Rauschanalyse für die Höheninformation des DWM1001 im Indoor-Bereich sowie Integration in ein bestehendes RC-Fahrzeug

Axel Schwab
Fakultät Informationstechnik
Hochschule Esslingen
Esslingen, Deutschland
axscit02@hs-esslingen.de

***Abstract*—Die Positionsbestimmung im Indoor-Bereich ermöglicht im Vergleich zu GPS komplett neue Anwendungsszenarien. Hierfür ist Technologie notwendig, welche immer weiter untersucht werden muss, um dessen Systemgrenzen aufzuzeigen. Die Firma Decawave stellt mit dem DWM1001 ein Ultrabreitbandmodul zur Verfügung, welches eine Positionsbestimmung bis zu 10 Zentimetern Genauigkeit erfüllen soll. Wie sich herausstellt, trifft dies nicht für die Höheninformation der Position zu.**

***Index Terms*—DMW1001, Rauschanalyse, Ultrabreitband, Decawave, Kalman-Filter**

I. Einleitung

Es gibt vielerlei Positionsbestimmungssysteme, das bekannteste dürfte GPS (Global Positioning System) sein, welches die Position mit einer Auflösung von wenigen Metern bestimmt [9]. Dies bezieht sich, wie der Name schon einleitet, auf die Position im globalen Kontext, wodurch beispielsweise Navigationssysteme auf der Straße, in der Luft oder auf dem Wasser aufbauen. Aus der Genauigkeit von GPS ergeben sich mögliche Anwendungsfälle. So ist es mittels GPS nicht möglich innerhalb von Gebäuden eine valide Position zu erhalten und damit beispielsweise eine Indoor-Navigation zu ermöglichen. Dieser Problematik hat sich die Firma Decawave angenommen und mit dem DWM1001 ein LPS (Local Positioning System) mit einer Genauigkeit von bis zu 10 Zentimetern auf den Markt gebracht [7].

Diesem Paper geht [8] voraus, mit dem Fokus auf einen Einsatz des DWM1001 im Außenbereich. Dabei wird die Funktionsweise, sowie der Softwarestack und die grundlegende Technologie erläutert. Im Kern behandelt dieses Paper das Rauschverhalten der Höheninformation der Position des DWM1001 im Indoor-Bereich. Hinzu kommt eine Beispielimplementierung, welche die Positionsinformation mittels eines Kalman-Filters glättet und über ein CAN Interface nach außen anbietet.

II. DWM1001

Der folgende Abschnitt geht auf das Prinzip der Positionsbestimmung ein, auf welchem der DWM1001 basiert, sowie auf die Konfigurationsmöglichkeiten des Systems. Weitere Informationen zu dem Softwarestack des DMW1001 sind in [2] und [3] zu finden. Zusätzlich wird in [4] die Systemkonfiguration erläutert.

A. Trilateration

Das Prinzip der Positionsbestimmung welches sich der DWM1001 zunutze macht, basiert auf einem geometrischen Verfahren. Die Trilateration ermöglicht es anhand von Distanzen unterschiedlicher Messpunkte zu demselben Objekt die Position, relativ zu den Messpunkten, anzugeben. Die einzelnen Distanzen können dabei als Radien von Kreisen verstanden werden, die einen gemeinsamen Schnittpunkt haben. Dieses Prinzip funktioniert in der Ebene mit mindestens drei Distanzen. Für die Positionsbestimmung im Dreidimensionalen sind vier Distanzen notwendig. Dabei können sich die drei Kreise, die sich in einem Punkt schneiden, nun wie vier Kugeln im dreidimensionalen Raum vorgestellt werden, welche einen gemeinsamen Schnittpunkt haben. Einzige Voraussetzung dafür ist, dass die Höhen der Mittelpunkte dieser Kugeln nicht in einer Ebene liegen, falls nicht, ergeben sich zwei Schnittpunkte und somit zwei Positionen, die sich in der Höhe unterscheiden. Das bedeutet, mindestens ein Kugelursprung muss tiefer oder höher liegen als die anderen. Die Abbildung 1 sowie Abbildung 2 zeigen das Prinzip der Trilateration in der Ebene sowie im dreidimensionalen.

