# Flappy Bird und ein Kalman Filter für den HC-SR04

Physical Computing WS 18/19
Prof. Dr. Torsten Edeler
HAW-Hamburg, Medientechnik

HAW HAMBURG

Raoul Bickmann 2217470, Ninja Rüsch 2203425

# Inhalt

# 1. Einleitung

## 1.1 Idee + Ziel

Die Idee des Projektes war es das Spiel “Flappy Bird” nachzubauen, als Steuerelement jedoch einen Entfernungssensor zu verwenden. Bei “Flappy Bird” muss der Spieler eine Spielfigur auf dem Bildschirm hoch und runter bewegen und so Hindernissen ausweichen.

In unserer Implementation soll der Entfernungssensor die Entfernung zur Hand des Spielers messen. So ist es dem Spieler dann möglich anstatt durch Drücken eines Buttons, die Spielfigur durch die Bewegung seiner Hand hoch und runter zu bewegen.

## 1.2 Planung

Als Ultraschallsensor soll ein HC-SR04 Ultraschall Modul verwendet werden, welches an einen Arduino Uno angeschlossen wird. Dieser wird über eine USB-Schnittstelle mit einem Computer verbunden und an diesen die Messdaten senden (Vgl. Abb. 1).

Das Spiel “Flappy Bird” soll in 2d in Unity umgesetzt werden. Dafür werden Hindernisse, die sich von rechts nach links über den Bildschirm bewegen, programmiert.

Die Messdaten können vom Arduino ausgelesen und damit die Spielfigur gesteuert werden, um diesen auszuweichen. Da der Ultraschallsensor jedoch verrauschte Daten liefert, soll zusätzlich noch ein Kalman Filter implementiert werden. Hierdurch werden etwaige Fehler des Sensors ausgebessert und somit eine genauere Steuerung ermöglicht.

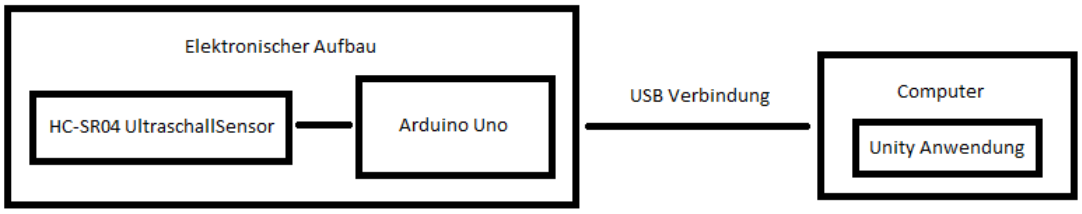


*Abbildung 1: Technischer Aufbau*

# 2. Umsetzung

## 2.1 Arduino

Das HC-SR04 Ultraschall Modul wird wie in Abbildung 2 gezeigt mit dem Arduino Uno angeschlossen. Dabei ist das Modul an die 5V Stromversorgung angebunden und der Triggereingang an Pin 7, sowie der Echoausgang an Pin 8 des Arduinos angeschlossen.

Das Ultraschall Modul misst alle 20ms die Entfernung des Sensors zur Hand des Spielers, indem der Triggereingang für 10µs auf 5V gesetzt wird. Anschließend wird auf den Empfang des Echos gewartet. Wird ein solches erkannt wird der Echoausgang des Moduls für 200ms auf 5V gesetzt. So kann aus der Zeit, die seit dem Senden des Signals vergangen ist, der Abstand bestimmt werden.

Der gemessene Abstand wird anschließend über den USB Anschluss des Arduino an den angeschlossenen Computer und damit an Unity gesendet.

