*rootLayer*) als Sublayer zuweisen.

```
[previewLayer setFrame:frame];
[rootLayer insertSublayer:previewLayer atIndex:0];
```

Im letzten Schrit erzeugen wir ein Objekt von AVCaputreStillImageOutput, welches für die Ausgabe der Metadaten verantwortlich ist. Dem Objekt von AVCaptureSession (*session*) weisen wir nun als Ausgabe dieses Objekt zu und starten die “Session”:

```
AVCaptureStillImageOutput *stillImageOutput;
stillImageOutput = [[AVCaptureStillImageOutput alloc] init];
[session addOutput:stillImageOutput];
[session startRunning];
```

GUI



Aufgetretene Probleme

Github

Schon von Anfang an hatten wir Probleme mit der Verwendung von Github. So kam es immer wieder zu Konflikten bei Commits und Pulls. Lange Zeit war dies wohl auf Versionsunterschiede von Xcode zurückzuführen, doch auch als wir beide auf derselben Version gearbeitet haben, kam es zu Problemen.
Die Lösung dieses Problems ist so einfach wie wirksam. Wir haben im Vorfeld Aufgaben abgeklärt, diese dann separat bearbeitet und uns für die Implementierung persönlich getroffen.

Kalibibrierung des Fadenkreuzes

Nachdem wir einen Fehler in der Berechnung gefunden und korrigiert haben, mussten wir feststellen, dass die ausgegebenen Werte noch immer fehlerhaft waren. Uns fiel dann ein, dass wir das Fadenkreuz beim erstellen des Storyboards “einfach irgendwo” platziert haben. Um das Fadenkreuz zu kalibrieren, haben wir auf einer vorher ausgemessenen Strecke die App angewandt und geschaut, wo sich das Fadenkreuz hätte befinden müssen. Nachdem wir die Position dann im Storyboard Editor angepasst haben, stimmten die Werte.

Höhe des Smartphones

Um die Distanz korrekt berechnen zu können, müsste der User die Entfernung zwischen Smartphone und Boden eingeben. Da diese aber in den meisten Fällen nicht ohne Weiteres genau ermittelt werden kann, haben wir uns für einen Workaround entschieden:
Wir weisen den User an, seine Körpergröße einzugeben. Darüber hinaus wird er darauf hingewiesen, dass er für die korrekte Nutzung die Frontkamera auf Augenhöhe halten soll. Bei der späteren Berechnung subtrahieren wir dann von der eingegebenen Körpergröße pauschal 10 cm und ermitteln so einen Näherungswert für die Höhe des Smartphones.

Fazit

Erfahrungen

Insgesamt haben wir viel Freude bei der Durchführung unseres Projektes gehabt. Die gute Dokumentation seitens Apple erleichtert einem die Arbeit ungemein. So haben wir beispielsweise fast ausschließlich *developer.apple.com* als Quelle für die notwendigen Informationen genutzt. Darüber hinaus ist Xcode eine leicht zu verstehende und trotzdem sehr mächtige Programmierumgebung mit der das Coden viel Spaß macht. Einziger Wehrmutstropfen war die Arbeit mit Github, die uns des Öfteren Probleme bereitete. Hier würden wir bei einem weiteren Projekt ggf. nach Alternativen suchen.

Ausblick

Wir planen derzeit die Implementierung einer weiteren Funktion, die nicht nur die Entfernung, sondern auch die Höhe von Gegenständen misst. Dazu dient die Distanzmessung als Grundlage. Die gemessene Entfernung dient bei der Höhenmessung als Wert für die Gegenkathete. Der User muss dann den höchsten Punkt des Objekts anpeilen und ermittelt somit einen Winkel β. Nun kann mit dem Tangens von β und der Gegenkathete wieder die Ankathete ermittelt werden, mit dem Unterschied, dass sie in diesem Fall eben die Höhe des Objektes darstellt.

Das Ergebnis / Screenshot

Abschließend kann man sagen, dass wir mit dem Ergebnis voll und ganz zufrieden sind. Bei Entfernungen bis zu 10m liefert die App Ergebnisse, die auf den Dezimeter genau sind. Bei Entfernungen von unter einem Meter sogar auf den Zentimeter genau. Voraussetzung dafür ist allerdings, dass der User ein ruhiges Händchen hat und das Smartphone auf der richtigen Höhe hält.