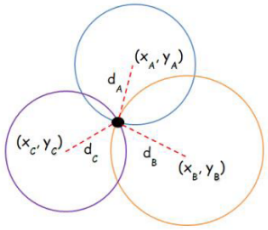

Fig. 1. Trilateration im zweidimensionalen [6]

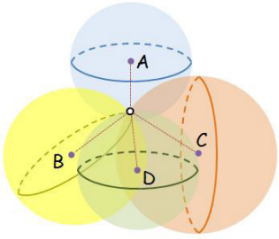

Fig. 2. Trilateration im dreidimensionalen [6]

Um die Position anhand der Distanzen im Dreidimensionalen zu bestimmen, muss im naiven Fall lediglich das Gleichungssystem nach [6] gelöst werden.

$$d_A^2 = (x - x_A)^2 + (y - y_A)^2 + (z - z_A)^2 \quad (1)$$
$$d_B^2 = (x - x_B)^2 + (y - y_B)^2 + (z - z_B)^2 \quad (2)$$
$$d_C^2 = (x - x_C)^2 + (y - y_C)^2 + (z - z_C)^2 \quad (3)$$
$$d_D^2 = (x - x_D)^2 + (y - y_D)^2 + (z - z_D)^2 \quad (4)$$

Der DMW1001 bietet, entsprechend dem Prinzip der Trilateration, zwei Modi an, um eine Positionsbestimmung durchzuführen.

B. Anchor

Die Anchor entsprechen den Kreismittelpunkten, und diese bilden somit fixe Punkte im Raum und spannen ein Koordinatensystem auf, welches als Referenz für die Position des Tags dient.

C. Tag

Der Tag bildet die Einheit, dessen Position es zu bestimmen gilt. Dafür misst dieser die Distanz zu den einzelnen Anchorn und kann anhand dessen Position im Raum, sowie Distanzen seiner Position im Koordinatensystem bestimmen. Im schnellsten Fall kann der Tag alle 100 Millisekunden seine Position bestimmen [7].

III. Messungen

Um eine Messung durchzuführen ist im ersten Schritt eine Systemkonfiguration erforderlich, bei der die Anchor ihre Position erhalten. Erst dann kann eine Messung durchgeführt werden.

A. Messaufbau

Das System wurde im Labor für Embedded Systems in der Hochschule Esslingen aufgebaut und dadurch ergibt sich eine kontrollierbare Umgebung. Die Abbildung 3 zeigt dies.

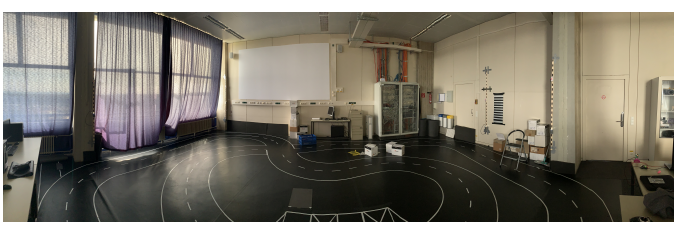

Fig. 3. Labor für Embedded Systems

Die Anbringung der Anchor im Raum ist die wesentliche Herausforderung des Systemaufbaus. Die Qualität der Position des Tags hängt von zwei Faktoren ab. Erstens müssen die Anchor im Raum richtig positioniert sein. Es gibt keine genaue Regel für eine *richtige* Anbringung, sondern nur ein grobes Vorgehen. Die Anchor dürfen nicht in der gleichen Höhe angebracht werden, mindestens einer von vier muss tiefer oder höher sein. Die Antenne der Module muss nach oben zeigen und es darf kein Metall in einem Radius von 3 cm um die Antenne herum vorhanden sein. Der zweite Faktor bezieht sich auf das Setzen der richtigen Systemkoordinaten der Anchors. Wenn diese zu große Abweichungen aufweisen, wird eine verfälschte Position ausgegeben oder das System liefert gar keine Positionsinformation [5].

Für den Tag wurde ein Adapter in dem 3D-Drucker angefertigt, damit dieser auf ein Stativ montiert werden kann und Ausrichtung sowie Höhe angepasst werden können. Nach der Montage der Anchors und dem Aufstellen des Tags muss eine Systemkonfiguration durchgeführt werden. Die Anchors müssen ihre Position in Form von (X, Y, Z) Koordinaten zugewiesen bekommen. Die Konfiguration wird aufgrund der drei Bettonsäulen, wie in Abbildung 3 zu sehen ist, erleichtert, da diese ein rechtwinkliges Dreieck aufspannen, wodurch nur die Distanz zwischen Anchor A zu D, sowie A zu B ausgemessen werden muss sowie die Höhen der Anchor. Hierfür wurde ein Laser-Messgerät verwendet. Der Anchor A erhält die Position $(0, 0, h_A)$, er definiert den Koordinatenursprung. Der Anchor B liegt auf der X-Achse und ist genau die Distanz AB von A entfernt, somit ist seine Position $(\overline{AB}, 0, h_B)$. Genau gleich verfährt man mit Anchor D, dieser liegt auf der Y-Achse. Seine Position ergibt sich zu $(0, \overline{AD}, h_D)$. Die Position des Anchor C erhält man, ohne zu messen von dem System selbst. Dafür wird dieser temporär als Tag konfiguriert, um seine Position im Zweidimensionalen zu erhalten. Diese wird dann noch um seine Höhe ergänzt und man erhält die finale Position des Anchors D[1]. Der finale Aufbau des Messystems wird in Abbildung 4 gezeigt.