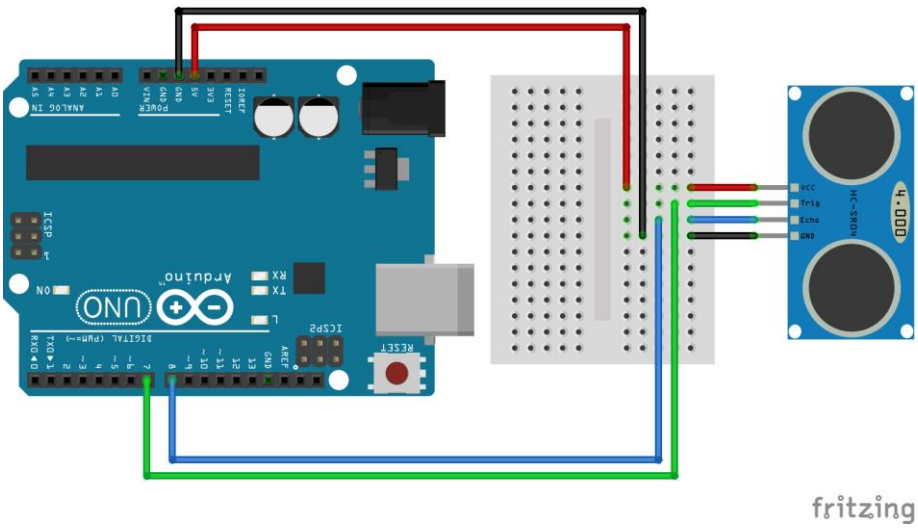


*Abbildung 2: Schaltplan Arduino + HC-SR04*

## 2.2 Unity

Da der Fokus auf der Implementierung des Kalman Filters lag, wurden für das Spiel nur die grundlegenden Funktionen entwickelt. So spawnt jede Sekunde ein neues Hindernis an der rechten Seite des Bildschirms mit einer Öffnung für die Spielfigur und bewegt sich kontinuierlich nach links. Berührt diese ein solches, so ist das Spiel verloren. Auf das Erstellen von Grafiken und Animationen für Spielfigur, Hindernisse und Hintergrund wurde verzichtet.

Für das Empfangen der Messdaten zur Steuerung der Spielfigur wurde die Library Ardity [1]verwendet. Ardity bietet einige einfach einzubindende Skripte, um über eine Serielle Schnittstelle mit einem verbundenen Gerät zu kommunizieren.
Wird eine neue Messung empfangen, so wird diese an den Kalman Filter weitergeleitet. Dessen Design, sowie Implementierung ist im folgenden Kapitel beschrieben. Anschließend wird der vom Filter zurückgegebene Wert verwendet, um den Y-Wert der Spielfigur zu setzen und so deren Position an die Höhe der Hand des Spielers anzupassen.

[1] https://ardity.dwilches.com/

## 2.3 Kalman Filter

### 2.3.1 Kalman Filter Design

Für die Anwendung des Kalman Filters gehen wir vereinfacht von einer konstanten Beschleunigung der Hand des Spielers aus. Dann können wir das Constant-Acceleration-Model für den Kalman Filter nutzen. Daraus ergeben sich die Matrizen für den Filter wie folgt:

Der Systemzustand $\boldsymbol{x(t)}$ besteht aus der Distanz, der Geschwindigkeit und der Beschleunigung:

$$x(t) = \begin{pmatrix} s \\ v \\ a \end{pmatrix}$$

Da nur die Distanz gemessen wird, ergibt sich die Messmatrix $C$ wie folgt:

$$s(t) = C * x(t) = C * \begin{pmatrix} s(t) \\ v(t) \\ a(t) \end{pmatrix} \qquad C = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Für die Systemdynamik Matrix $\boldsymbol{A}$ gilt:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Hieraus kann $\boldsymbol{A_d}$ berechnet werden:

$$A_{d} = e^{A \cdot T_S}$$

$$= I_3 + \frac{A \cdot T_s}{1} + \frac{(A \cdot T_s)^2}{2}$$

$$= \begin{pmatrix} 1 & T_s & \frac{{T_s}^2}{2} \\ 0 & 1 & T_s \\ 0 & 0 & 1 \end{pmatrix}$$

Aus dem Constant-Acceleration-Model ergibt für sich für die System-Rauschmatrix $\boldsymbol{G_d}$ mit

$$G = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

folgendes:

$$G_d = \int_0^{Ts} e^{A \cdot \tau} \cdot G \, d\tau$$

$$= \int_0^{Ts} \begin{pmatrix} 1 & \tau & \frac{\tau^2}{2} \\ 0 & 1 & \tau \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} d\tau$$

$$= \int_0^{Ts} \begin{pmatrix} \frac{\tau^2}{2} \\ \tau \\ 1 \end{pmatrix} d\tau$$

$$G_d = \begin{pmatrix} \frac{T_s^3}{6} \\ \frac{T_s^2}{2} \\ T_s \end{pmatrix}$$

Es soll keinen Control-Input geben, sodass $\boldsymbol{B} = \boldsymbol{0}$ und $\boldsymbol{D} = \boldsymbol{0}$ gilt.