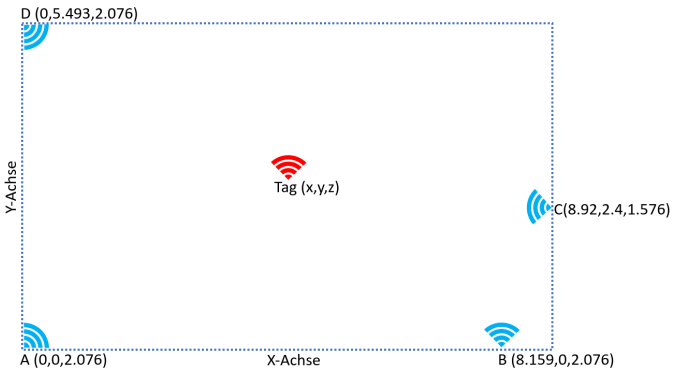

Fig. 4. Aufbau des Gesamtsystems

B. Messdurchführung

Für die Messungen wurden fünf Messpunkte im Systemraum definiert. Die ungelöste Problematik hierbei ist, dass die tatsächliche Position der Messpunkte unbekannt ist. Versuche, diese manuell auszumessen, waren zu fehlerbehaftet, und dadurch ist eine Abweichungsbestimmung

[1]Die Position des Anchors D wurde durch manuelles Ausmessen überprüft, um die Position in seiner Validität zu bestätigen.

zwischen tatsächlicher Position und gemessener nicht möglich, lediglich die gemessene Höhe kann Überprüft werden.

Es wurde pro Messpunkt in drei unterschiedlichen Höhen gemessen. Zum einen in 0.115 m Höhe, dies entspricht der ungefähren Höhe, die der Tag auf dem autonom fahrenden Auto des Labors hatte. Die zweite Höhe ist 0.9 m, dies entspricht der mittleren Hosentaschen höhe eines Menschen mit einer Körpergröße von 1.8 m. Die dritte Messhöhe ist 1.2 m, dies entspricht ungefähr der Höhe, in welcher Smartphones in den Händen gehalten werden bei einer stehenden Person mit einer Körpergröße von 1.8 m.

Zusätzlich zu den 5 Messpunkten mit 3 unterschiedlichen Höhen wurde eine weitere Messoption hinzugefügt. In [8] konnte beobachtet werden, dass sich anhand der Orientierung des Tags zu den Anchorn die Position einen Offset erfährt. Aus diesem Grund wird pro Messpunkt für jede Höhe mit vier Orientierungen gemessen. Die Abbildung 5 zeigt dies anschaulich.

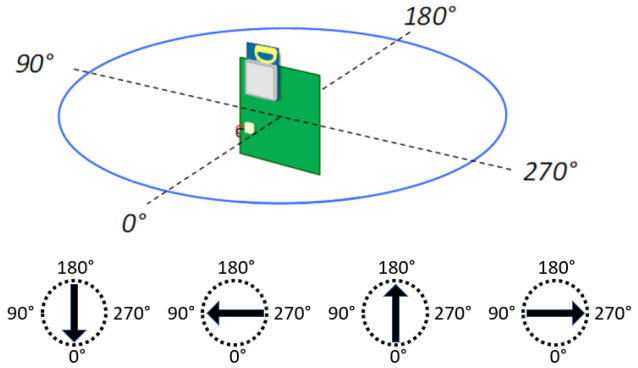

Fig. 5. Orientierung des Tags

Da das System in seltenen Fällen dazu neigt, invalide Daten zu produzieren (als Position wird konstant (0,0,0) ausgegeben) werden 1300 Positionsdaten pro Messung gesammelt und später auf 1000 valide Daten reduziert, damit die Daten untereinander vergleichbar sind.

Final bedeutet dies für die Messdurchführung: Es wird an fünf Messpunkten im Raum in drei unterschiedlichen Höhen und für jede Höhe 4 Orientierungen gemessen. In der Summe macht dies $4\,\text{Orientierungen} \cdot 3\,\text{Höhen} \cdot 5\,\text{Messpunkte} = 60\,\text{Messungen}$. Die Abbildung 6 zeigt die Messpunkte im Gesamtsystem.

C. Ergebnis

Es hat sich gezeigt, dass mit zunehmender Höhe das Rauschen ansteigt, die Abbildung 7 verdeutlicht dies. Aus den Daten sticht dabei heraus, dass die Orientierung mit 0° am stärksten ein Rauschen aufweist.

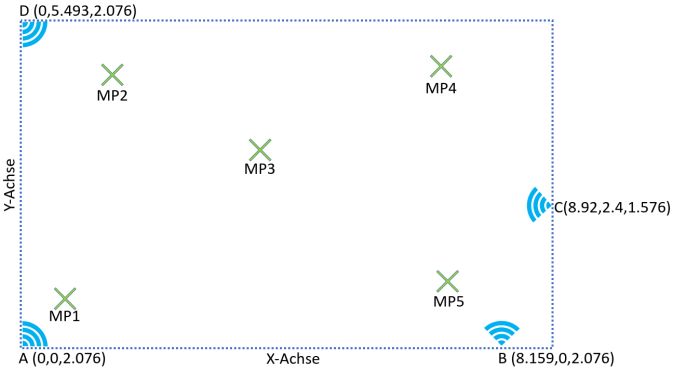

Fig. 6. Gesamtsystems mit Messpunkten

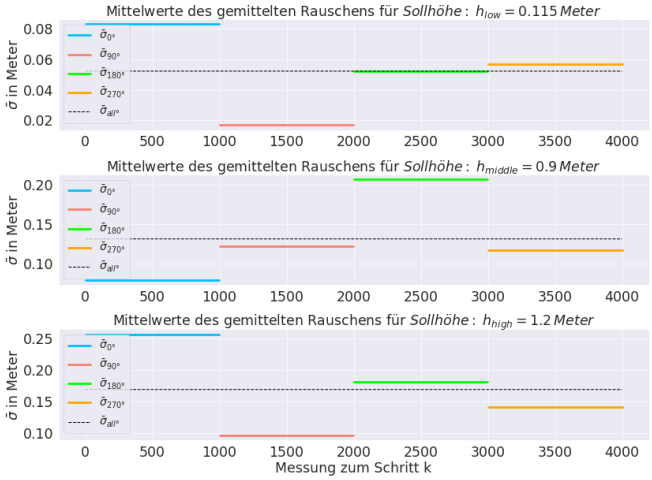

Fig. 7. Anstieg des Rauschens über alle Messpunkte mit zunehmender Messhöhe

Auch der Messfehler steigt mit zunehmender Höhe an, dies wird durch Abbildung 8 deutlich.

Das Phänomen des Anstieges des Rauschens, sowie der zunehmende Messfehler, kann nicht final beantwortet werden. Zum einen hat sich gezeigt, dass die Messungen dann zu rauschen beginnen, sobald sich die Vorhänge (links von Messposition 1 und 2 aus Abbildung 3) bewegt haben. Dabei reagiert das System mit Sprüngen in die Höhe unabhängig der Messhöhe. Es ist auch möglich, dass mit Annäherung des Tag an die Höhe des Anchors C bei 1.576 Metern das System keine eindeutige Höhe zuweisen kann. Schlussendlich lieferte das System nur für die Messhöhe $h_{0.115}$ über alle Messpunkte hinweg nahezu den tatsächlichen Wert.

IV. Implementierung

A. Design

Um an die Positionsinformation aus dem DWM1001 zu gelangen wurde die UART Schnittstelle und der entsprechende Softwarestack, welchen Decawave in C++ für den Raspberry Pi zur Verfügung stellt, auf ein Teensy 4.0 Board geportet. Das Design setzt sich aus drei Modulen zusammen. Aus dem DWM1001 wird die Positionsinformation von einem Teensy