Laut Modulbeschreibung[2] beträgt der Messfehler des Sensor ungefähr 3,4% woraus sich für eine Entfernung von $20\ cm$ eine Ungenauigkeit von $0{,}68\ cm$ ergibt. Deshalb wurde das Sensorrauschen $\boldsymbol{R}$ auf 0,68 festgesetzt.

Die Kovarianz des Prozessrauschens $\boldsymbol{Q}$ wurde anschließend experimentell auf $\mathbf{100000}\ \frac{\boldsymbol{cm}}{\boldsymbol{s^3}}$ festgelegt. Durch dieses Verhältnis von $\boldsymbol{Q}$ und $\boldsymbol{R}$ hält sich der Filter sehr stark an die Sensordaten.

### 2.3.2 Kalman Filter Anpassung

Da der Sensor hin und wieder keine Daten (0) oder Falsche, zu hohe Daten sendet, musste zusätzlich ein Weg gefunden werden, den Einfluss dieser Werte zu minimieren. Um dies zu erreichen wurde eine zusätzliche Mechanik in den Filter eingefügt, die das Sensorrauschen drastisch erhöht wenn der Messwert stark von der Vorhersage abweicht, wodurch der Ausreißer als sehr unsicher eingestuft wird und sich so komplett auf die Vorhersage des Filters

[2] https://www.mikrocontroller.net/attachment/218122/HC-SR04_ultraschallmodul_beschreibung_3.pdf

verlassen wird. Hierdurch tritt erst ein Problem auf, wenn viele falsche Werte hintereinander empfangen werden.

### 2.3.3 Kalman Filter Evaluierung

Für die Evaluierung des Filters wurden Simulationsdaten erzeugt und der Filter auf diese angewendet. Die Simulation wurde mit 50 Werten pro Sekunde vorgenommen, so viele wie auch der Arduino an Unity in der fertigen Anwendung sendet. Für die Werte der Beschleunigung wurden dabei ausgedachte Werte, die gaußverteilt verrauscht wurden, verwendet.
Das Filterergebnis, sowie die Differenz zwischen den gefilterten und den echten Daten, ist in Abbildung 3 zu sehen.