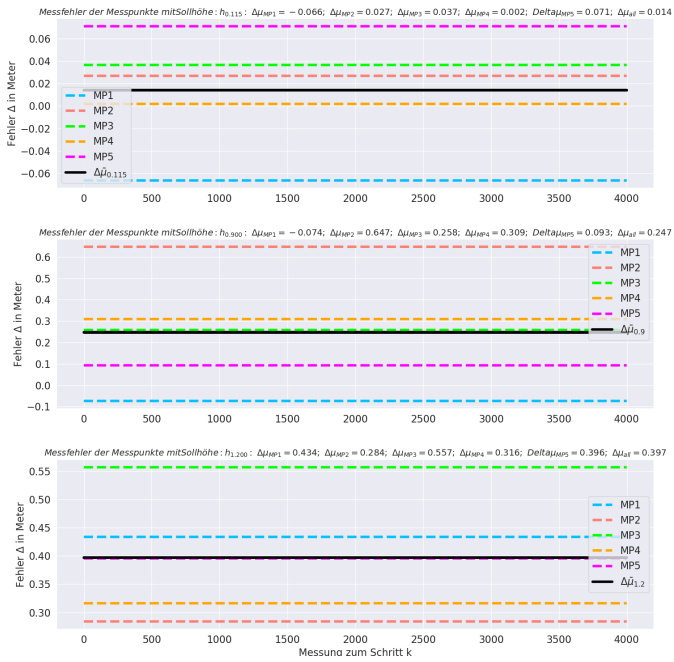

Fig. 8. Anstieg des Messfehlers im Mittel über alle Messpunkte mit zunehmender Messhöhe

4.0 Board über UART durch `polling` abgefragt. Auf dem Teensy selbst befindet sich ein Kalman-Filter, welcher hierfür in C++ implementiert wurde. Es handelt sich dabei um einen Extended-Kalman-Filter, welcher Eigenschaften eines ROSE-Filter aufweist. Dieser Filter wurde zu Beginn der Arbeit von Professor Marchthaler in Matlab zur Verfügung gestellt. Weitere Informationen zu Kalman-Filter und ROSE-Filter sind in [10] zu finden.

Als letztes Element wird über ein `CAN Breakout Board` die gefilterte Position über CAN den Clients zur Verfügung gestellt.

Da es sich bei dem Kalman-Filter um einen zeitdiskreten Filter handelt, setzt dies eine konstante Abtastzeit voraus. Die Hauptschleife des Kalman-Filters muss in einem konstanten Intervall den Filter aktualisieren. Dies wird erzielt, indem der Filter die Zustandsvariablen nur dann verändert, wenn neue Messdaten anliegen. Diese kommen in einem konstanten Intervall an, das um die 100 ms liegt.

B. UART Polling

Wie die Abbildung 9 zeigt wird der Status `new_meassurments`[2] durch Polling ermittelt.

Das Abfragen des Status ist blockierend und benötigt eine gewisse Zeit. Ebenso benötigt das Holen der neuen Positionsinformation auch eine gewisse Zeit. Das bedeutet, dass die Positionsinformation im besten Fall um die Blockierzeit von `new_meassurments` und

[2]Intern wird die Funktion `dmw_status_get` des DWM1001 Stacks aufgerufen.

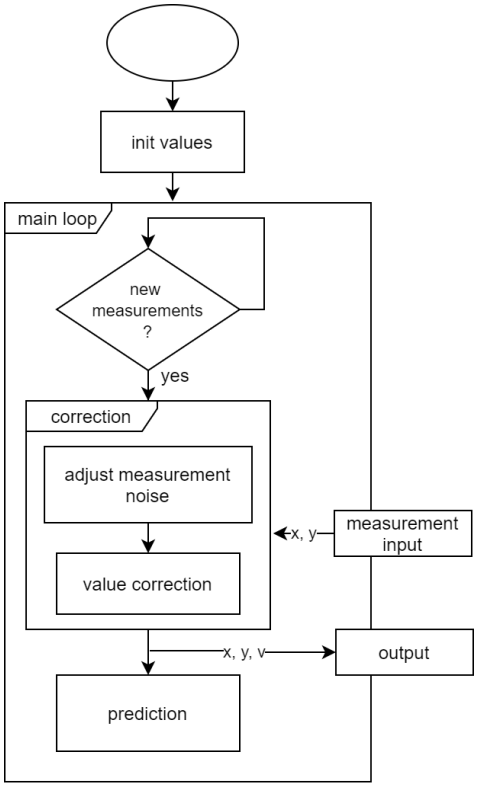

Fig. 9. Schematische Implementation des Kalman Filters

`measurment_input` gealtert ist. Man könnte auch sagen, dass die Position dem System nicht sofort vorliegt, sondern nach eben dieser Zeitspanne.

Diese Zeitspanne, im Folgenden als **Polling Delay** bezeichnet, darf nicht zu groß werden, ansonsten hat sich das RC-Fahrzeug von seiner eigentlichen Position zu weit entfernt. Deswegen wurden die Wartezeiten der zwei Funktionen ermittelt und daraus unterschiedliche Fälle betrachtet, bezüglich des Alters der Positionsinformation und des Zeitpunktes, zu dem sie im System vorliegt.

C. Polling Delay

Die Abbildung 10 zeigt, wie der Polling Delay zustande kommt und welche Fälle auftreten können. Mit dem Einschalten des Systems werden Teensy 4.0 und DWM1001 hochgefahren. Im Folgenden wird die Zeit ab dem ersten Event $E_0 = 0\,ms$ (das erste Vorliegen einer Position) zum Zeitschritt $k = 0$ aus der Perspektive des DWM1001 betrachtet.

Es ist davon auszugehen, dass das Teensy 4.0 Board aufgrund von Initialisierungen und entsprechenden Start Delays nicht zeitgleich, sondern nach einem gewissen *OFFSET*, zum Zeitpunkt X zur Verfügung steht. Von hier an pollt das Teensy 4.0 Board den Status des DWM1001, bis eine neue Position vorliegt. Im Startfall, also zum Zeitpunkt X ergibt sich die erste Polling Delay zu

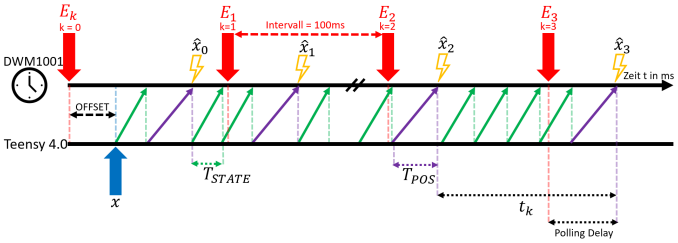

Fig. 10. Polling Delays

$\mathrm{PD}_0 = E_0 + \mathrm{OFFSET} + T_{\mathrm{STATE}} + T_{\mathrm{POS}}$. Dabei ist T_{STATE} die Dauer, um zu erfragen ob eine neue Position vorliegt und T_{POS} die Dauer, um die neue Position in das Teensy Board zu laden.

Ab dem Augenblick, in dem die erste Positionsinformation $\hat{x}_0$ im Teensy Board vorliegt, wird dieses einmal durch den Kalman-Filter durchgeschickt und per CAN versendet. Dieser Prozess dauert weniger als 10 Microsekunden, weshalb dies im Weiteren vernachlässigt wird.

Das Teensy Board wird solange die Funktion `dwm_status_get` aufrufen, bis das Event E_1 eintritt. Die Events treten alle 100 ms ein, da das DWM1001 entsprechend konfiguriert ist.

Anhand des Zeitpunktes $\hat{x}_0$ und dem Event E_1 sowie der Zeitdauer von T_{POS} und T_{STATE} kann $\hat{x}_1$ berechnet werden und daraus auch PD_1. Die Folgenden Gleichungen stellen diesen Sachverhalt dar:

$$E_k = 100 \cdot k \tag{5}$$

$$E_{\mathrm{k+1}} = 100 \cdot (1 + k) \tag{6}$$

$$\lceil \alpha_k \rceil = \frac{E_{\mathrm{k+1}} - \hat{x}_k}{T_{\mathrm{STATE}}} \tag{7}$$

$$t_k = \lceil \alpha_k \rceil \cdot T_{\mathrm{STATE}} + T_{\mathrm{POS}} \tag{8}$$

$$\hat{x}_{k+1} = \hat{x}_k + t_k \tag{9}$$

$$\mathrm{PD}_k = \hat{x}_k - E_k \tag{10}$$

Bei der Gleichung 7 wird die Differenz zwischen dem zukünftig eintretenden Event und dem Zeitpunkt des letzten Eintreffens einer Position gebildet und durch die Zeitdauer des Pollens dividiert und am Ende auf die nächste höhere ganze Zahl aufgerundet. So erhält man die Anzahl an Polling Zyklen, bis das nächste Event eintritt. Diese wird wiederum mit T_{STATE} multipliziert und T_{POS} dazu addiert. Somit ergibt sich die zeitliche Differenz, die Gleichung 8 zeigt. Dieses t_k zu $\hat{x}_k$ addiert führt zu $\hat{x}_{k+1}$. Am Ende lässt sich wieder die Differenz zwischen letztem Event E_k und dem Augenblick des Eintreffens der neuen Position bestimmen, so erhält man den entsprechenden Polling Delay.