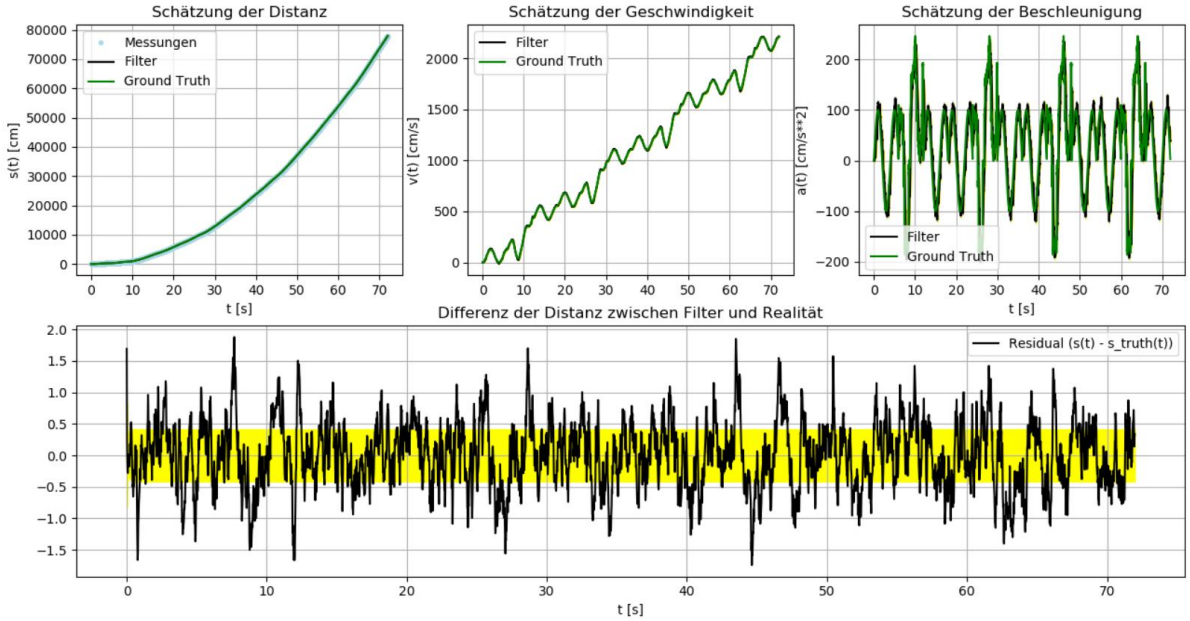


*Abbildung: 3: Ergebnisdarstellung der Simulationsdaten*

Das Filterergebnis der simulierten Daten zeigt, dass der Zustand des Filters meist innerhalb der erwarteten Standardabweichung bleibt. An den Stellen, an denen sich die Beschleunigung rapide ändert, und damit vom Constant-Acceleration-Model abweicht gibt es größere Ausschläge. Diese führen jedoch nicht dazu, dass der Filterzustand komplett von der Realität abdriftet. Stattdessen pendelt er sich wieder ein sobald die Beschleunigung wieder besser zu dem angenommenen Modell passt.

Für die Auswertung der echten Daten, die durch den Sensor reinkommen, war ein weiterer Zwischenschritt notwendig. Um die tatsächlichen Werte zu bestimmen, wurde der User Input parallel mit einer Kamera aufgenommen und später mittels openCV ein Tracking des Input Gegenstandes (Pappkarton) eingefügt.
Durch das Tracking des grünen Bereiches (Vgl. Abb. 4) konnte der genaue Abstand zum Sensor aufgezeichnet werden. Hierbei wurde immer an der unteren Kante gemessen, da dort auch der Ultraschall des Sensors abprallt.
Ca. 30 Sekunden Auf- und Abbewegungen haben wir aufgenommen und analysiert. Fehldaten, die entweder null oder extrem hoch waren, haben wir vor der Analyse herausgefiltert.
Nun konnten die nahezu genauen echten Daten mit den Daten des Filters verglichen werden.

*Abbildung 4:*
*Aufnahme der echten Daten*

Das Diagramm (Abb. 5) zeigt die Gegenüberstellung der Daten aus dem Kalman Filter und der echten Daten, die wir mit OpenCV aus dem Video gewonnen haben.