Diese Berechnungen können abhängig von dem OFFSET sowie T_{STATE} und T_{POS} durchgeführt werden, um einen Überblick über die Größenordnungen des Polling Delay zu erhalten. Entscheidend für die Berechnung des Polling Delays sind die Werte T_{STATE} und T_{POS}, denn durch diese ergibt sich der Polling Delay. Diese Werte können gemessen werden, indem der Clockcounter Wert vor dem Aufruf der Funktionen `dwm_pos_get` und `dwm_status_get` mit dem Clockcounter-Wert am Ende der beiden Funktionen verglichen wird.

So ergibt sich bei 1000 Messungen für T_{POS} in 32.6 % der Fälle eine Zeit von 3.58 Millisekunden und in 67.4 % der Fälle eine Zeit von 3.785 Millisekunden. Die Abbildung 11 zeigt dies.

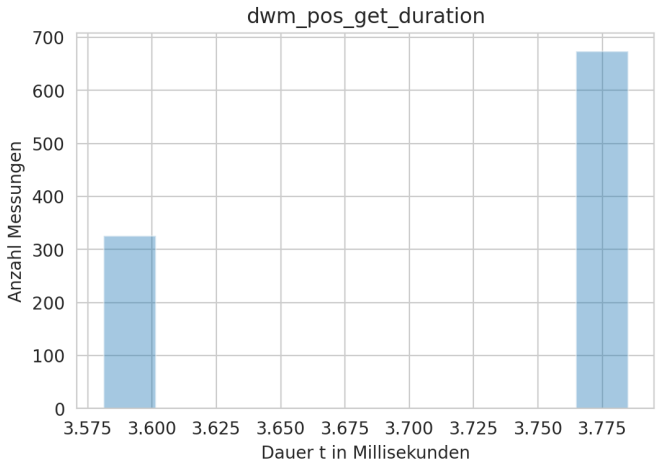

Fig. 11. Blockierzeit der Funktion `dwm_pos_get`

Ebenfalls ergibt sich bei 1000 Messungen für T_{STATE} in 84.3 % der Fälle eine Zeit von ungefähr 2.8 Millisekunden und in 12.3 % der Fälle eine Zeit von ungefähr 2.985 Millisekunden. Zusätzlich liegen noch 2.3 % bei ungefähr 3.195 Millisekunden und die kleinste Gruppe liegt mit 1.1 % bei 3.375 Millisekunden. Die Abbildung 12 zeigt dies.

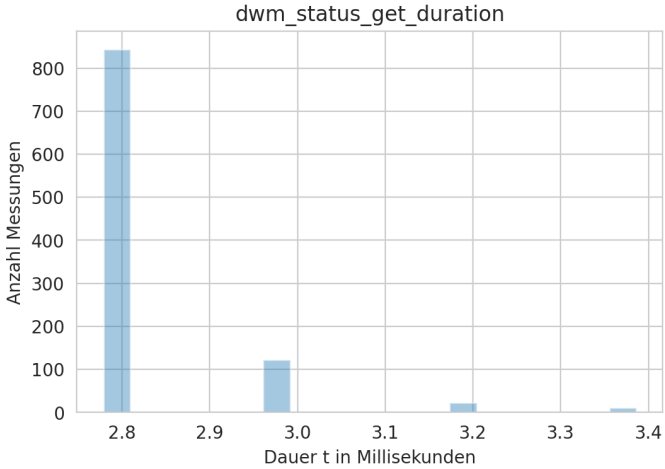

Fig. 12. Blockierzeit der Funktion `dwm_status_get_duration`

Wird der Algorithmus 1 (im Anhang) mit den Werten aus Abbildung 11 und Abbildung 12 gefüttert, ergibt sich die

Polling Delay für einen beliebigen OFFSET im Wertebereich von 1 bis 100 Millisekunden. Dabei werden für jeden OFFSET 1000 Polling Delays berechnet. Somit ergeben sich 100.000 Polling Delays wie die Abbildung 13 zeigt. Der Wertebereich erstreckt sich dabei für den Polling Delay in einem Bereich von 4 bis 6.5 Millisekunden.