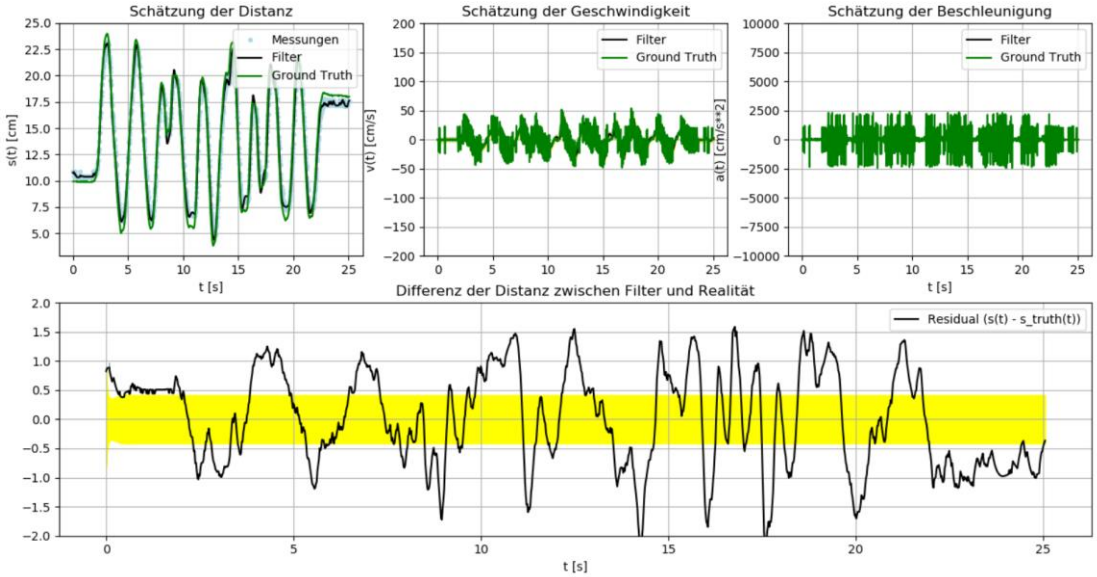


*Abbildung 5: Gegenüberstellung der echten Daten und der gefilterten Daten*

Hierbei ist zu erkennen, dass wie bei der durchgeführten Simulation, die Daten zwar abweichen, jedoch nicht komplett wegdriften.
Zusätzlich wurde die Differenz der echten Daten, jeweils mit den Sensordaten und den gefilterten Daten gebildet. Hieraus ergab sich, dass die Standardabweichung der Differenz der echten Daten und der Sensors bei ca. $0{,}414\ cm$ und bei den echten Daten und dem Kalman Filter bei ca. $0{,}287\ cm$liegt. Daraus ergibt sich eine Verbesserung der Genauigkeit von ca. $30{,}5\%$, gegenüber der Verwendung des Sensors ohne einen Filter.

Die Beurteilung, ob der Filter gut oder schlecht ist, hängt immer ganz von den Erwartungen oder in unserem Fall der Anwendung ab. Die Anforderung an den Filter ist eine präzise Steuerung der Spielfigur, um möglichst lang am Leben zu bleiben. Passiert der Spieler ein Hindernis, darf der Sensor mit oder ohne Filter nicht ausschlagen, da der Spieler sonst verliert.

Demnach sind die Vorteile des Kalman Filters sind zum einen die Glättung der Sensordaten, bei denen kleine Sprünge zu einer fließenden Bewegung werden und zum anderen der Einsatz in Echtzeitanwendungen, wie z.B. Flappy Bird, da es sich um einen iterativen Prozess handelt. Außerdem
Ein Nachteil des Filters ist der Umgang mit Extremen, wenn man keinen Pre-Filter einbaut, pendelt sich der Filter nach einem sehr hohen oder sehr niedrigen Wert zu langsam wieder ein, sodass es für eine Anwendung wie Flappy Bird fatal ist, den Filter zu nutzen. Die Extrema werden zwar entsprechend reduziert und geglättet, jedoch rechnet der Spieler nicht mit einem Output wie diesem bei einer intuitiven Steuerung des Objektes.

### 2.3.3 Implementierung des Filters in Unity

Für die Einbindung des Filters in Unity wurde eine Kalman Filter Implementation von github.com[3] verwendet. Diese bietet bereits eine Implementation von Matrizen und des Kalman Filters in C#. Hierdurch müssen lediglich die in Kapitel 2.3.1 beschriebenen Matrizen angelegt, sowie der Startzustand definiert werden. Anschließend kann dem Filter durch einen einfachen Methodenaufruf ein neuer Messwert übergeben und so der Zustand des Filters und damit auch die Position der Spielfigur geupdated werden.

# 3. Fazit

Schlussendlich lässt sich sagen, dass der HC-SR04 mit dem Kalman Filter eine sehr flüssige Bewegung der Auf- und Abbewegung ergibt. Für eine Echtzeitanwendung, die es in unserem Fall ist, benötigt der Filter jedoch eine Ergänzung, die die extremen Werte ignoriert. Bei großen Ausschlägen pendelt der Filter zu stark aus, sodass der Versatz zu den Echtzeitdaten zu hoch ist. Dadurch ginge der flüssige Spielfluss verloren und der Anwender wäre zusätzlich irritiert.
Außerdem liefert der Sensor mit Ausnahme der fehlerhaften Daten sehr genaue Daten, sodass man den Kalman Filter für unsere Anwendung Flappy Bird nicht zwingend braucht. Jedoch ist die flüssige Bewegung optisch deutlich angenehmer als das Zucken ohne den Filter. Hierfür könnte man jedoch einen anderen Filter verwenden und braucht nicht zwingend den Kalman Filter.

# 4. Quellenverzeichnis

Arduino, Python und Unity Code befindet sich im folgenden Repository:
Github: https://github.com/RaoulBickmann/FlappyBird

[3] https://github.com/asus4/UnityIMU/tree/master/KalmanSample/Assets/Plugins/Kalman