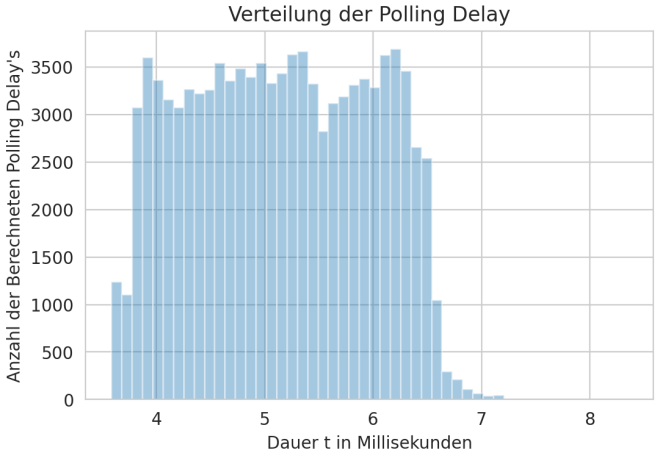

Fig. 13. Analytische Berechnung der Polling Delay

D. Resultat

Unter der Annahme, dass sich das RC-Fahrzeug mit einer maximalen Geschwindigkeit $v_{\max} = 20\,\text{km/h}$ bewegt, was $5.56\,\text{m/s}$ entspricht, hat sich das Fahrzeug von der eigentlichen Position im schlimmsten Fall um maximal $7\,ms \cdot 5.6 m/s = 3.9$ Zentimeter vom Ursprungsort wegbewegt. Diese Abweichung von der eigentlichen Position befindet sich noch innerhalb der Toleranz des Systems von ungefähr 10 cm nach Herstellerangaben [7].

Im weiteren Verlauf gilt es den DWM1001 Tag auf einem RC-Fahrzeug zu implementieren und die Daten aus dem Kalman-Filter zu überprüfen, um diesen feiner einzustellen.

Danksagung

Mein Dank gilt Herrn Professor Marchthaler für die Bereitstellung eines Kalman-Filter Modells in Matlab und Herrn Alexander Huebener, der mich bei der Portierung des DWM1001 Stacks und Kalman-Filter Models auf das Teensy Board massgeblich unterstützt hat.

References

[1] Decawave Ltd., DWM1001 System Overview And Performance, Version 2.0, 2018.
[2] Decawave Ltd. DWM1001 Firmware Application Programming Interface (API) GUI- DE. Version 2.2. 2019.
[3] Decawave Ltd. DWM1001 Firmware User Guide. Version 1.3. 2017.
[4] Decawave Ltd. DWM1001 Firmware User Guide. Version 1.3. 2017.
[5] Decawave Ltd. MDEK1001 Quick Start Guide. Version 2.0. 2018.
[6] Yeong-Sheng Chen, Der-Jiunn Deng und Chun-Chien Teng. Range-Based Localiza- tion Algorithm for Next Generation Wireless Networks Using Radical Centers“. In: IEEE Access 4 (Jan. 2016), S. 1–1. doi: 10.1109/ACCESS.2016.2551704.
[7] Decawave Ltd. DWM1001 Datasheet. Version 1.10. 2017.
[8] Schwab Axel. Konzeption und Realisierung eines Systems zur Positionsbestimmung im Außenbereich durch Einsatz von Ultra-Breitband-Technologie gestützt durch die Anwendung eines stochastischen Schätzverfahrens“. Deutsch. Bachelor’s Thesis. Esslingen, Deutschland: Hochschule Esslingen, Fakultät Informationstechnik, 28. Jan. 2019.
[9] Zafer Sahinoglu, Sinan Gezici und Ismail Guvenc. Ultra-Wideband Positioning Sys- tems: Theoretical Limits, Ranging Algorithms, and Protocols. CAMBRIDGE UNIV PR, 11. Juni 2011. 269 S. isbn: 0521187834.
[10] Reiner Marchthaler und Sebastian Dingler. Kalman-Filter: Einführung in die Zustandsschätzung und ihre Anwendung für eingebettete Systeme. Springer Vieweg, 2017.

Appendix

```
#include <stdio.h>
#include "dwm_status_get_duration.h"
#include "dwm_pos_get_duration.h"

int main() {
  int alpha= 0;

  int k = 0;
  int E = 100*(1+k);

  double t_k = 0;

  double x_hat_new = 0; // x_hat_k+1
  double mean = 0;

  double forecast_in_seconds = 10; //
      sekunden in die zukunft
  int  steps = forecast_in_seconds*100;
      // steps, 1 step = 100ms


  for (int OFFSET = 0; OFFSET < 100;
      OFFSET++) {
    x_hat_new = (OFFSET+T_STATE[k]+T_POS[
        k]);
    for (k=0; k < steps; k++) {
      E = 100*(1+k);
       alpha = (E-x_hat_new) / T_STATE[k];
       alpha +=1;
       t_k = alpha * T_STATE[k] + T_POS[k
           ]; // Delta
       x_hat_new += t_k;
       printf("%i;%i;%lf\n",OFFSET,k,
           x_hat_new-E);
      mean += x_hat_new-E;
    }
  }
  return 0;
}
```

Listing 1. Analytische Berechnung des Polling Delay für 10 Sekunden mit einem beliebigen Offset